The University Of Sheffield.

Danica D. Damljanović

Natural Language Interfaces to Conceptual Models

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy at
The University of Sheffield
Department of Computer Science

July 2011

Abstract

Accessing structured data in the form of ontologies currently requires the use of formal query languages (e.g., SeRQL or SPARQL) which pose significant difficulties for non-expert users. One way 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. Additionally, they often require specific adherence to a pre-defined syntax which, in turn, means that users still have to undergo training.

In this thesis, we study the usability of NLIs from two perspectives: that of the developer who is customising the NLI system, and that of the end-user who uses it for querying. We investigate whether usability methods such as feedback and clarification dialogs can increase the usability for end users and reduce the customisation effort for the developers. To that end, we have developed two systems, QuestIO and FREyA, whose design, evaluation and comparison with similar systems form the core of the contribution of this thesis.

Acknowledgements

This work would not have been possible without help and support of many people. I am grateful to my supervisor, Hamish Cunningham, for encouragement, guidance and support. My examiners, Enrico Motta and Mark Stevenson for an interesting discussion during my viva and useful comments that improved this thesis. Kalina Bontcheva and Valentin Tablan for their support, and guidance in the course of the TAO project. The members of the GATE team who were kind to participate in the QuestIO user evaluation. The participants of the First GATE Summer School, who participated in the FREyA user evaluation. Members of the NLP Group in Sheffield, and particularly Mark Hepple, Rob Gaizauskas, John Derick, Louise Guthrie, David Guthrie, Sanaz Jabbari, Kumutha Swampillai and Leon Derczynski for encouragement and useful discussions. Johann Petrak, for useful discussions while he was a visiting researcher in the NLP group, and also for developing Virtual Documents plugin later which became a part of FREyA. Yaoyong Li for answering my questions about statistics. The GOOD OLD AI Network, and all members who are always a great inspiration, and most of all Vladan Devedźić. Ivan Dimitrijević for pointing me to the JIT library which became a part of FREyA. Nicolas Garcia Belmonte for support with the JIT library. Ontotext, and in particular Ivan Peikov, Danail Kozuranov and Atanas Kyriakov, for their support with OWLIM and LKB which are used in QuestIO and FREyA. Vanessa Lopez from The Knowledge Media Institute for giving me access to the AquaLog system. Abraham Bernstein and Esther Kaufmann from the University of Zurich, for sharing with me the Mooney dataset in OWL format, and J. Mooney from University of Texas for making this dataset publicly available. David and Kumutha for reading and correcting English in the parts of this thesis; Milan Agatonović for reading the thesis

draft and suggesting many structural improvements; Hamish and Kalina also read and provided many useful comments to this thesis draft; Sanaz read parts of this thesis and gave useful comments – their suggestions helped me improve this thesis significantly. Chris Moran with whom I had lots of discussions on usability of semantic web technologies, and who triggered many interesting questions in my head that, I believe, made this thesis stronger. My family and friends for their love and understanding. Milan Agatonović for bearing with me through all difficult moments that existed while working on this thesis and helping out with all those *ClassNotFoundExceptions* –this work would not have been possible without him.

I would like to thank European Commission for funding majority of the work in this thesis: the first part of this work has been supported by the EU Sixth Framework Program project TAO (FP6-026460), while the second half has been partially supported by the EU Seventh Framework Program project LarKC (FP7-215535).

I offer my regards and blessings to all those who supported me in any respect during the completion of this thesis and whose name I have not mentioned, including many colleagues I met and had very inspirational discussions during SSMS'07, LREC'08, ESWC'08, ISWC'08, CNL'09, K-CAP'09, LREC'10, ESWC'10, CNL'10, and project meetings in the course of TAO and LarKC.

Publications

The work presented within this thesis is published in the following papers:

- D. Damljanovic, M. Agatonovic, H. Cunningham: *FREyA: an Interactive Way of Querying Linked Data using Natural Language.* In: Proceedings of 1st Workshop on Question Answering over Linked Data (QALD-1), collocated with the 8th Extended Semantic Web Conference (ESWC'11). Heraklion, Greece, June 2011.
- H. Wang, D. Damljanovic, T. Payne, N. Gibbins, and K. Bontcheva: *Transition of Legacy Systems to Semantically Enabled Applications: TAO Method and Tools.* Semantic Web Journal, IOS Press, 2011, to appear. Available from http://www.semantic-web-journal.net/
- H. Wang, D. Damljanovic and J. Sun: *Enhanced Semantic Access to Formal Software Models.* In the Proceedings of the 12th International Conference on Formal Engineering Methods, Shanghai, China, November 16 - 19, 2010.
- D. Damljanovic. *Towards Portable Controlled Natural Languages for Querying Ontologies.* In Rosner, M., Fuchs, N., eds.: Proceedings of the 2nd Workshop on Controlled Natural Language. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, Marettimo Island, Sicily, September 13-15, 2010.
- D. Damljanovic, M. Agatonovic, H. Cunningham: *Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction.* In Proceedings of the 7th Extended Semantic Web Conference (ESWC'10), Springer Verlag, Heraklion, Greece, May 31-June 3, 2010.

- D. Damljanovic, M. Agatonovic, H. Cunningham: *Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction.* In Proceedings of the 7th Language Resources and Evaluation Conference (LREC'10), ELRA 2010, La Valletta, Malta, May 17-23, 2010.

- A. Wyner, K. Angelov, G. Barzdins, D. Damljanovic, B. Davis, N. Fuchs, S. Hoefler, K. Jones, K. Kaljurand, T. Kuhn, M. Luts, J. Pool, M. Rosner, R. Schwitter, J. Sowa: *On Controlled Natural Languages: Properties and Prospects.* In N. Fuchs, ed.: Controlled Natural Language. Volume 5972 of Lecture Notes in Computer Science, pp. 281–289, Springer Berlin/Heidelberg, 2010.

- D. Damljanovic, K. Bontcheva: *Towards Enhanced Usability of Natural Language Interfaces to Knowledge Bases.* In V. Devedzic and D. Gasevic (Eds.), Special issue on Semantic Web and Web 2.0, Annals of Information systems, Springer-Verlag, 2009.

- D. Damljanovic, M. Agatonovic, H. Cunningham: *Usability of Natural Language Interfaces for Querying Ontologies*, Workshop on Controlled Natural Language (CNL'09), Marettimo Island, Italy, June, 2009.

- D. Damljanovic, K. Bontcheva: *Enhanced Semantic Access to Software Artefacts.* In Workshop on Semantic Web Enabled Software Engineering (SWESE'08) held in conjunction with ISWC'08, Karlsruhe, Germany, October, 2008.

- V. Tablan, D. Damljanovic, K. Bontcheva: *A Natural Language Query Interface to Structured Information.* In Proceedings of the 5h European Semantic Web Conference (ESWC'08), Tenerife, June, 2008.

- D. Damljanovic, V. Tablan, K. Bontcheva: *A Text-based Query Interface to OWL Ontologies.* In: 6th Language Resources and Evaluation Conference (LREC'08), Marrakech, Morocco, ELRA, May, 2008.

- D. Damljanovic: *Natural Language Queries for Enhanced Knowledge Access*, Summer School on Multimedia Semantics Analysis, Annotation, Retrieval and Applications (SSMS'07) , Glasgow, UK, July 15-21, 2007.

Contents

List of Figures

Part I

What are Natural Language Interfaces to Conceptual Models?

Chapter 1

Introduction

The context of this thesis is the task of *automatically answering Natural Language questions by a machine as if it were a human.* This task has long been a subject of research in the Natural Language Processing (NLP) and Knowledge Representation (KR) fields, and is a project still some way from completion. These two fields are subfields of Artificial Intelligence, and complement each other. Recent advances in KR have been influenced by the invention of World Wide Web, and driven by work on the *Semantic Web*: the idea to make the Web and all information on the Web interoperable and understandable by computers, so that applications (e.g., agents) can understand, use, share and reason about them. Many KR languages have been invented for this purpose including RDF – Resource Description Framework [Manola and Miller, 2004] and OWL – The Web Ontology Language [Smith et al., 2004]. These languages encapsulate knowledge of the world through a set of *concepts* and *relations* between them. These are organised into *triples* which have the form

```
SUBJECT <predicate> OBJECT
```

That is, two concepts – subject and object – related by a predicate.

These together form *conceptual models* or *ontologies*. In the domain of computer science, the term *ontology* refers to a logical schema of roles and concepts and the relationships between them (TBox) [Antoniou and van Hermelen, 2008]. *Knowledge base* refers to the actual data such as instances

or individuals that are generated based on the definitions in the ontology (ABox). In practice, the ontology and the knowledge base are often published together, and hence these terms are often used interchangeably in literature referring to both the TBox and ABox. In this thesis we will use the term *semantic resources* to refer to both *ontologies* and *knowledge bases*.

Consider the following example:

> Mary works for University of Sheffield, which is located in Sheffield. Sheffield is located in the United Kingdom. Mary lives in Sheffield.

As triples:

```
MARY <is a> PERSON
UNIVERSITY OF SHEFFIELD <is an> ORGANISATION
MARY <works for> UNIVERSITY OF SHEFFIELD
SHEFFIELD <is a> CITY
UNIVERSITY OF SHEFFIELD <is located in> SHEFFIELD
UNITED KINGDOM <is a> COUNTRY
SHEFFIELD <is located in> UNITED KINGDOM
MARY <lives in> SHEFFIELD
```

If these triples were written in a specific KR language such as OWL, accessing them to answer queries such as *In which country does Mary live?* would require knowledge of a formal query language such as SPARQL – Simple Protocol And RDF Query Language [Prud'hommeaux and Seaborne, 2008]. This poses significant difficulties for non-expert users, while the experts need to be familiar with the existing ontology structure. The role of Natural Language Interfaces (NLIs) to conceptual models is to, given a Natural Language question, find the correct answer in the model. NLIs are more intuitive than alternatives such as formal query languages as they hide complexities of both formal languages and the knowledge structure. In this thesis, our main goal is to *explore Natural Language Interfaces to Conceptual Models in order to improve the task of answering Natural Language questions by machines.*

1.1 Motivation

The knowledge representation languages built on RDF and OWL are becoming increasingly popular. With billions of triples being published in recent years, such as those from Linked Open Data (LOD)[1], there is a need for more user-friendly interfaces which will bring the advantages of the data to casual users. Research has been very active in developing interfaces for accessing structured knowledge, from faceted search, where knowledge is grouped and represented through taxonomies [Croft et al., 2009], to menu-guided and form-based interfaces such as those offered by the Knowledge and Information Management (KIM) platform [Popov et al., 2003]. While hiding the complexity of underlying query languages such as SPARQL or SeRQL (Sesame RDF Query Language) [Broekstra and Kampman, 2003], these interfaces still require that the user is familiar with the queried knowledge structure. However, casual users need to be able to access the data despite their queries not matching exactly the queried data structures [Hurtado et al., 2009].

According to the interface evaluation conducted in Kaufmann and Bernstein [2007], systems developed to support Natural Language Interfaces are perceived as the most acceptable by end-users. This conclusion is drawn from a usability study, which compared four types of query language interfaces to knowledge bases and involved 48 users of general background. The full-sentence query option was significantly preferred to keywords. However, using keywords for querying was preferred to menu-guided, or graphical query language interfaces.

On the other hand, evaluation of the CHESt [Linckels and Meinel, 2007] system (dealing with computer history and accepting both keywords and NL queries as input), revealed users preference for keywords. When asked if they would accept typing full blown questions instead of keyword-based queries, 22% of users answered positive, 69% said they would accept only if this yielded better results, and 8% of users disliked this option.

Web users are used to typing primitive questions into the text box of a search engine. Search engines like Google are capable of answering simple questions

[1] `http://linkeddata.org/`

like *what is the capital of Serbia*. However, the power of Linked Data is in the capability to answer more complex queries for which the answer cannot be found through Google, as the question as such is not contained in any document, and requires retrieving various data in different resources, and then combining and possibly reasoning about them.

Also, much data on the Web is accessible through the use of applications based on relational databases. According to Iskold [2008] semantic technologies are here to help us represent relational data spread over the entire Web: it is relational queries that semantic search engines would excel at. As it is concluded in Iskold [2008], the semantic web is going to help us resolve complex, inferencing queries asked over the entire Web as if it was a database. Expressing such complex queries requires using Natural Language (NL), as a set of keywords is not sufficient.

1.2 Challenges

Building NLIs to structured data requires handling challenges related to the Natural Language understanding such as *ambiguity* and *complexity* (e.g. [Church and Patil, 1982]), see Figure 1.1. Ambiguity can be avoided through the use of Controlled Natural Language (CNL): a subset of the respective natural language that is specifically designed to serve as a documentation, specification or knowledge representation language [Fuchs et al., 2006]. A CNL typically includes a set of vocabulary, grammar rules and restrictions that have to be followed by end-users. In addition, when interpreting questions, CNLs use some predefined strict rules. One example is Purposefully Restricted English (PRE) [Epstein, 1985], a restricted English database query language, which solves ambiguity by following the rule: *relative clauses modify the rightmost available heads*. Hence, if the query is *Find an employee who was hired by a recruiter whose salary is greater than $30000*, the relative clause *whose salary is greater than $30000* would always modify *recruiter*, not *employee* (see Epstein [1985]). The problem with a CNL is that it remains *formal*, and although more intuitive to casual users than languages like SPARQL, still must be *learned* to be used efficiently.

Another big challenge is related to the *expressiveness* of the natural language:

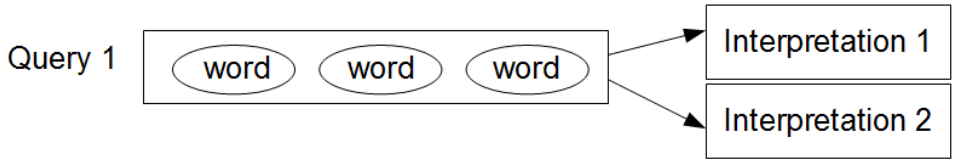

Figure 1.1: Ambiguity

it is possible to express the same meaning using different constructions (see Figure 1.2). This increases the difficulty of automatically interpreting natural language [Cimiano et al., 2007], and it is very challenging to build a *robust* NLI. *Robust* in this context means being able to interpret the same way several alternative natural language queries which have the same meaning but are expressed differently (for example, *what is the largest city in Germany?* vs. *give me the largest city in Germany*, etc.). In order to support all morphological inflections of words NLIs usually operate on the *lemmas* rather than on the *exact string* matches. To handle synonyms, many systems use external sources such as WordNet [Fellbaum, 1998]. To support as many grammatical constructions as possible, NLIs often enumerate the question patterns in advance, and then detect the *question category* based on which the question is further interpreted. The question category is identified either by using *manually constructed rules* for automatic classification, or by using *fully automatically* constructed classifiers usually based on Machine Learning algorithms. A problem with the former approach is that it is time-consuming as the rules are hand-crafted. A problem with the latter approach is that the automatic classifiers must be trained using large dataset in order to work effectively.

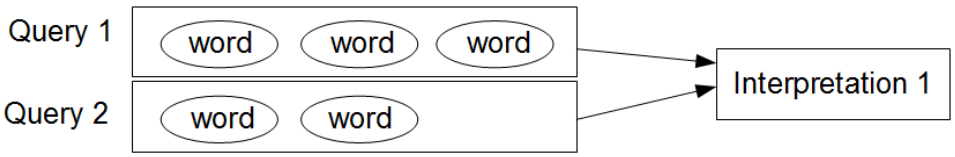

Figure 1.2: Expressiveness

According to [Grosz et al., 1987], a major challenge when building NLIs is to provide the information the system needs to bridge the gap between the

way *the user* thinks about the domain of discourse and the way *the domain knowledge is structured* for computer processing. This implies that in the context of NLIs to conceptual models, it is very important to consider the ontology structure and content. Two ontologies describing identical domains (e.g., music) can use different modeling conventions. For example, while one ontology can use a datatype property *artistName* of class *Artist*, the other one might use instances of a special class to model the artist's name[2]. A *portable* NLI system would have to support both types of conventions without sacrificing performance. *Portable* or *transportable* NLIs are those that can be adapted easily to new domains (or new ontologies covering the same domains). Although they are considered as potentially much more useful than domain-specific systems, constructing transportable systems poses a number of technical and theoretical problems because many of the techniques developed for specialised systems preclude automatic adaptation to new domains [Grosz et al., 1987]. Moreover, it is noted that portability affects retrieval performance: "the more a system is tailored to a domain, the better its retrieval performance is" [Kaufmann and Bernstein, 2007, p.281].

In general, existing NLI systems tend to be either domain independent (i.e., portable) but with lower performance, or more domain-specific (i.e., portable only 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 experts (e.g., domain experts, language engineers). Semantic resources can be constructed to include sufficient lexical information to support a domain-independent query analysis engine. However, due to different processes used to generate ontologies and knowledge bases, the lexicon might be of varying quality. In addition, some words might have different meanings in two different domains. For example, *How big* might refer to *height*, but also to *length*, *area*, or *population* – depending on the question context, but also on the ontology structure. This kind of adjustments – or mappings from words or phrases to ontology concepts/relations, is performed during customisation of NLIs.

Finally, while NLIs are intuitive, having only one text box for a query can pose difficulties to users, who need to express their information need

[2]See for example how class *Alias* is used in the Proton System Module ontology: `http://proton.semanticweb.org/`

through a natural language query efficiently [Stojanovic, 2005b]. In order to address this problem, several methods have been developed with the aim to either assist the user to formulate the query, or to communicate the system's interpretation of the query to the user. However, a real challenge when building NLIs is to hide all complexities of the underlying knowledge from the casual user, and to provide him either the answer, or appropriate guidance on how to reformulate the query in order to get the answer.

To summarise, the major challenges when building NLIs to conceptual models are:

- *Ambiguity*: unambiguous transformation from a NL query into a formal query.
- *Robustness/Expressiveness*: supporting query variations which have the same meaning although expressed using different constructions.
- *Portability*: being able to easily port an NLI system from one domain or ontology to another.
- *Keeping the supported language intuitive.*
- *Hiding complexities of the queried knowledge structure*: showing results without imposing users to the underlying complexities of the structured knowledge.
- *Guiding the user* through the process of formulating queries.

1.3 Contribution

This thesis reports work from a research programme on minimal-lexicon CNL and NLIs to structured data. This programme began in 2003 as part of the Semantic Knowledge Technologies project[3], and was initially concerned with how to reduce the costs and lack of flexibility associated with the need to provide precise and extensive lexical data for each CNL system. NLI and CNL systems are increasingly relevant for information systems fronting rich structured data stores such as RDF and OWL repositories, largely because

[3] `http://www.sekt-project.com/`

of the complexity and syntactic unfamiliarity of the underlying triple models and the query languages built on top of them.

The first result from the work was CLOnE, a **C**ontrolled **L**anguage for **ON**tology **E**diting [Tablan et al., 2006, Funk et al., 2007], which proved capable of expressing simple ontologies in an English-like language without a sophisticated lexicon (only a fairly small number of key terms and phrases stored in a gazetteer were required). The work reported in this thesis built on this concept and has resulted in two further systems, QuestIO and FREyA. These two systems, their design, implementation, comparison with related work, and quantitative performance evaluation form the core contribution of the thesis.

QuestIO, a **Quest**ion-based **I**nterface to **O**ntologies [Damljanovic et al., 2008, Damljanovic and Bontcheva, 2008, Tablan et al., 2008], turns from the construction and editing of the knowledge store (as in CLOnE) to the querying of the data. As with the previous work our motivation was to provide simpler interfaces (as noted above the leading query language, SPARQL, is prohibitively complex for casual users) while avoiding the cost of producing and maintaining a separate sophisticated lexicon. In this case we are working in the context of existing semantic resources (authored in CLOnE or extracted automatically from text, or generated by other tools and processes). Our approach was therefore to use the terminology explicit in the ontology and the knowledge base, along with the structural relationships between concepts and the properties of concepts (and a CLOnE-like mechanism for analysing key terms and phrases such as "how many...?").

With QuestIO we have addressed the following challenges:

- *Ambiguity*: QuestIO resolves ambiguities automatically, based on a ranking derived from the ontology structure. The ranking is based on an algorithm which combines similarity measures based on ontology hierarchy with existing algorithms for string similarity.
- *Portability*: the *domain lexicon* is extracted automatically from the ontology and the knowledge base and no customisation is necessary. This approach is very similar to the existing approaches used in the NLIs developed at about the same time as QuestIO. The component

implemented for this purpose is a gazetteer called OntoRoot Gazetteer, which became a GATE plugin in 2007 and since then is used extensively by the GATE users. Its main application is the semantic annotation based on the provided ontology/knowledge base.

- *Expressiveness*: the supported query language allows different ways of expressing the same meaning as long as the question terms exist in the *domain lexicon*. QuestIO is one of the first NLIs for the semantic web which supports relaxed queries (ill-formed or incomplete) as well as the full-blown grammatically correct questions. This flexibility comes from the above mentioned gazetteer. The other similar system with respect to expressiveness of the language is NLP-Reduce [Kaufmann et al., 2007], which is developed in Zurich at about the same time.

QuestIO is evaluated on two domains:

- *General knowledge data* from KIM [Popov et al., 2003], which contains facts about people, organizations, geographical locations and the like.
- *Software engineering data* created in the TAO project (Transitioning Applications to Ontologies[4]) and with the questions from users in that project.

The results were both positive and negative. On the positive side, our performance was as good or better than related systems (we have performed comparative evaluation with AquaLog [Lopez and Motta, 2004, Lopez et al., 2007], a mature query system from the Open University). On the negative side:

- Resolving ambiguities automatically relies heavily on the ontology structure and hence only results in good performance for small and manually crafted semantic resources, while for larger repositories the query execution time for some queries can be intolerable.
- The expressiveness of the supported query language (language coverage) was sufficiently limited as to regularly cause problems for users:

[4] `www.tao-project.eu`

the vocabulary used by the users often differed from the lexicon derived from the semantic resources.

QuestIO and its evaluation forms the first half of our work.

The second half of the work was motivated by the problems just referred to, and also by the challenges which have arisen when using this technology to query resources from the Linked Open Data initiative. It became clear that the significance of our work would be increased if we could develop mechanisms that were appropriate to work with the large linked data, as well as with manually-crafted data. We investigated whether *user interaction* coupled with deeper syntactic analysis and usability methods such as feedback, extending vocabulary, and query refinement can be used in combination to improve the usability of NLIs to conceptual models. These form the base of the FREyA system: **F**eedback, **R**efinement and **E**xtended Vocabular**Y** **A**ggregation [Damljanovic et al., 2009, 2010a,b, Damljanovic, 2010]. Work on FREyA is reported in two parts:

- In the first part we further address the problem of *ambiguity* by combining automatic ranking (as in QuestIO) with the user's selections. Our approach of solving the ambiguity by involving the user into dialog is very similar to the ones used in AquaLog [Lopez and Motta, 2004, Lopez et al., 2007] and Querix [Kaufmann et al., 2006], with the difference in the underlying ranking and automatic disambiguation mechanisms which precede the dialog. However, the approach of using the dialog to show *feedback* to the user is novel and has not been researched extensively. We explore the effect of *feedback*, by showing the user the list of system interpretations of the query and the context from which the answer is derived. Unlike QuestIO, which is fully automatic and does not give the opportunity to the user to validate the machine interpretation, in FREyA the user can choose alternatives if the one selected by the system does not seem valid. This approach has been evaluated in the task-based user-centric evaluation which specifically assessed whether the new usability features of FREyA had any effect in comparison to QuestIO. The evaluation was conducted using two domains:

  - *software engineering*: which is used in the evaluation of QuestIO; the data have been generated in the TAO project mentioned above
  - *US geography*: the Mooney GeoQuery dataset[5] which has been extensively used for evaluation of NLIs to databases and recently for NLIs to ontologies/knowledge bases

- The second part is concerned with
  - *improved ambiguity resolution*: disambiguation is moved from the *query interpretation level* to the *concept interpretation level*: instead of trying to automatically interpret the whole question at once, we are interpreting each concept individually, and engaging the user in the dialog only if necessary. This way, we reduce the cognitive overhead for the user, while at the same time each query is interpreted in one unambiguous way where the user is in control.
  - *the more expressive query language*: our approach enriches its own lexicon (generated from RDF data, and extended by WordNet [Fellbaum, 1998]) from the user's language. The lexicon enrichment is powered by a light learning model, which is designed in a way that can be reused by other NLI systems. Our approach of extending vocabulary is more generic than existing approaches which focus on mapping question terms to ontology relations, examples include AquaLog and ORAKEL [Cimiano et al., 2007].
  - *the deeper grammar analysis*: while QuestIO uses very shallow NLP, FREyA uses the parsed syntax tree in combination with the ontology-based lookup in order to interpret NL questions. We implemented a novel consolidation algorithm which attempt to automatically merge the results of the two processes.
  - *learning from the users*: our learning algorithm is a novel approach to using the ontology as the context for improving the system over time and learning to map query terms to ontology concepts and relations. Other similar approaches exist but they

[5]The original dataset is available from `http://www.cs.utexas.edu/users/ml/nldata.html`. The dataset used in this thesis is available from `http://www.ifi.uzh.ch/ddis/research/talking-to-the-semantic-web/owl-test-data/`.

model the context differently and also focus on personalising the vocabulary (e.g., AquaLog).

- *ranking algorithms*: each dialog consists of a term which needs to be resolved, and a list of suggestions for the user to choose from. The ranking which is implemented is used to reduce the cognitive overhead and combines existing string similarity algorithms with the synonym detection based on WordNet.
- *the dialog sequence*: answering a question correctly might require more than one dialog to be modelled. Selecting the order in which to model dialogs can significantly affect results. We implemented a novel algorithm for deciding in which order the concepts will be disambiguated or mapped to an ontology concept/relation.
- *the scope* in QuestIO was limited to using one RDF document at the time. In FREyA we extend the scope by making it flexible in terms of the number of ontologies that can be queried. Indeed, it is possible to connect to a remote repository using FREyA (e.g., using a SPARQL endpoint) as well as to load a set of RDF documents (ontologies and knowledge bases) locally into its internal repository. Most of the existing NLI systems were evaluated using one domain at the time, with the exception of the PowerAqua [Lopez et al., 2009b] system, which evolved from AquaLog. PowerAqua aims to serve as a Question-Answering system for the Semantic Web and was evaluated in the open-domain scenario [Lopez et al., 2011] (e.g. through querying the semantic resources indexed by Watson [d'Aquin et al., 2007]).
- *showing feedback to the user*: the concise answer is derived based on the novel algorithm for identification of the answer type which does not require strict adherence to syntax. Hence, although the user does not have to enter a grammatically correct question, the ontology structure in combination with the grammar analysis is used to correctly identify the answer type. In addition, the user is presented with all concepts and relations that are used before the concise answer is derived.

FREyA was evaluated with the Mooney GeoQuery dataset for the sake of

comparison to other similar systems such as PANTO [Wang et al., 2007], NLP-Reduce [Kaufmann et al., 2007] and Querix [Kaufmann et al., 2006]. FREyA outperformed all other similar systems (although it required dialog with the user).

In conclusion, our key findings are that:

> a) *combining syntactic parsing with ontology-based lookup in an interactive process of feedback and query refinement can* ***increase the precision and recall*** *of NLIs to ontologies/-knowledge bases, while*
>
> b) ***reducing porting and customisation time*** *by shifting some tasks from application developers to end-users*

It is important to distinguish the usability of Natural Language Interfaces from the point of view of *application developers* who are customising the system, and *end-users* who are querying the system. While addressing *ambiguity* and *expressiveness* the NLI system becomes more usable for end-users, the *portability* issue is tightly coupled with the usability from the application developer's point of view. The less time the application developers spend customising the system, the more usable it becomes from their point of view. Our proposed methods attempt to strike a balance between heavy customisation, and the end user needs to explore the available knowledge without being constrained by the query language.

The rest of the thesis is structured as follows:

Part I introduces Natural Language Interfaces to Conceptual Models and contextualises our work in relation to that of others in this and related fields:

- Chapter 2 briefly introduces conceptual models and the interfaces which are used for browsing them.
- Chapter 3 reviews the history of Natural Language Interfaces including NLIs to databases, open-domain Question-Answering systems, interactive NLIs, and NLIs to ontologies.

- Chapter 4 outlines evaluation strategies for Natural Language Interfaces, making clear the distinction between the point of view of developers customising NLI systems, and users who are querying them.

Part II details and analyses the approaches which have been applied by previously existing NLIs:

- Chapter 5 reviews Natural Language Interfaces to ontologies (with special attention to their customisation).
- Chapter 6 reviews and classifies existing methods for increasing the usability of NLIs from the end-users point of view.

Part III describes our approach to Natural Language Interfaces to ontologies:

- Chapter 7 details the design and evaluation of QuestIO.
- Chapter 8 reports our initial FREyA design, the implementation of feedback, and its evaluation with users.
- Chapter 9 reports the final FREyA design, the evaluation of its subcomponents, and of the system as a whole.

Part IV concludes with a summary of the contribution of the thesis (Chapter 10) and plans for future work (Chapter 11) in our research programme.

Chapter 2

Conceptual Models

2.1 What are Conceptual Models?

The inventor of the World Wide Web (WWW), Tim Berners-Lee, proposed a new generation of the Web [Berners-Lee, 1999], called the *Semantic Web*, where data has well-defined meanings expressed in a form that can be easily interpreted by both computers and people. The idea is to have data on the Web defined and linked in such a way that it can be used for more effective discovery, automation, integration, and reuse across various applications [Guha et al., 2003]. As envisaged by Guha et al. [2003], the Semantic Web will contain resources corresponding not just to media entities (such as Web pages, images, audio clips, etc.) as the current Web does, but also to objects such as people, places, organisations and events. Furthermore, the Semantic Web will define structured relations, not just hyperlinks, among the different types of resources mentioned above. Each resource can have metadata attached to it.

The metadata is usually described using a special language which is capable of expressing *concepts* and the different *relations* between them. These together form *conceptual models* or *an ontology* which is defined as "*an explicit specification of a conceptualisation*"[Gruber, 1993], where *conceptualisation* is "*an abstract, simplified view of the world that we wish to represent for some purpose*".

In other words, an ontology formally describes a domain of discourse by

defining concepts and how they relate to each other. For example, in the domain of tourism, a set of concepts which are frequently used are: *tourists*, *destinations*, and *events*, while one of the relations which describes a connection between a *tourist* and a *destination* is *interestedIn*, so that using formal expressions it can be expressed as

```
TOURIST <interestedIn> DESTINATION
```

This is an example of the basic element of an ontology which is a *triple*, with the form

```
SUBJECT <predicate> OBJECT
```

Creating instances of ontology classes and connecting them using ontology relations (predicates) leads to generating a *knowledge base* – as previously discussed in Chapter 1, *ontologies* and *knowledge bases* are usually published together and hence both terms are used interchangeably in literature to refer to both the schema and the data.

W3C organisation recommends OWL, The Web Ontology Language, as a semantic markup language for publishing and sharing ontologies on the World Wide Web[1]. Early knowledge modelling languages include F-logic [Kifer et al., 1995], OCML [Motta, 1999], DAML+OIL [Horrocks, 2002] and others.

Ontologies facilitate *semantic search* [Davies et al., 2002]. Semantic search is an application of the Semantic Web to search, and it attempts to augment and improve traditional search results (based on Information Retrieval technology) by using data with explicit semantics from the Semantic Web [Guha et al., 2003].

Starting with the initiative of the Linked Open Data[2] project, the term *Linked Data* gained in popularity. According to the definition from the Web site of Linked Open Data project, *the term Linked Data refers to a set of best practices for publishing and connecting structured data on the Web.* The Linked Data project can be seen as a simplification of the initially proposed

[1] `http://www.w3.org/TR/owl-ref/`
[2] `http://linkeddata.org/`

idea of the Semantic Web. Described by Tim Berners-Lee (who has also been credited for coining *Linked Data* term), Linked Data is "*the Semantic Web done right*".

2.2 Browsing Conceptual Models

One of the most popular tools which, among other features, allows browsing ontologies and knowledge bases is Protégé[3]. For querying, users can use a template-based form where they specify parts of a triple they are interested in, and the missing parts will be given as a result - if found in the semantic repository. Another way to query an ontology with Protégé is by writing SPARQL queries: results are given in the form of triples. This platform is very useful for experts who are familiar with query languages although they also have to be experienced Protégé users. In comparison to Protégé, KIM [Popov et al., 2003] goes one step further in simplifying the browsing process – it provides predefined query templates, where users can construct SeRQL queries 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 structure.

At almost the same time as KIM, the TAP system was developed [Guha et al., 2003]. The idea of TAP is to enable browsing and searching for the specific semantic resources. Two graphical interfaces for querying ontologies have as a starting point the node which is described by a URI , and they return a graph describing the given URI. The third interface of TAP is called *Search* – it takes a string as an input and returns all resources whose *title properties* contain the string. *Title property* is specific to the TAP knowledge base. A more widespread approach nowadays is to use the property *rdfs:label* for the same purpose.

TAP interfaces were tested with RDF files maintained by W3C. In addition, for larger applications dealing with musicians, athletes, places, and so forth, HTML scrapers are built to get the data from the popular sites such as Amazon or AllMusic. That is, their Web crawler is dynamically locating

[3] `http://protege.stanford.edu/`

and converting relevant pages into machine readable data so that they become available for search by the TAP interfaces. The dynamically created knowledge base contains many millions of triples.

While trying to improve traditional search by using a denotation of the search term, Guha et al. [2003] encounter the following problems:

Denotation: determining the concept denoted by the search query is not straight-forward. The biggest problem is *ambiguity*, which is solved by preferring some denotations driven by a few heuristic rules, for example: popularity of the term (*Paris* as *the capital of France* preferred to *Paris* in *Texas*), the user profile, and the search context. Another problem is related to what is called a *complex search term*: the subsets of the search term map to different nodes. For example, to cite example from [Guha et al., 2003] the query "eric miller rdf" can be broken down into "eric miller"+"rdf". The first mapping to the node corresponds to the person Eric Miller and the second mapping to the Resource Description Framework. Due to the complexity which arises with having several terms together in a query, complex terms are restricted to two denotations only.

What to show: which data to pull from the semantic web and in which order to present them. The node that is the selected denotation of the search term provides a starting point. The next problem is which subgraph around this node to show. A more balanced subgraph is produced by using heuristic rules based on the average branching factor (i.e. bushiness) of the graph around the anchor node.

In the *GetData* interface, for example, there is a possibility to customise the system by specifying which properties should be shown with each resource. These properties are then presented first. If nothing is specified, the TAP shows all available properties. The problem with this approach is that it is very difficult to know the names of properties in advance – the user must be very familiar with the structure and available knowledge in order to perform this customisation. However, when a system is not customised and the queried knowledge is large this approach might confuse the user rather than help find what is being searched for.

Presentation : the main problem is how to present the resulting data/triples.

Seven years later, these problems remain big challenges for semantic search and browse interfaces as we will discuss later in this thesis.

Probably due to its visual similarities to common search engines, *TAP Search* interface has received a lot of attention. The main goal with this interface was to make search engines capable of interpreting the different occurrences of the same input string as different semantic concepts. In *Search* interface, this problem is solved by asking the user to choose between available options.

Four types of question are supported by the TAP:

1. Entity search e.g. *Johnny Depp*
2. Comparison of entities e.g. *buildings taller than the Tower Bridge*
3. Attributes of entities e.g. *birthday of Johnny Depp*
4. Group queries e.g. *birthday of Johnny Depp and Nicole Kidman or countries with population greater than 100 million*

These four types of questions are handled by 37 patterns which contain the rules of how to handle and answer them. If the input query/question is not recognised as belonging to one of these 37 patterns, the answer is not returned.

The user-centric evaluation of the TAP Search interface revealed how dominant the influence of search engines on casual users is. According to the evaluation presented in da Costa et al. [2005], it appears that users expect the semantic search interface to be similar to that of the search engine's. Namely, the first prototype of TAP Search interface rendered the results on the right side of the page, and information about ontology entities (the position of the queried entity inside the ontology) on the left. After the evaluation, this was changed as it was disliked by users, who "learnt" to ignore the right side, as that is the place for advertisements. TAP changed the interface so that the results are shown underneath the query, in a similar way to popular search engines like Google or Yahoo.

With regard to the previously mentioned form-based interfaces they are convenient for repetitive searches but not for ad hoc queries [Tran et al., 2010]. *Faceted search* is very similar to form-based, with the difference that facets are generated dynamically based on the user's query, and are not predefined such as in the case of forms. It has been argued that faceted search browsers are extremely helpful in cases when the user's information need is vague [Mäkelä, 2006]. The example of faceted search is displayed in Figure 2.1, which is a screenshot of the *museumFinland* portal[4], which uses this kind of interface for searching data about three museums in Finland. Even without any intention to search, the user can browse the available categories and explore the knowledge step-by-step. Different approaches can also be combined such as in Wang et al. [2009] where the authors introduce a *hybrid query* which combines a keyword query with the precise structured query.

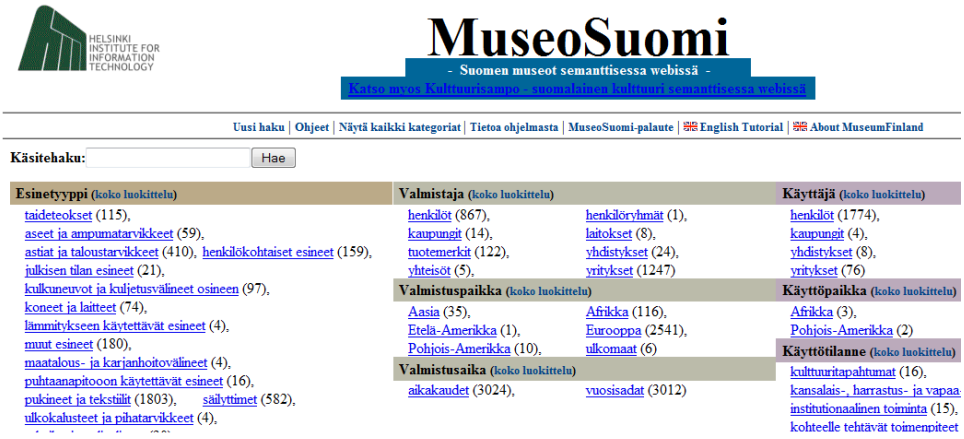

Figure 2.1: The initial search page from the MuseumFinland portal

Ontotext[5] developed several interfaces for exploring a part of Linked Data, including *RDF Search*. RDF Search is powered by an auto-complete option (see Figure 2.2), the implementation of which is based on Lucene[6] used to index the underlying knowledge. While the user is typing in a string or a URI, the suggestions generated by indexing the available knowledge will be offered to the user. These suggestions are generated not only by considering

[4] `http://www.museosuomi.fi`
[5] `www.ontotext.com`
[6] `http://lucene.apache.org`

available URIs and the local names, but also the labels, and literals used to describe nodes.

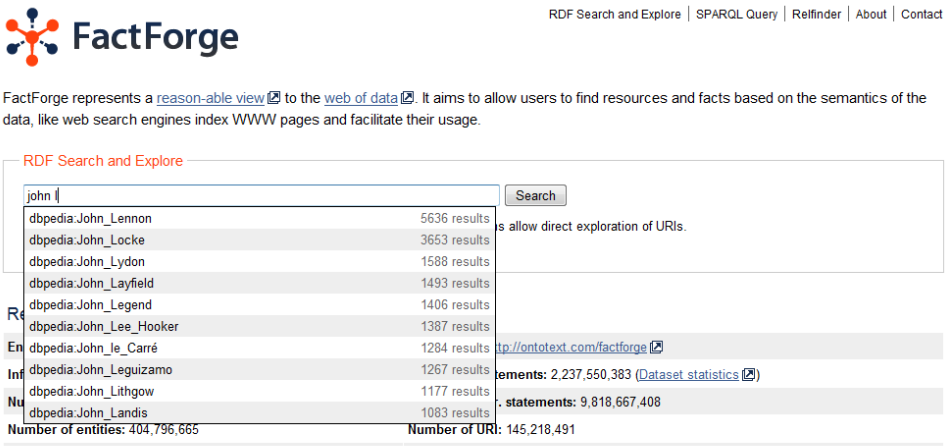

Figure 2.2: RDF Search by FactForge (www.factforge.net)

The most powerful way to query OWL/RDF ontologies is still SPARQL, however, it does remain complex and is time-consuming even for experts. The Franc Inc.[7] company developed a graphical interface (Gruff, AllegroGraph triple-store browser) where the user can drag-and-drop nodes and combine them in order to generate SPARQL. A sample graph displaying parts of an ontology from the geography domain is shown in Figure 2.3.

While graph-like structure appears to be the most natural way to display RDF graphs, displaying large amounts of data remains tempting, for example to show all mountains in Figure 2.3. The user unfamiliar with graphs might be confused by having two or more nodes with the name *Mountain* and wonder whether there is any difference between the two. Designers of user interfaces must make a presentation choice which is in-line with expectations of their users.

Many systems support semantic search over documents, including the above mentioned TAP and KIM, the SHOE Search tool [Heflin and Hendler, 2000], the DOPE Browser [Stuckenschmidt et al., 2004] (see Figure 2.4), SemSearch [Lei et al., 2006], or a system developed by Ding et al. [2006]. While a

[7] `http://www.franz.com/`

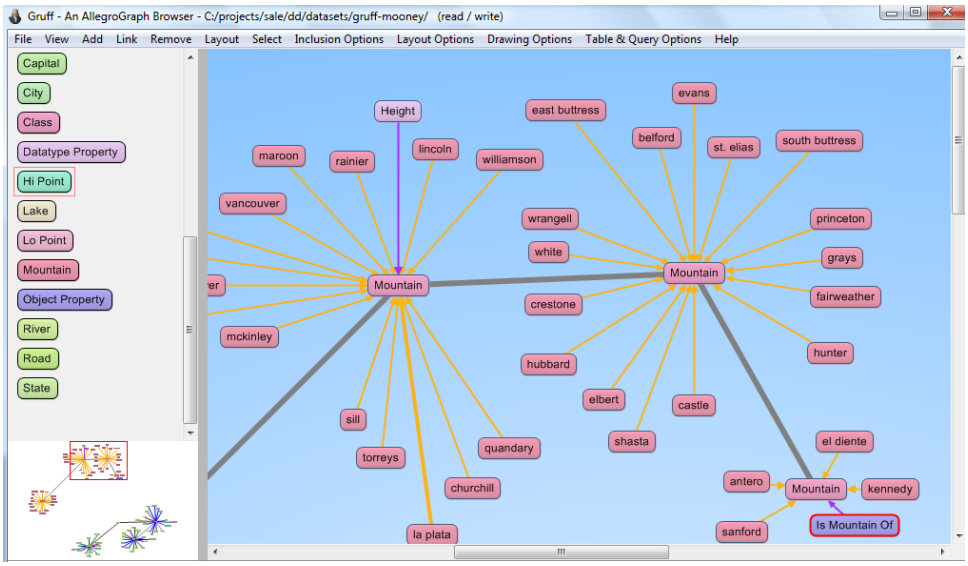

Figure 2.3: The Gruff interface showing the *Mountain* concept from the geography domain ontology

considerable amount of work has been done in this area, our focus is on browsing conceptual models and thus the interfaces for semantic search over documents will not be discussed further in this thesis.

Summary Many interfaces have been developed for browsing and searching RDF spaces. The range varies from form-based, faceted searches to graphical interfaces, and also keyword-based and Natural Language Interfaces. While graphical interfaces put some limitations on the users in terms of what can be queried, free text searches and Natural Language interfaces seems more intuitive and less constrained.

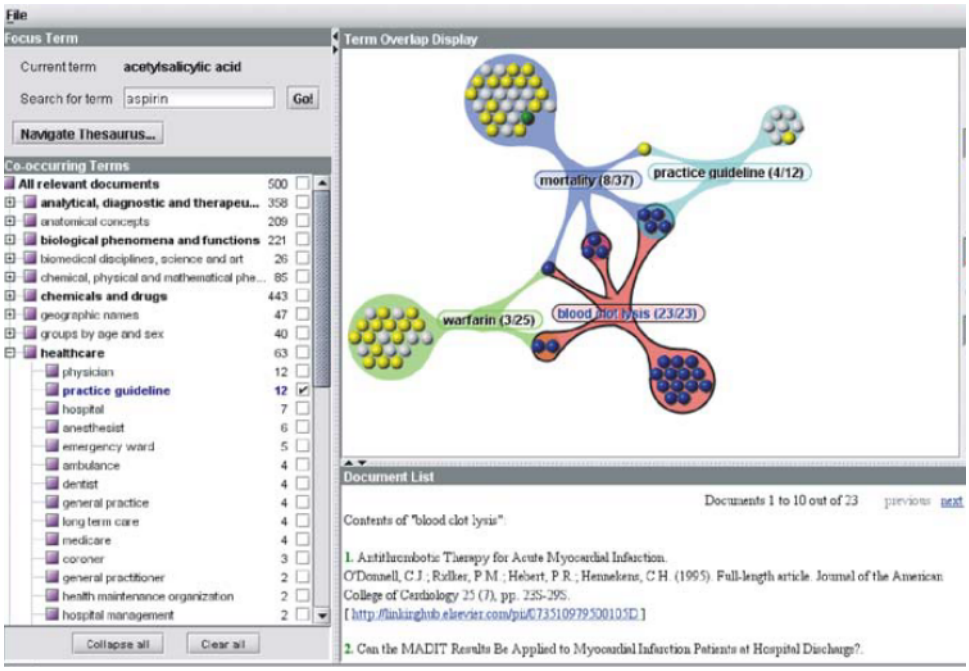

Figure 2.4: The DOPE Browser: searching for documents related to 'aspirin'

Chapter 3

Natural Language Interfaces: a Brief Overview

Research in the area of Natural Language Interfaces (NLIs) has been around for more than four decades. From the end-users point of view natural language is easy to use as it is used everyday in human to human communication, and is therefore considered as a useful and efficient way for people to interact with computers [Ogden and Bernick, 1997]. NLI systems have Natural Language questions as input and are built for various purposes. Most of them are concerned with the knowledge access problem, and among these, they further differ in terms of the underlying knowledge structure, and therefore can be grouped into three main categories:

NLIs to structured data. NLIs to structured data allow users to interact with a system using written or spoken language (e.g. English) to perform tasks that usually require knowledge of a formal query language. The intention behind building NLIs to structured data is enabling users with no knowledge of formal languages to use them with minimal (ideally no) training. These systems are often domain-specific, and are usually referred to as *closed-domain question answering* systems. Two major subgroups include:

- *NLIs to relational databases (NLIDBs)* translate Natural Language into SQL in order to retrieve answers from the database.

Most of the developed NLIs to structured data belong to this group (e.g., [Popescu et al., 2003], [Thompson et al., 2005], [Hallett et al., 2007], to mention a few recent ones). Recently, these evolved towards interfaces to semantically-richer data in the form of ontologies.

- *NLIs to ontologies* translate a Natural Language query into the formal query language which is used to retrieve the knowledge expressed in one of the knowledge representational languages (such as OWL). The most common query language is SPARQL. Recently developed systems include ORAKEL [Cimiano et al., 2007], AquaLog [Lopez et al., 2007] and PowerAqua [Lopez et al., 2009b], PANTO [Wang et al., 2007], and Querix [Kaufmann et al., 2006].

NLIs to unstructured or semi-structured data differ from the previous group in that they do not translate the Natural Language query into any formal language but they rather process the collection of documents (e.g. News articles on the Web, or Frequently Asked Questions as in Burke et al. [1996]). However, similar to the previously mentioned NLIs, the aim of these systems is to also find the answer to the question posed by the user. The most prominent systems of this kind are open-domain Question-Answering systems which process large collections of documents in order to find answers. Examples include MURAX [Kupiec, 1993], MULDER [Kwok et al., 2001], and AnswerBus [Zheng, 2002]. Another group which belongs here are Reading Comprehension systems such as Deep Read [Hirschman et al., 1999], which are used to test the reading level of children. They find the answer to the set of questions related to a story which is written in simple Natural Language.

Interactive NLIs are systems which are used for dialogue systems [Cimiano et al., 2007], e.g., a chat bot called Asimov which answers simple questions in English (`http://asimovsoftware.com`). These kind of systems do not consider a set of questions as an independent collection, but rather act as agents or robots which are involved in a conversation with the user, with the capability to *remember* the sequence of previously asked questions, and to interpret the input from the user, and

learn the answers to questions which they could not answer before. These are more challenging to develop in comparison to the previously mentioned NLIs, due to the requirement to model multiple conversational turns. These turns can refer one to another, and such systems must have the ability to remember and respond to all this *context*.

Lastly, a few NLI systems are developed for purposes other than knowledge access, such as systems to replace a programming language, e.g., the NLC system [Alan W. Biermanna and Sigmon, 1983].

In order to put the work of NLIs to ontologies in the context of similar systems, we give an overview of NLIDBs (Section 3.1), Question-Answering systems (Section 3.2), and interactive NLI systems (Section 3.3). We end this chapter with a discussion of how these systems can benefit from each other, and how NLIs to ontologies can be used to boost the performance of other similar systems (Section 3.4).

3.1 Natural Language Interfaces to Relational Databases

First NLIs to relational databases were developed in the late 1960s and early 1970s, among which the most popular was LUNAR [Woods et al., 1972]. LUNAR was built on the top of a database about chemical analysis of moon rocks. Soon after, several other systems were developed such as dialogue-based RANDEZVOUS [Codd, 1974] which is capable of generating clarification dialogs with multiple choice in case it fails to parse the question, and LADDER [Hendrix et al., 1978] which was targeted at large and distributed databases. An impressive feature of LADDER was use of semantic grammars, similar to PLANES [Waltz, 1975, 1978] which answers questions related to airplane maintenance and flight histories. However, this feature had a trade-off which is the requirement to develop a new grammar for each new application domain. PLANES was based on the principles that the input should be non-restrictive for the user (for example, supporting ellipsis – omission of one or more words that can be understood in context), and it also used the dialogue-based features developed by RANDEZVOUS.

In the 1980s, NLIDBs continued to be a popular topic for research with the main focus on portability – at this time many systems have been developed such as

- CHAT-80[1] which translated the limited subset of NL queries into a Prolog internal database [Warren and Pereira, 1982].
- TEAM [Grosz et al., 1987] which was translating NL queries into SODA query language.
- PARLANCE [Bates, 1989a] can be configured by hand or using a component called *Learner*, which is used to generate domain specific configurations.

According to Grosz et al. [1987], "One of the main functions of the NLI is to make the necessary transformations and thus to insulate the user from the particularities of the database". This seemed to be a very hard task, due to many different designs and ways the data can be encoded in the specific structure such as the database schema.

Although several systems have proved to have a great performance especially in particular application domains, the uptake in industry was very slow [Androutsopoulos et al., 1995] – it has not become a standard option for users of DBMS, although several commercial options have appeared. One example is INTELLECT [Harris, 1984], which is capable of translating the NL query into SQL. An example which is used to motivate the usage of INTELLECT in commercial applications is illustrated in Figure 3.1.

As described in Kho [2008], one of the first users of INTELLECT were employees of the Hartford Hospital – one of the largest teaching hospitals and tertiary care centres in New England. Due to the data about patients, doctors and procedures being stored in databases, they were closely controlled by the IT department and it was not possible for domain experts to perform ad hoc queries that could answer critical questions e.g. about identifying new cases of hospital-acquired infections. Therefore, in order to enable easy querying using Natural Language, they deployed INTELLECT

[1]The code of this system is available from `http://nltk.googlecode.com/svn/trunk/doc/howto/chat80.html`

USER: I WONDER HOW ACTUAL SALES FOR LAST MONTH COMPARES TO THE FORECAST FOR PEOPLE UNDER QUOTA IN NEW ENGLAND

INTELLECT: PRINT A COMPARISON OF LAST NAME, 82 JUL ACT SALES AND 82 JUL EST SALES OF ALL SALES PEOPLE WITH REGION = NORTH-EAST & 82 YTD ACT % QUOTA UNDER 100 00

THE NUMBER OF RECORDS TO BE SEARCHED IS 40

LAST NAME	1982 JULY SALES	1982 JULY EST SALES	CHANGE	% CHANGE
SMITH	$54,474	$52,868	$1,606	2 95
ALEXANDER	$54,833	$52,936	$1,897	3 46
ADKINS	$76,072	$75,631	$441	0 58
ASIN	$42,144	$38,214	$3,930	9 33
BRADY	$40,530	$39,569	$961	2 37
COOKE	$41,318	$40,406	$912	2 21
GIRTON	$40,423	$40,393	$30	0 07
GNANDT	$64,213	$63,878	$335	0 52
GOLDSTEIN	$37,977	$37,942	$35	0 09
GOODWYN	$63,779	$63,906	$127-	0 20-
GOULD	$77,161	$75,870	$1,291	1 67
HILTON	$79,412	$75,388	$4,024	5 07
JOCHEM	$66,103	$66,316	$213-	0 32-
JOCHEM	$103,455	$100,808	$2,647	2 56
KENWORTH	$76,879	$77,633	$754-	0 98-

NEXT REQUEST

Figure 3.1: A sample session with INTELLECT, the example taken from [Harris, 1984][p. 45]

– a product from Artificial Intelligence (AI) Corporation which was founded by Larry Harris, Ph.D., who sold it and founded EasyAsk about ten years later. The company was acquired by the Progress Software Corp. in 2005 and now offers solutions for e-commerce sites, including EasyAsk for the Enterprise – which gives a natural-language based access to data and content. In Hartford Hospital, EasyAsk was first deployed in the payroll and microbiology department, where the users could get answers to questions such as *how much vacation time was accrued in a particular department*, *on what date did a particular employee begin working for purposes of employment verification*, or have access to hospital-acquired infections or other patient information through the use of Natural Language queries. EasyAsk is used by many other customers, the full list is available on their website: `http://www.easyask.com/customers/index.htm`.

In the 1990s, the field of NLIDBs focused on learning approaches, with the work from Mooney and colleagues being dominant. Mooney researched machine learning methods, in order to answer the question of whether semantic grammars can be automatically generated from the available examples in the specific domain [Zelle and Mooney, 1993]. Semantic grammars have been successfully applied in NLIDBs, however, as previously discussed, each new domain required newly written grammars – the size of the grammar needed by general applications can make the manual construction infeasible. Also, according to Mooney [1999], in addition to studying syntactic parsing extensively, researchers should focus on understanding the logical representation of the sentence meaning. This logical representation is usually what is strongly related to the underlaying structure of the knowledge (e.g. the structure of the relational database or the ontology).

The early work of Mooney focused on applying Inductive Logic Programming [Zelle and Mooney, 1993, 1996] which is tested within a system called Chill. The input to Chill is a set of questions paired with the respective parses. This set is used to train parsers map NL database queries into executable logical form. According to Mooney [1999], it is a growing trend in computational linguistics to focus on shallow but broad-coverage NL tasks. Logic based learning can be used to develop narrower, domain-specific systems that perform deep processing. Although this learning approach is applied to NLIDBs, the query language to which the natural language queries are transformed is in a logical form, rather than SQL. This is because the logical form is more straight-forward to be mapped from Natural Language and in addition, translating from an unambiguous logical form into query languages such as SQL can be easily automated [Zelle and Mooney, 1996]. In [Zelle and Mooney, 1996], the authors describe experiments in the domain of United States geography. This was motivated by the availability of the system called *Geobase*, which was included in the commercial Prolog for PCs (Turbo Prolog 2.0, Borland International 1988), and was a NLI for a simple geography database. The Geobase data covered information about the United States: population, area, capital cities, states, rivers, the highest and the lowest points and their elevations. A sample question and its query representation in Prolog look like the following:

```
NL: What are the major cities in Kansas?
answer(A,(major(A),city(A),loc(A,B),const(B,stateid(kansas))))
```

The system they developed is still available as an online demo at `http://userweb.cs.utexas.edu/users/ml/geo-demo.html`.

Other approaches for learning the semantic parsers include application of machine learning for learning from ambiguous data [Kate and Mooney, 2007], and also the application of statistical methods such as in Miller et al. [1996] and Wong and Mooney [2006].

The Mooney GeoQuery database was used for evaluation of many systems including PRECISE [Popescu et al., 2003], which focused on portability. The demo of PRECISE is available from `http://www.cs.washington.edu/research/nli/`. The lexicon in PRECISE is generated by automatically extracting value, attribute, and relation names from the database. The authors mention that they manually augmented the lexicon with relevant synonyms, prepositions, etc. [Popescu et al., 2004]. The work in Popescu et al. [2003] focuses on precision, highlighting that PRECISE can distinguish between the questions which it can understand and those it cannot. In the comparative evaluation with the Inductive Learning Programming (ILP) by Tang and Mooney [2001] and also with the Microsoft's English Query (EQ), using Mooney GeoQuery, Job and Restaurant datasets, PRECISE made no errors thus outperforming the ILP approach, while both systems were significantly better than Microsoft's EQ.

Precision was also a concern for the NLIDB that assists users by offering auto complete options while users are entering the text presented in Hallett [2006]. Users are guided through the available kinds of questions that can be handled by the system. In a way, users are limited because questions are chosen from the finite set of inferred queries. Their tool automatically infers the set of possible queries that can apply to a given database. On the other hand, the idea of helping users with available options for a query appears to be promising, especially in cases when the user is not familiar with the domain knowledge.

Finally, recent years have seen online systems such as Wolfram Alpha (`http://www.wolframalpha.com/`) answering factual queries directly by comput-

ing the answer from the structured data, rather than providing a list of documents or web pages that might contain the answer as a search engine would. Another system of this kind is Powerset (`http://www.powerset.com/`). On May 11, 2008, the company unveiled a tool for searching a fixed subset of Wikipedia using conversational phrases rather than keywords. On July 1, 2008, it was purchased by Microsoft. Another recent online service of this kind is TrueKnowledge (`http://www.trueknowledge.com`) which is powered by semantic technologies. Answers are found in two main sources: information that has been imported initially and facts added by users.

With regard to architecture of NLIDBs, there are several kinds of systems:

- *Pattern-matching systems* are simple to design but prone to return incomplete answers if they recognize certain patterns but not the whole sentences. For instance, in *What is the capital of France?* if the system does not have a pattern for *capital* followed by *Country*, but has a pattern for *capital* only, it might list all capitals it has in its knowledge base.

- *Syntax-based systems* map the parse tree of the question into the formal language directly.

- *Semantic grammar systems* map the parse tree into the formal language, but the non-leaf trees are not referring to syntactic but semantic concepts. The advantage of such systems is that they can have a high performance, but the downside is that they are not easily portable.

- *Intermediate representation language systems* translate a NL query into an intermediate logical query, expressed in some internal meaning representation language and independent of the database structure. This logical query is then translated into the formal query language. In such systems, the syntax rules which link non-leaf nodes in the parse tree into the semantic rules is usually domain-independent, while the leaf nodes and logic expressions corresponding to them are domain-dependent, and are found in the lexicon.

3.2 Open-domain Question-Answering Systems

Studying Question-Answering (QA) systems goes back to as far as 1965, when Simmons [1965] reviewed the existing question-answering systems for English, which had been developed in the period between 1960 and 1965 [Greenwood, 2006][p.11]. After almost fifty years, the problem of how to automatically answer questions, similar to how would human do it remains challenging. While these early question-answering systems were extracting answers from the database by *computing* it (e.g., LUNAR and others mentioned in the Section 3.1), open-domain question-answering systems came later and were *finding* the answer in free-text (a set of documents) [Webb and Webber, 2009].

One of the first systems was MURAX [Kupiec, 1993], which was capable of answering *closed-class questions*. A close-class question is a direct question whose answer is assumed to lie in a set of objects and is expressible as a noun phrase. e.g. *Who's won the most Oscars for costume design?*. The answers are found in Grolier's on-line encyclopaedia[2]. MURAX was evaluated with 70 questions, which are factoid questions starting with *What* or *who*. The correctness was 53%.

With the emergence of the WWW, the popularity of open-domain QA systems has increased. Unlike Information Retrieval systems, such as Google, which search for the list of *relevant documents* to the user's query, QA systems search for the *answer* to the user's question (which might be located within the collection of relevant documents).

However, majority of QA systems do rely on the Information Retrieval components. Question-Answering systems usually consist of three modules: a question processing module, a document processing module, and an answer extraction module. The task of returning a concise answer from a set of documents is different from that in Information Retrieval, or Information Extraction, but requires a combination of the two and depends on both [Strzalkowski and Harabagiu, 2006]. In the *question processing module*, the question is translated into a set of keywords which are then passed on to an Information Retrieval engine to retrieve relevant documents (those that

[2] `http://teacher.scholastic.com/products/grolier/index.htm`

may contain the answer). The *document processing module* then identifies the passages in these relevant documents, in which the answer is most likely to be found. Finally, the *answer extraction module* extracts the snippet which represents the answer to the posed question.

Most QA systems contain a classifier module which detects a question category, based on which, each question is assigned an *answer type*. This classification often relies on Machine Learning approaches, which require a large amounts of data in order to work effectively. Moreover, classical question-answering approaches do not often apply to all domains. For example, in Niu et al. [2003] the differences between general and medical QA are outlined.

Evaluation of open-domain QA systems was the subject of the competitive evaluation TREC, Text Retrieval Conference (`http://trec.nist.gov`). This initiative started in 1999 with TREC-8, and ran on a yearly cycle until 2007. Participating systems have been given a corpus and a set of questions, for which they had to return the two types of answer lengths: 250 and 50 bytes. The former one was usually easier as all evaluated systems had the better performance when returning the larger snippet of text. Clearly, these were not always exact answers but more snippets of text which contained the answer. This changed in TREC 2002, when the additional requirement was that the systems must return the exact answer [Voorhees, 2003].

The results of the first large-scale evaluation of QA systems in TREC-8 was with fact-based, short-answer questions [Voorhees, 1999]. The most accurate system in the more difficult 50-bytes run, was from Cymfony, Inc. which returned the answer in 72.72 % (144 out of 198 questions) of the cases. The Mean Reciprocial Rank of 0.66 indicates that if the answer was returned it was almost always correct. A very similar performance had the system from Southern Methodist University, returning an answer in 68.18 % of the cases, with an MRR 0.555.

TREC-9 was similar to the previous track, with the main difference in the number of questions (500, plus 198 reformulated questions) and also in the way the questions are obtained. While for the TREC-8, the questions were artificially generated by humans looking into documents and generating questions, the TREC-9 questions were extracted from various user logs such

as that from Excite[3]. The best system was Falcon from Southern Methodist University which answered 65% of questions correctly, with other systems ranked much lower at 42%. The lower performance than in the previous year is not suprising – given that the tasks were harder; questions were not generated artificially but collected from the users [Voorhees, 2000].

Each year, TREC QA Track was designed to be more challenging and to bring new previously unseen complexities such as the inclusion of questions where the answer does not have to be in the collection in TREC-10 (2001) and to include questions for which the answer is scattered among several documents [Voorhees, 2001]; the requirement that the answer is exact answer and not the text snippet which contains the answer in TREC 2002; and also the consideration of *context questions*, see Figure 3.2.

CTX3a What grape variety is used in Chateau Petrus Bordeaux?

CTX3b How much did the futures cost for the 1989 vintage?

CTX3c Where did the winery's owner go to college?

CTX3d What California winery does he own?

Figure 3.2: A sample *context question* introduced in TREC 2002 (the example from [Voorhees, 2002])

TREC 2003 contained list and definition questions [Voorhees, 2003]. In 2004, the addition was the grouping of the factoid and list questions into different series, where each series is associated with a target and the question in the series is asking for some information about the target. The target was a person, an organization, or a thing that was a plausible match for the scenario assumed for the task. An example of series questions is illustrated in Figure 3.3.

A new addition in TREC 2005 was the new target featuring events, and the new tasks for document ranking and relationship retrieval. TREC 2006 systems were required to give the most up-to-date answer found in the corpus. This restriction was more in line with the real world, where users would want the *best*, and not just *any* answer to their question. Relationship

[3]Excite was one of the first companies to become famous on the Web in the "dotcom" boom, together with Yahoo, Lycos, and Netscape. Today it is a portal offering many services not only search: `http://www.excite.com/`

21	Club Med		
	21.1	FACTOID	How many Club Med vacation spots are there worldwide?
	21.2	LIST	List the spots in the United States.
	21.3	FACTOID	Where is an adults-only Club Med?
	21.4	OTHER	
22	Franz Kafka		
	22.1	FACTOID	Where was Franz Kafka born?
	22.2	FACTOID	When was he born?
	22.3	FACTOID	What is his ethnic background?
	22.4	LIST	What books did he author?
	22.5	OTHER	

Figure 3.3: Sample series questions with the target Organization and Person (the example from [Voorhees, 2004])

task from the TREC 2005 was extended with the interactive QA task, with the idea *"to push the frontiers of question answering away from* factoid *questions towards more complex information needs that exist within richer user contexts, and to move away from the one-shot interaction model implicit in previous evaluations towards a model based at least in part on interactions with users"* [Dang et al., 2006][p.2].

The main task in the TREC 2007 QA Track repeated the question series format, however the corpus was not only newswire, but included blogs. Mining blogs for answers introduced significant new challenges in at least two aspects that are crucial for functional QA systems: 1) being able to handle language that is not well-formed, and 2) dealing with discourse structures that are more informal and less reliable than newswire [Dang et al., 2007]. In addition to the main task, the TREC 2007 QA track repeated the complex, interactive QA (ciQA) task of TREC 2006.

In 2008, TREC QA Track was replaced by TAC QA Track – TAC is sponsored by NIST and other U.S. government agencies and is overseen by an Advisory Committee consisting of representatives from government, industry, and academia. The focus of this track was answering opinion-related questions and also questions requiring summarization [Dang, 2008]. The track was not run in 2009, neither was the call announced for 2010.

Performance-wise, TREC evaluation reveals that there is a huge gap between the best performing system and the rest of the field. The number of correctly answered questions was usually around 70% for most of the TREC

QA tracks, with the best system reaching 76% and 83% for the main task (factoid questions) in TREC 2001 and TREC 2002, respectively. While factoid questions generally reach the highest performance (e.g. 0.76 in TREC 2003), the other types of questions such as definition and list perform much worse reaching only 0.396 and 0.442 in 2003, respectively. The drop in accuracy of factoid questions occurred in TREC 2006 by introducing the task of returning the most up-to-date answer. It increased to the accuracy of 0.706 for the best performing system (LCC) in the next year (TREC 2007), while other tasks remained challenging. These include interactive Question-Answering and also opinion-focused QA Track which was the focus of the TAC 2008.

Also, the TREC evaluation did not consider the time to return the answer in its results [Greenwood, 2006]. At TREC 2004, Language Computer Corporation (LCC) which developed PowerAnswer – a very well performing QA system, introduced Palantir – a system which was answering questions in under 20 seconds, and it was used in their dialogue system Ferret. Forcing Palantir to return an answer within 20 seconds involves a trade-off between the precision of the answer and the speed with which the answer is returned [Greenwood, 2006]. When Palantir was compared with PowerAnswer, the factoid score dropped from 0.77 to 0.339 and the overall combined score dropped from 0.6 to 0.254 over the TREC 2004 test set, highlighting the effect of attempting to show results in real time.

3.3 Interactive Natural Language Interface Systems

Answering complex questions in open-domain Question-Answering requires sophisticated approaches, which might involve hypotheses and the consequences, analogies and comparisons, which are not usually found in simple factoid or definition questions, and moreover cannot be answered through one or more independent questions [Webb and Webber, 2009]. While involving the user in dialog with the system in Question-Answering has not been researched extensively, the idea of involving the user in dialog is almost as old as the Question-Answering systems themselves. One of the earliest sys-

tems called SHRDLU [Winograd, 1972] was designed in the late 1960s and 1970s, where the user is engaged in dialog (through text) with an animated or robotic agent that is capable of performing limited set of actions and also which is aware of its own behaviour, including this interaction. More recent dialogue systems, although with some limitations, support speech [Allen et al., 1996].

A sample conversation[4] with SHRDLU looks like the following:

```
Person: PICK UP A BIG RED BLOCK.
Computer: OK. (does it)
Person: GRASP THE PYRAMID.
Computer: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Person: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU
        ARE HOLDING AND PUT IT INTO THE BOX.
Computer: BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH
          IS TALLER THAN THE ONE I AM HOLDING.
Computer: OK. (does it)
Person: WHAT DOES THE BOX CONTAIN?
Computer: THE BLUE PYRAMID AND THE BLUE BLOCK.
Person: WHAT IS THE PYRAMID SUPPORTED BY?
Computer: THE BOX.
Person: HOW MANY BLOCKS ARE NOT IN THE BOX?
Computer: FOUR OF THEM.
Person: IS AT LEAST ONE OF THEM NARROWER THAN THE ONE
        WHICH I TOLD YOU TO PICK UP?
Computer: YES, THE RED CUBE.
Person: IS IT SUPPORTED?
Computer: YES, BY THE TABLE.
```

In this dialogue, the user is asking or telling the system to perform some actions ("Grasp the pyramid.") and the system is either accepting the action, or rejecting it by providing an explanation ("I don't understand which pyramid you mean").

[4] Taken from the demo available at `http://hci.stanford.edu/winograd/shrdlu/`

Jupiter [Zue et al., 2000] is another dialogue system developed by MIT, which can answer the questions about the weather forecast on the phone. More information, including the number which one can dial in order to converse with Jupiter is available from: `http://groups.csail.mit.edu/sls/research/jupiter.shtml`. According to this Website "Since May 1997, Jupiter has fielded over 300,000 calls, achieving average word accuracies of over 90% on in-domain queries for novice users (over 98% for experienced users), and correct understanding of about 80% of in-domain queries for novice users (over 95% for experienced users). The system has a vocabulary of just over 2000 words.". A sample session with Jupiter is shown in Figure 3.4.

JUPITER:	Hi! Welcome to JUPITER, the MIT Lab for Computer Science weather information system. This call is being recorded for system development. You may hang up or ask for help at any time. How can I help you?
User:	CAN YOU TELL ME WHAT'S THE WEATHER LIKE IN BOSTON TODAY
JUPITER:	In Boston this afternoon. Partly sunny early. Then becoming cloudy with 30 percent chance of showers toward evening. Tonight: rain. Possibly heavy at times. Near 100 percent chance of rain. Can I help you with something else?
User:	YES WHAT ABOUT SAN FRANCISCO
JUPITER:	In San Francisco today: patchy fog near the ocean morning. Otherwise mostly sunny. Tonight: areas of coastal fog and low clouds. Otherwise fair. Is there something else I can do for you?
User:	YES HOW COLD IS IT IN DALLAS TEXAS TODAY
JUPITER:	In Dallas today: high upper 40s. Tonight: lows near 32. What else would you like to know?
User:	I'D LIKE TO KNOW WHETHER IT'S SNOWING IN DETROIT
JUPITER:	I'm sorry, I didn't understand you. Please rephrase your query.
User:	WHAT IS THE WEATHER IN DETROIT
JUPITER:	In Detroit this afternoon: occasional light rain and drizzle. Tonight: rain showers likely. Becoming snow showers by late tonight. 70 percent chance of precipitation.

Figure 3.4: A sample session with Jupiter, taken from Zue et al. [2000][p.101]

Note that for the understanding some of the questions, the system must connect it to the previous ones. Similar system called Mercury (developed also by MIT, available from `http://groups.csail.mit.edu/sls/research/mercury.shtml`) provides information about flight schedules and pricing. Mercury enables users to book and price complex multi-leg travel itineraries to over 200 cities within the United States and around the world. A sample conversation with Mercury is available from `http://groups.csail.mit.edu/sls/research/mercury.wav`.

While in this type of systems, there are no specific goals, the recent work in this area is concerned with so called *tutoring* systems. The goal of tutoring systems is very concretely specified, and these systems are used to assess the student's knowledge, or to correct the student's errors. One of the well known systems which also allows spoken interaction is ITSPOKE [Litman and Silliman, 2004, Litman and Forbes-Riley, 2006]. One sample session looks like in Figure 3.5, with identified emotional states of the students shown in red square brackets (the example is taken from Litman [2006]): More informa-

tion is avaliable at `http://www.cs.pitt.edu/~litman/itspoke.html`.

ITSPOKE: What else do you need to know to find the box's acceleration?
Student: the direction *[UNCERTAIN]*
ITSPOKE : If you see a body accelerate, what caused that acceleration?
Student: force *[CERTAIN]*
ITSPOKE : Good job. Say there is only one force acting on the box. How is this force, the box's mass, and its acceleration related?
Student: velocity *[UNCERTAIN]*
ITSPOKE : Could you please repeat that?
Student: velocity *[ANNOYED]*

Figure 3.5: An example session with ITSPOKE

Developing dialogue systems is among the most researched topic in Artificial Intelligence, and one sample list of such systems can be found at `http://www.ling.gu.se/~sl/dialogue_links.html`.

3.4 Summary and Discussion

As noted by Androutsopoulos et al. [1995], the most challenging problems with regard to the linguistic features of NLIDBs (and for any kind of system which deals with Natural Language understanding) is related to **correctly handling**:

- *nominal compounds*,
- *anaphora*,
- *disjunction and conjunction*,
- *quantifiers*,
- *modifiers*,
- *elliptical sentences*: as a follow up to the already asked questions which contain enough information for the context, so that the user can follow up with small, incomplete questions which could still be understood,

- *extragrammatical utterances* – namely linguistic theories describe the structure and meaning of grammatically correct utterances. However, every-day language often contains syntactically ill-formed input. If the main goal of an NLI is to assist the user, then the system must be able to understand the user's requests, even when they are ill-formed.

According to Androutsopoulos et al. [1995], an advantage of NLIs as opposed to other kinds of interfaces such as form-based is that there is no learning overhead because the language is not artificial. This is at the same time a disadvantage, as NLIs usually rely on a controlled language which has to be learnt by the user. If the user is not very familiar with the required controlled language, he might not be able to judge whether the system did not return an answer because there is no information about the concepts in the knowledge base, or the query was not properly formulated. In addition, while natural language is potentially easier to use than formal query languages, it is still prone to errors (e.g., misspellings). In comparison to alternative ways of searching, natural language provides an easy way to express some questions including those using negation and quantification. On the negative side, it is usually unclear to the user what the linguistic coverage of the supported language is. Another problem noted by Androutsopoulos et al. [1995] is known as *linguistic vs. conceptual failure* which means that it is not often the case that the NLIDB gives a clear message to the user in terms of whether the failure to return a result was due to the system not having the information about it (conceptual failure), or due to the system requiring the reformulation of the question (linguistic failure). Further disadvantages include: users assumptions that the system is intelligent, ambiguity of NL, and tedious configuration.

In recent years, the popularity of NLIDBs and even open-domain Question-Answering systems is replaced to some extent by the new kind of NLIs – those which are finding answers in an ontology or a set of ontologies. As this is quite a young research topic which has been around for less than a decade, it can also be seen as a continuation of the work which has been researched for more than five decades now. There are many similarities between all systems which have been discussed mainly related to ways of solving the language complexity problem. The advantages brought by the NLIs to

ontologies are related to the possibility to link the word meanings, inherit the relationships based on the existing structure and deal with ambiguities more effectively. Moreover, in comparison to NLIDBs, these new systems have been promoting the benefits of **reasoning** over structured data, and **portability** – extracting the lexicon from the ontology directly, without any need for customisation.

In other words, as noted by Grosz et al. [1987] who developed the TEAM system, one feature which is missing in NLIDBs is what makes NLIs to ontologies attractive:

> [TEAM] shares such constraints of customized interfaces as being restricted to single queries and being able only to *retrieve* the facts from a database, not to *reason* about them. [Grosz et al., 1987][p. 237]

A good example to demonstrate the reasoning was given by Professor Daniel Weld from the University of Washington, during his invited talk at K-CAP 2009 [Weld, 2009]:

```
what vegetable prevents osteoporosis?
```

If we enter this query into Google, there will be no answer (or rather, no documents which contain the answer). The answer can be found in the documents available on the Web, however, Information Retrieval engines can not locate it as they do not implement reasoning. Namely, *kale* is a vegetable which prevents osteoporosis – but no such documents exist on the Web which mention this, however, there are documents which mention the following:

```
kale is a vegetable           (1)
kale contains calcium         (2)
calcium prevents osteoporosis (3)
```

NLI systems which interface ontologies are built to answer these and similar kinds of questions.

Another important advantage of NLIs to ontologies is **interoperability** - the possibility to easily combine and merge resources from various locations

on the Web. For the example above, statements 1, 2 and 3, could or could not be contained within one single resource on the Web. Therefore, the knowledge which has been collected for decades can now be merged in order to successfully accomplish what has been a great challenge for such a long time: answer the questions automatically using the distributed sources available on the Web. This has not been possible with databases as they are distributed over the Web and not interoperable, while Question-Answering systems have to process large amounts of unstructured text and use techniques such as Information Retrieval to locate the documents in which the answer may appear. This step can be misleading as Information Retrieval methods although scale well, do not often capture enough semantics — documents with the answer could be easily disregarded if the answer was hidden in a form which is not in-line with the patterns expected by the QA systems. Finally, NLIs to ontologies can also use the techniques applied in interactive NLI systems in order improve the user's experience. Instead of a single question session, they can move towards conversational systems which can give answers simulating the human, but not being restricted to a topic or a domain.

Chapter 4

Evaluation of Natural Language Interfaces

In this chapter[1] we discuss measures used to evaluate NLI systems. First we discuss *habitability* which gives an indication of how much effort is required for users to make use of the language supported by the system (Section 4.1). By identifying the habitability problems (e.g. difficulties that the user faces) we can focus on correcting these to increase usability. Measuring usability is described in Section 4.2.

[1]The content of this chapter is an updated and extended version of Section 2 in *D. Damljanovic, K. Bontcheva: Towards Enhanced Usability of Natural Language Interfaces to Knowledge Bases. In V. Devedzic and D. Gasevic (Eds.), Special issue on Semantic Web and Web 2.0, Annals of Information systems, Springer-Verlag, 2009.* I am grateful to the contribution of K. Bontcheva who read and commented on the initial version of the paper and suggested improvements.

The discussion about *habitability* from this chapter is contributed to *A. Wyner, K. Angelov, G. Barzdins, D. Damljanovic, B. Davis, N. Fuchs, S. Hoefler, K. Jones, K. Kaljurand, T. Kuhn, M. Luts, J. Pool, M. Rosner, R. Schwitter, J. Sowa:* On Controlled Natural Languages: Properties and Prospects. *In N. Fuchs, ed.: Controlled Natural Language. Volume 5972 of Lecture Notes in Computer Science, pp. 281–289, Springer Berlin/Heidelberg, 2010*, which is an outcome of a collaboration amongst the listed contributors during the CNL'09 workshop. The publication is largely based on the original collaborative document accessible from: `http://docs.google.com/Doc?id=dd3zb82w_03976bbfm`. My contribution to that document is largely based on the work described in this thesis.

4.1 Habitability

NLIs were invented to assist communication between users and computers. However, some studies ([Chin, 1984], [Krause, 1980]) show that users behave differently when communicating with computers than with humans. In the latter case, their conversation relies heavily on context, whereas with a computer the language they use is restricted as they are making assumptions about what computers can and cannot understand [Ogden and Bernick, 1997].

One particular approach to the human-computer communication problem is to keep it brief and use restricted natural language syntax [Malhotra, 1975]. However, a big challenge when restricting the vocabulary of an NLI system is its habitability. *Habitability* is a term coined in 1965 by Watt [1968] – it indicates how easily, naturally, and effectively users can use a language to express themselves within the constraints imposed by the system. If users can express everything they need for their tasks, using the constrained system language, then such NLIs are considered *habitable* [Ogden and Bernick, 1997]. In other words, habitable languages are languages that people can use fluently [Epstein, 1985]. According to Epstein [1985], a language is habitable if 1) users are able to construct expressions of the language which they have not previously encountered, without significant conscious effort; and 2) users are able to avoid easily constructing expressions that are not part of the language. Another way of viewing habitability is as the mismatch between user expectations and the capabilities of an NLI system [Bernstein and Kaufmann, 2006].

Ogden and Bernick [1997] describe habitability in the context of four domains:

The conceptual domain of the language supported by the system describes the area of its coverage, and defines the complete set of objects and the actions which are covered. In other words, conceptual domain determines *what* can be expressed. This means that this domain is satisfied if the user does not ask about the concepts which can not be processed by the system. To cite the example from Ogden and Bernick [1997], the user should not ask *What is the salary of John Smith's*

manager? if there is no information about *managers* in the system. The conceptual domain of the language can be expanded to inform the user that there is no information about managers in the system.

The functional domain determines *how* a query to the system can be expressed. Natural Language allows different ways of expressing the same fact, especially taking into the account the knowledge of the listener and the context. The functional domain is determined by the number of built-in functions or knowledge the system has available. If, for example, the answer to a question requires combining several knowledge sources, the system itself might not be able to answer it and would require the user to ask two questions instead of one. A habitable system provides the functions that the user expects. Note that this is different from rephrasing the question in order to get the answer, which is related to the syntactic domain.

The syntactic domain of a language is determined by the number of paraphrases of a single command that the system understands. For example, the system might not be able to understand the question *What is the salary of John Smith's manager?* but, could be able to process a rephrased one such as *What is the salary of the manager of John Smith?*.

The lexical domain is determined by the words available in the lexicon. For example, in order to improve the coverage, many systems extend their lexicon through the use of external sources for finding synonyms.

In order for an NLI to be considered habitable, it should cover all four domains. As mentioned by Ogden and Bernick [1997], the most habitable NLI would be the one capable of passing a Turing Test or winning the Loebner Prize[2]. An interesting fact is that some NLIs have been quite habitable without making use of any Natural Language Processing technologies. For example, COMODA is a conversational natural language information system for publicly distributing information about the disease AIDS to the public [Patrick and Whalen, 1992]. This system won the Loebner's prize by being

[2]Winning Loebner's prize involves convincing users that they are conversing with another human, when, in fact, they are communicating with a computer.

able to process the actual words and phrases people used when discussing the topic.

Habitability is an important aspect of a system to measure because it can affect the usability of NLIs. By identifying why the system fails to be habitable, we can identify the ways to improve them [Ogden et al., 2006].

4.2 Usability

According to Brooke [1996], *usability* can be defined as "being a general quality of the appropriateness to a purpose of any particular artefact". In other words, usability is evaluated in the context in which an NLI system is used, by measuring its appropriateness for that context. Firstly, it is important to identify the system's target users, and secondly – the tasks that these users will have to perform.

NLIs are used by the two types of users:

- *application developers* who are responsible for porting a system to a specific domain, and whose task is to customise the system to work with that domain (if the system requires customisation); and
- *end-users* who are querying the customised systems in order to retrieve domain knowledge (e.g., domain experts).

Therefore, the usability of NLI systems should be evaluated from the point of view of these two types of users. According to ISO 9241-11, usability measures should cover [Brooke, 1996]:

1. *effectiveness* – the ability of users to complete tasks using the system, and the quality of output of these tasks;
2. *efficiency* – the level of resource consumed in performing tasks; and
3. *satisfaction* – users' subjective reactions to using the system.

4.2.1 Effectiveness

Customisation

Effectiveness with regard to the customisation of an NLI system is determined by the ability of application developers to complete the *customisation process* successfully, and also by the quality of output of this customisation[3]. Usually, the customisation process includes creating a domain-specific lexicon when it is ported from one domain to another. The quality of the output can be measured through the *coverage* of the system. Given a set of questions collected from a real-world application, the percentage of those which are answerable (e.g., covered by the domain lexicon) describes the system's coverage. The richer the lexicon is, the higher the value for the coverage. This term should not be confused with the *language coverage*, which usually refers to the complexity of questions covered by an NLI system.

End-user's point of view

From the end-user's perspective, effectiveness indicates whether they could find the answer to their question using the system, and also whether the answer was correct. Typically NLI systems are evaluated in terms of precision and recall, which are measures adapted from information retrieval. *Precision* measures the number of questions correctly answered divided by the number of questions for which some answer is returned [Tang and Mooney, 2001], [Cimiano et al., 2007]. The definition of recall varies and the most widely used are as follows:

- According to Tang and Mooney [2001], *recall* is defined as the number of correctly produced formal queries, divided by the total number of questions. This definition is also used by Cimiano et al. [2007].
- According to Popescu et al. [2003], recall is interpreted as the number of questions **answered** by an NLI system, divided by the total number of questions. While this definition is different from the previous one, in the evaluation of the system described by Popescu et al. [2003], **the**

[3]Note that this is appropriate only for the systems which can be customised or ported to other domains.

number of questions answered is equal to the number of those correctly answered, as the precision is 100%.

However, the second definition is used for the evaluation of several NLIs to ontologies where **the number of answered is not equal to the number of correctly answered**, for example in Wang et al. [2007] and Kaufmann and Bernstein [2007]. For the evaluation of the systems developed as a part of this thesis, we use the first definition.

4.2.2 Efficiency

Efficiency refers to the level of resources consumed in order to perform the specific task, e.g. how fast a user can accomplish a task. In the case of NLI users, this is usually measured by the time needed to customise the system for a specific domain (the developer's point of view), or by the time needed to successfully find some particular information (the end-user's point of view). In the letter case, the efficiency is usually expressed by the execution time for queries of various complexity.

4.2.3 User Satisfaction

There is no definitive way of measuring user satisfaction. The most common methodology is to engage users into a session with the system, and ask them to fill out a questionnaire where they can express their views on the different features of the system. One of the most popular questionnaires used for evaluating different interfaces is SUS - System Usability Scale – a simple ten-item scale giving a global view of subjective assessments of usability [Brooke, 1996].

Before conducting a user-centric evaluation, each system should undergo a laboratory evaluation. One of the most popular types is a *heuristic evaluation* which has been proven to give significant results even with a small number of users (e.g., evaluation of TAP Search Interface presented in da Costa et al. [2005]). Heuristic evaluation is the analysis that utilises history and experience to discover problems with particular user interface designs. It requires experimenting with individual components and noting any disparities

between component design and the suggested user interface design principles. A very effective method for heuristic evaluation is summarised in the ten general principles for user interface design presented in Neilsen [1994].

Other measures of effectiveness and user satisfaction are often related to the specific methods used in NLI systems. The majority of systems express the effectiveness of individual methods by comparing the system's performance with and without the specific method. For example, to test whether the *query refinement* module has any effect on performance, it is usually compared to the baseline model of the system – the one without the query refinement.

More details about the evaluation of NLI systems is given in Ogden and Bernick [1997] and also in Ogden et al. [2006].

4.3 Summary

Natural Language Interfaces are typically used by two types of users: *application developers* who customise the system, and *end-users* who query the customised system. Different evaluation measures are used to test *usability* from different users' perspectives, relative to the tasks the users perform. Irrespective of the task type, usability measures should cover: *effectiveness* – whether the users can finish tasks successfully using the system or not, *efficiency* – how quickly they can finish tasks, and *user satisfaction* – the user's subjective reactions to using the system.

Common evaluation strategies with regard to Natural Language Interfaces are related to *habitability* which reflects *usability* from the end-user's point of view – if the end-users can use the system effectively and easily avoid the constructions that are not supported then the system is considered *habitable*. *Effectiveness* can be measured through the number of correctly handled questions, and in that respect *precision* and *recall* measures are commonly used. Usability includes other aspects as well and these are related to *efficiency* – how quickly the users can find the answers to their questions, but also *user satisfaction* which is usually measured through questionnaires. The most popular questionnaire for measuring user satisfaction is System Usability Scale.

With regard to the usability from the application developers' point of view, the measures are less standardised and include *effectiveness* – measuring whether the users can customise the system for the particular domain successfully or not, *efficiency* – how quickly they can customise it, and also the *user satisfaction* based on how they like the customisation interface.

Part II

Usability of Natural Language Interfaces to Conceptual Models: State of the Art

Chapter 5

Portability of Natural Language Interfaces to Structured Data

In this chapter[1] we review existing Natural Language Interfaces to conceptual models with special emphasis on their customisation.

5.1 Introduction

Building portable NLIs is a very challenging task, and, as we have discussed previously, has been addressed in several different ways in previous work. One of the first attempts to enable portability was in the 1980s when several NLIDBs were developed. One of them is TEAM [Grosz et al., 1987], which is envisaged to be used by

- a *database expert* who customises the system through an *acquisition dialogue* in order to port it to a new database. The customisation

[1]The content of this chapter is an updated and extended version of Section 3 in *D. Damljanovic, K. Bontcheva: Towards Enhanced Usability of Natural Language Interfaces to Knowledge Bases. In V. Devedzic and D. Gasevic (Eds.), Special issue on Semantic Web and Web 2.0, Annals of Information systems, Springer-Verlag, 2009.* I am grateful to the contribution of K. Bontcheva who read and commented on the initial version of the paper and suggested improvements.

includes extending the lexicon by adding new verbs, adjectives or synonyms for existing words in the database. In addition, it includes adding the information about the fields in the database and the conceptual content they encode, and also the words and phrases used to refer to these concepts. Therefore, a *database expert* must be familiar with the database structure and also with the domain that it covers, but he does not need any knowledge about language-processing terminology;

- *end users* who query the database.

There are two lexicons used by the TEAM system:

- *An open-class words* lexicon includes domain-specific words such as nouns, adjectives and verbs. This lexicon is enriched through the acquisition dialogue with a domain expert.
- *A closed-class* lexicon is built-in to the system as the initial lexicon of TEAM and it does not depend on the domain. This lexicon includes determiners, pronouns and conjunctions.

An example of acquisition dialog in TEAM is shown in Figure 5.1.

The difference between the approaches for portable systems such as TEAM and CHAT-80 [Warren and Pereira, 1982], and the domain-dependent ones (such as those developed in the 1970s, e.g. LUNAR) is that the former do not directly translate NL into a formal language, but instead use an intermediate logical representation which is subsequently translated into the formal language. This logical representation allows for more generality and the possibility of designing more portable systems.

Unlike with relational databases, where it is difficult, although not impossible, to attach metadata to the fields in the tables (due to implementation dependent features being present), with ontologies this is more natural. Each concept and each relation in the ontology is envisaged to be accompanied by a human-understandable label which describes the concept (or a relation). Therefore, addressing portability in NLIs to ontologies does not seem to be a big issue at first sight and many systems have been developed with the

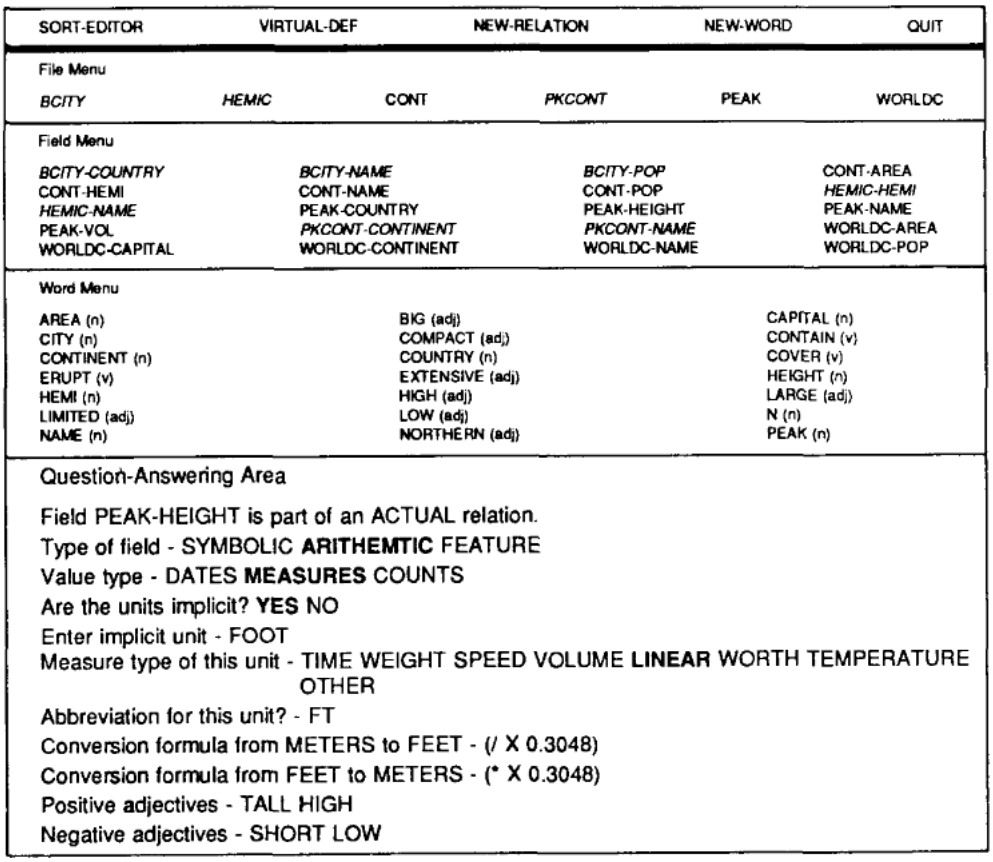

Figure 5.1: An acquisition dialogue from TEAM (taken from [Grosz et al., 1987][p.10])

claim that they are portable. However, especially for ontologies which are generated automatically, many concepts can have missing labels or too many labels, which again complicates the issue.

In what follows, we review and compare some of the existing NLIs to ontologies, emphasising customisation's effect on performance. This comparison is not a trivial task, due to the variation in evaluation conditions (e.g., ontologies) and measures used. To begin with, the datasets used to evaluate the different systems are not the same and their size, coverage, and quality varies. In addition, benchmark queries are of a different complexity. Overall, these differences make comparative system evaluation somewhat unreliable, because the evaluation metrics and, consequently, the reported system results, are heavily dependent on which datasets are used and how difficult the queries are. Nevertheless, these results still provide an insight into the achievements in the field.

A brief overall summary is shown in Table 5.1, subdivided by dataset, as no reliable comparison of precision and recall can be made across different datasets. This table only covers a sub-set of NLI systems to ontologies, i.e., those that reported evaluation results. The main conclusion to be drawn from this table is that although systems with zero customisation tend to have reasonable performance, it varies significantly across systems – in general, the more complex the supported queries are, the lower the performance is.

Table 5.1: Natural Language Interfaces to Knowledge Bases

Dataset	**System**	**Precision**	**Recall**	**Customis.**
Mooney: geography	PANTO	88.05%	85.86%	zero
	Querix	86.08%	87.11%	zero
	NLP-Reduce	70.7%	76.4%	zero
Mooney: restaurants	PANTO	90.87%	96.64%	zero
	NLP-Reduce	67.7%	69.6%	zero
Mooney: jobs	PANTO	86.12%	89.17%	zero
Geographical facts about Germany	ORAKEL	80.60-84.23%	45.15%-53.7%	customised
library data	E-librarian	97%	-	-
biology	CPL	38%	-	-
chemistry	CPL	37.5%	-	-
physics	CPL	19%	-	-

5.2 ORAKEL

ORAKEL is an NLI to knowledge bases [Cimiano et al., 2007] which supports factual questions, starting with WH-pronouns such as who, what, where, etc. Factual here means that answers are ground facts as found in the knowledge base, and not complex answers to *why* or *how* questions that require explanation. The most important advantage of ORAKEL in comparison to other similar systems is its support for compositional semantic construction i.e. the ability to handle questions involving quantification, conjunction and negation.

ORAKEL's lexicon is composed of two parts:

- *General lexicon* which is shared among different domains, where words such as *what*, *which*, etc. are stored.
- *Domain-specific lexicon* which has two parts:
  - *Ontological lexicon* generated automatically from the domain ontology. It contains lexical entries and the semantics of instances and concepts which are typically represented by proper nouns and nouns respectively.
  - *Lexicon for mapping ontology relations to words*: this part is created manually and contains mappings of *subcategorisation frames* to ontology relations. Subcategorisation frames are essentially linguistic argument structures, e.g. verbs with their arguments, nouns with their arguments, etc. For example, the verb *to write* requires a subject and an object, as it is a transitive verb. This *triple* of subject-verb-object in this case could be considered a subcategorisation frame, and could be mapped to the ontology relation *writes*. Subcategorisation frames are created by the domain experts who do not have to be familiar with computational linguistics, although they are expected to have some very basic knowledge of subcategorisation frames. The adaptation is performed in several iterative cycles through the user interaction sessions. In this way, the *coverage* of the lexicon is being increased with each iteration.

In the user study carried out by Cimiano et al. [2007] the aim was to test whether it is feasible for users without NLP expertise to customise the system without significant problems. The evaluation knowledge base contained geographical facts about Germany, covering 260 entities in total. The experiment was conducted with 27 users. Three people had to customise the lexicon, while the remaining 24, who did not have any background knowledge in computational linguistics, received a brief explanation about the scope of the covered domain and were told to ask at least 10 questions; they also had to explicitly say if the received answer was correct or not.

Only one of the three people in charge of creating the domain lexicon, was very familiar with the lexicon acquisition tool (user A), while the other two users (user B and user C) were not and received 10 minutes of training on the software (the FrameMapper tool) and 10 minutes explanation about the different subcategorisation types, illustrated with examples. User A constructed the lexicon in one iteration, whereas users B and C constructed it in two rounds, each lasting 30 minutes. In the first round they created the model from scratch, while in the second round they were presented with those questions which the system had failed to answer in the sessions with the 24 users. Overall, users B and C had one hour each to construct the lexicon. The customisation system of ORAKEL is designed so that in each iteration, the created lexicon is extended and therefore the system is expected to give better performance. Consequently, the more time users spend customising the system, the better the performance of the system is expected to be.

The results showed that querying the system using the lexicons created by users B and C gives comparable precision and recall to that of the system using the lexicon created by user A. Namely, after the second iteration, recall for users B and C was 45.15% and 47.66% respectively, in contrast to the recall when using the user A created lexicon which was 53.67%. Precision varied from 80.95% (user B) to 84.23% (user A).

One weak point of the approach implemented in ORAKEL is that it maps ontology relations to words. This approach assumes that all classes and instances have understandable and useful lexicalisations, which is not always the case. Moreover, while the user interaction is used for customisation, with regard to the end users, the system either interprets the question and returns

the answer, or it fails. Hence, the end-user has no control over the overall process of interpreting the NL into the formal language.

5.3 AquaLog and PowerAqua

AquaLog [Lopez and Motta, 2004] is a portable question-answering system which takes Natural Language queries and an ontology as input, and returns answers as output. The supported questions are mainly factual queries beginning with *what*, *which*, *who* and the like.

Although customisation of AquaLog is not mandatory (except providing the URL of the different ontology), it can increase the performance of the system [Lopez et al., 2007]. The role of a person who customises the system is to associate certain words with relevant concepts from the ontology. For example, *where* needs to be associated with ontology classes which represent a location such as *City* and *Country*; similarly, *who* needs to be associated with classes like *Person* and *Organisation*. Additionally, it is possible to add the so called *pretty names* to the concepts or relations in case the term used when referring to a concept is not in the ontology. For example, if the property *locatedIn* is usually lexicalised as *in*, this will be added as a pretty name for that property. AquaLog also uses WordNet [Fellbaum, 1998] for extending the system vocabulary.

In an evaluation with 10 users who are not familiar neither with the KMI knowledge base[2] nor with AquaLog, they were given an introduction about conceptual coverage of the ontology pointing out that its aim is to model the key elements of a research lab such as people, publications, projects, research areas, etc. They were also told that temporal information is not handled by AquaLog and that the system is not a conversational system, as each question is resolved on its own without references to the previous questions.

From the 69 collected questions, 40 of them (57.97%) were handled correctly [Lopez et al., 2007]. However, this includes

[2]KMI knowledge base is populated based on AKT ontology `http://kmi.open.ac.uk/projects/akt/ref-onto/` and they are both a part of the KMI semantic portal: `http://semanticweb.kmi.open.ac.uk`

- 7 queries with failures which happened when the ontology lexicalisations and query terms did not match, or when the ontology was not designed in-line with the system's expectations. For example, when instances are modelled for certain concepts, whereas AquaLog would parse the query if the same terms were modelled as classes.

- 10 questions for which the answer was not in the knowledge base.

To evaluate portability, AquaLog was trialled with the wine ontology[3]. To customise the system to work with the new domain, first words like *where*, *when*, and *who* were associated with relevant ontology resources; then synonyms for several ontology resources were manually added. As pointed out in Lopez et al. [2007], this step was not mandatory, but due to the limitations of WordNet coverage, it increases the recall. Overall, the system was able to handle 17.64% of questions correctly. The system failed to answer 51.47% of questions due to the lack of knowledge inside the ontology[4]. The lack of knowledge was not the only cause for low performance. Many problems arose due to the *problematic* ontology structure, which is designed so that it contains a lot of restrictions over properties. In order to be handled properly by AquaLog, the ontology needs a simpler hierarchy structure; also, the terms in a query can only refer to ontology concepts between which the path length is not greater than two. For example, if the query were *which cities are located in Europe*, *cities* might refer to the ontology class *City*, and *Europe* might refer to an instance of the class *Continent*. If these concepts are related so that a *City* is located in a *County* and a *County* is located in a *Country*, where *Country* is located in a *Continent*, this query could not be handled by AquaLog. However, if in this chain *County* did not exist, and there was a direct relation between *City* and *Country* (located in), the query would be processed and answered as the path length between the terms *City* and *Europe* (as a continent) is two. In addition, all ontology resources should be accompanied by labels, as the performance of the system directly depends on them [Lopez et al., 2007].

AquaLog evolved into PowerAqua [Lopez et al., 2009b] – a QA system which

[3] `http://www.w3.org/TR/2003/CR-owl-guide-20030818/`

[4] Note that these numbers do not refer to the precision or recall as defined in Chapter 4.

works with ontologies available on the Web (e.g. crawled through Watson[5]). In contrast to AquaLog, PowerAqua addresses the challenges related to this heterogeneity: locating the ontology which contains the answer, resolving relations between the recognized resources, filtering duplicate questions which come from different sources and the like. The downside is the performance: in the initial evaluation with several ontologies (which were a collection of ontologies saved into an online repository, and not directly queried from the Web) and 69 questions, the time to answer queries was in the range of 0.5 to 78 seconds. PowerAqua successfully handled 48 questions resulting in 69.5% success rate. However, these 48 questions include those that have not returned any answer due to the knowledge not being available in the ontology (conceptual failure).

5.4 E-librarian

The E-librarian [Linckels and Meinel, 2007] system accepts a natural language question as input and returns the result found in the knowledge base as output. The knowledge base contains a set of short multimedia documents (clips), each of which documents one subject or a part of a subject. Hence, the system does not directly return the answer, but rather a clip in which the user can find the answer. All clips are semantically described using an ontology which is also used to interpret the user's question. The ontology serves as a hierarchical dictionary with domain-specific knowledge and relations of words such as synonyms, homonyms, hypernyms and hyponyms. This dictionary is carefully designed and used instead of external sources such as WordNet. There is no evaluation on how expensive it is to build this dictionary, however it needs to be built manually (see [Linckels and Meinel, 2007]).

The E-librarian service was applied in two applications: one is CHESt – about computer history, and the other is about fractions in mathematics – MatES. The performance of MatES is evaluated with 229 questions created by a mathematics teacher who was not involved in the implementation of the prototype. The system returned the set of documents that contained the

[5] `http://watson.kmi.open.ac.uk/WatsonWUI/`

correct answer for 97% of the questions, however the paper does not present sufficient information on the complexity of those questions.

5.5 PANTO

PANTO [Wang et al., 2007] is a portable NLI to ontologies. According to Wang et al. [2007], there is no specification for what types of questions are supported, but it is claimed that the system correctly parsed 170 questions taken from AquaLog's website, so we can assume that PANTO supports a set of questions that is similar to that supported by AquaLog. Similar to AquaLog, WordNet is used for the vocabulary extension, and the user lexicon is configurable - there is no need to manually customise the system unless the user is interested in adding associations to the ontology resources in order to improve the system's performance.

PANTO was evaluated with test data provided by Mooney[6] which had been used previously to evaluate NLIs to databases. This dataset covers three domains: geography, restaurants and jobs. As shown in Table 5.1 precision and recall for this dataset is quite high[7]. In addition, the range of supported NL queries is limited to those handled by SPARQL, e.g. questions starting with *how many* are not supported. Additionally, they do not report if the answer of the question was found in the knowledge base, as is the case with most other systems, but rather if the generated SPARQL query was correct. It is not clear from Wang et al. [2007] whether the system was customised prior to experimenting with the three different domains.

5.6 Querix

Querix [Kaufmann et al., 2006] is another ontology-based question answering system that translates generic natural language queries into SPARQL. Querix relies on clarification dialogs in the case of ambiguities. When Querix

[6] `http://www.cs.utexas.edu/users/ml/nldata.html`

[7] Note that the recall here is calculated with *number of answered* questions, even if they are not all correct.

was evaluated on the Mooney geography domain (215 questions) the precision was 86.08% and recall 87.11%. Similar to the performance of PANTO, if the answer was returned by the system, it was almost always correct. The system vocabulary is derived from the semantic resources and enriched by synonyms from WordNet. Hence, there is no need for customisation. The downside of this approach is that both the lexicalisations attached to the semantic resources and the availability of synonyms in WordNet strongly affect the system's performance.

5.7 NLP-Reduce

NLP-Reduce [Kaufmann et al., 2007] accepts full sentence queries, sentence fragments, or keywords as input. However, the relaxation of supported queries seems to have a negative impact on performance: when trialled with Mooney geography and restaurants datasets, the performance was lower than that of similar systems. Similar to Querix, the system requires ontology lexicalisations which are inline with the user's language, in order to return the answer.

5.8 CPL

Computer Processable Language (CPL) [Clark et al., 2005] is capable of translating English sentences to formal Knowledge Representation (KR). KR is Knowledge Machine (KM) language - a mature, advanced, frame-based language with well-defined semantics.

CPL was evaluated by two users in three domains: biology, physics and chemistry. They all received 6 hours of training individually, followed by one week using the question-answering system. Our understanding is that the domain knowledge was created using the CPL language, however, in Clark et al. [2007], it is not clear how much time was needed to create the domain knowledge used in the evaluation. In physics, 131 questions were asked, and the correctness of answers was 19%[8]. This low figure is due to the fact that

[8]It is important to point out that although Table 5.1 shows these measures as precision, this result is calculated on the overall set of questions, whereas most other systems removed

some questions were very complex, comprising several sentences. The total number of questions in biology was 146, and the average correctness was 38%. In chemistry, 86 questions were answered with 37.5% correctness.

Examination of the system's failures revealed that one third were caused by the fact that the user did not create the query that was understandable for the system (some common sense facts were not expressed explicitly enough). Another third were because the knowledge base did not have an answer and the last third were caused by mistakes by the CPL interpreter, so the system failed to find the solution.

5.9 Attempto Controlled English (ACE)

Attempto Controlled English (ACE) [Fuchs et al., 2008] is one of the oldest natural language interfaces which have been developed to serve as a formal language for knowledge representation. Unlike NL which is ambiguous, vague and potentially inconsistent, Controlled Languages have well-defined syntax, unambiguous semantics, and they can also support formal methods, reasoning in particular [Fuchs et al., 2008]. ACE is a controlled English, *"a precisely defined, tractable subset of full English that can automatically and unambiguously be translated into first-order logic."* [Fuchs et al., 2008][p.104]. ACE is translated into several variations of first-order logic, such as Discourse Representation Structures (DRS) [Blackburn and Bos, 1999], OWL, SWRL [Horrocks et al., 2004], and RuleML [Boley, 2003]. ACE is supported by many tools and a reasoner (RACE) which has been developed and used with it. For example, there is ACE Wiki which enables generating text in ACE which is being translated into OWL and SWRL and uses a predictive editor which supports a user while generating sentences (it supports a subset of ACE); AceRules translates ACE into rules; ACE View is a plugin for the ontology editor Protégé.

ACE is applied in domains such as software and hardware specifications, database integrity constraints, agent control, legal and medical regulations, and ontologies.

the questions for which the answer was not in the ontology before calculating precision.

Unlike majority of similar systems, ACE supports both knowledge representation and querying. However, the querying is feasible only if the knowledge is generated using ACE sentences. That is, the questions supported work well only if the knowledge has been generated by ACE. ACE supports yes/no and WH-queries. For example, if the ACE sentence which was translated to the knowledge representation was

```
A customer inserts a card.
```

we can ask a question such as:

```
Does a customer insert a card?
```

or another example:

```
A new customer inserts a valid card manually.
```

questions that could be answered are as follows:

```
Who inserts a card?
Which customer inserts a card?
What does a customer insert?
How does a customer insert a card?
```

What is most impressive about ACE is that the user can generate ACE sentences which are declarative, and then ask questions at the end. For example:

```
John enters a card. John drinks some water.
What does John drink? What does he enter?
```

This feature is unique to ACE and a few other similar systems like CPL.

5.10 Summary and Discussion

Figure 5.2 generalises the performance of NLIs in terms of precision, with regard to two factors: questions' complexity and the size of the evaluation

dataset. Here we assume that the *complexity* of questions is low if the system supports a limited set of grammatically correct questions (e.g., CNL), while it is high if the system supports both ill-formed, incomplete and fully grammatical questions.

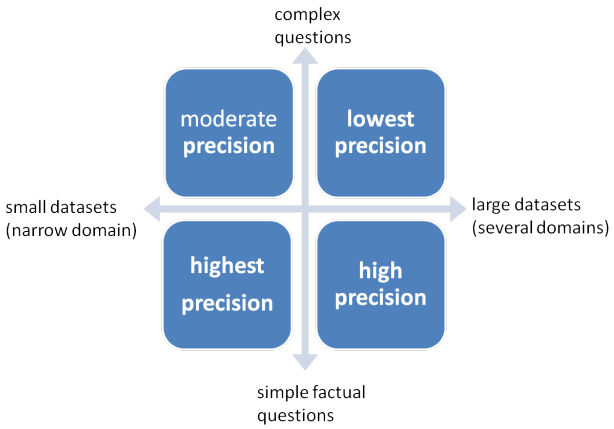

Figure 5.2: Performance variation based on question complexity and the dataset size

Systems that support *simple questions* and evaluated on *narrow domains*, tend to have a very high precision. The reason is *domain limitation*: the narrow domain narrows the scope of the questions which can be asked. In addition, the specification of the supported language poses a *language limitation* for the user, while having a positive influence on the performance. As the complexity of questions grows, the *language freedom* given to the user causes the precision to degrade for several reasons:

- While *domain limitation* narrows the scope of questions, supporting more relaxed queries gives more freedom to the user, and introduces more ambiguities which need to be resolved by the system.
- Systems which support relaxed questions with no full grammar required, tend to return an incomplete or incorrect answer, as they usually rely on shallow natural language processing, and do not support deep understanding of the question.

If systems which support complex questions are trialled with several do-

mains, the precision is affected even more. Here reasons related to the *language freedom* also apply, but in addition, there is *domain freedom*. This introduces a high challenge for building high-performance NLI systems as in addition to handling *any grammatical constructions* they also need to handle *any vocabulary*.

Finally, the systems which support simple questions, but trialled with several ontologies:

- have better performance than the systems which support complex questions. The reason is the *language limitation*.
- have the lower performance than the systems which work on narrow domains. The reason is the *domain limitation*.

With regard to the recall, it usually increases for systems which support simple questions. In addition, recall is strongly influenced by the customisation — more customisation is performed, the richer the lexicon and hence the higher the recall is.

Portability with *zero customisation* was claimed to be possible with many NLIs to ontologies which have been developed in the last few years. This was achieved by automatically building the lexicon from the ontology. Majority of described systems use external sources to extend the vocabulary and include synonyms, hypernyms and hyponyms. The most common resources are WordNet [Fellbaum, 1998], FrameNet [Ruppenhofer et al., 2005], and OpenCyc[9]. Few systems use resources accessible through the Semantic Web using relation *owl:sameAs* (e.g., PowerAqua, see Section 5.3).

However, the more technical the domain gets, the less chance there is that one can rely on lexical matching alone. In fact, it is not expected that the complete lexical knowledge necessary for very technical domains is present in general resources such as WordNet [Cimiano et al., 2008]. This is especially the case for properties. That is why domain lexicons, which contain only domain-specific vocabulary, tend to be used by systems such as E-Librarian or ORAKEL. Manually engineering a lexicon as in the ORAKEL system certainly requires substantial effort, but it allows one to directly control quality

[9] `http://www.opencyc.org/`

and coverage of the lexicon for the specific domain [Cimiano et al., 2008]. Moreover, it has been shown that the more time users spend customising the system, the better its performance (see Section 5.2).

However, this means that systems which have a good performance involve three kinds of users, see Figure 5.3:

- *Ontology engineers* who design the ontology based on the knowledge from a domain expert.
- *Domain experts* who are familiar with the terminology and abstract concepts in the domain.
- *Application developers* need to align the language of a *domain expert* with the generated ontology.

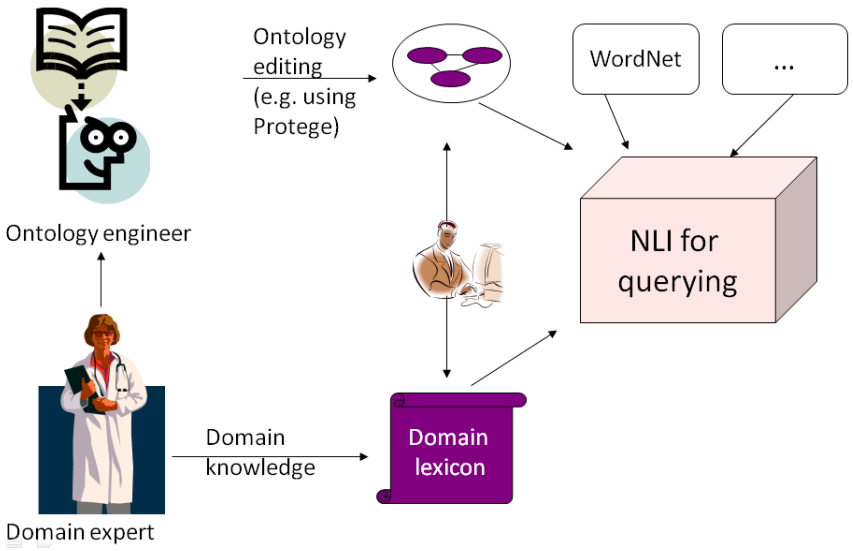

Figure 5.3: Semi-automated process of creating the domain lexicon from the ontology

An alternative way to generate/enrich the lexicon for NLIs to ontologies is shown in Figure 5.4. Now instead of having three kinds of users, we can involve only *domain experts* whereas they would have to use NLIs for knowledge representation, such as ACE [Fuchs et al., 2008], CPL [Clark

et al., 2005], or recently developed SOS (Sydney OWL Syntax) [Cregan et al., 2007], CLOnE [Funk et al., 2007], and Rabbit [Hart et al., 2008].

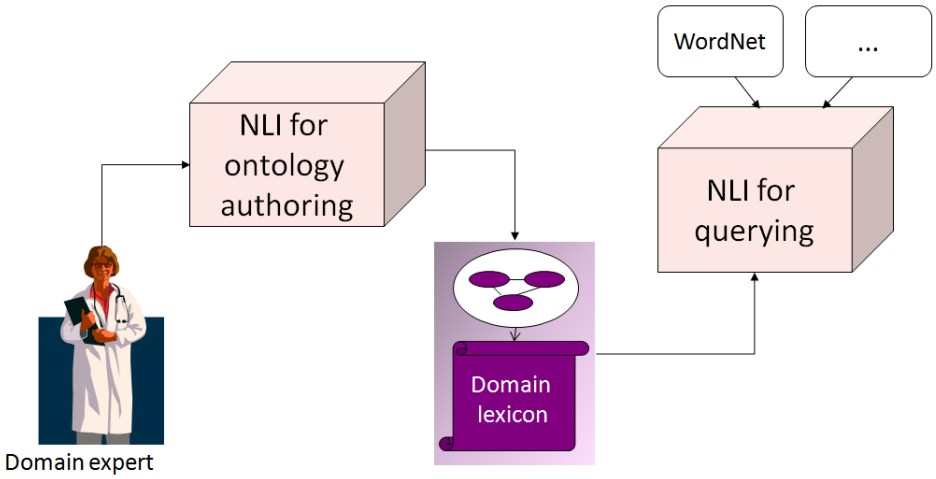

Figure 5.4: Automated process of creating the domain lexicon from the ontology

ACE is probably the most powerful, not only because of its maturity, but also due to many support tools, such as OWL Verbaliser, which can be used to generate the lexicon from the ontology which is built externally; the lexicon can be updated/enriched by changing/adding new ACE sentences. In fact, systems like ACE which can be used both for *generating* and *querying* the knowledge have very good performance in *querying* if the knowledge is *generated* using the same language.

While neither of the NLIs for knowledge representation are tailored to a specific domain, porting them requires knowledge of the supported language, in order to generate/update the domain knowledge. The question is which of the following is the easiest:

1. to learn a required supported language (e.g., a CNL) for knowledge representation;
2. to learn how to use the customisation software (such as FrameMapper in the case of ORAKEL);
3. to learn the ontology structure and then place mappings between certain words such as *Where* and ontology concepts such as *Location*

through configuration files (such as in the case of AquaLog and PANTO);

4. to use any other tools for ontology editing in order to enrich an already existing lexicon in the ontology, so that automatically generating the lexicon from the ontology can be sufficient for reasonable performance.

The last two options are not practical for large ontologies or for a set of ontologies, while the second option might involve *domain-experts* who will generate a set of sample questions, so that *application developers* can associate question terms with the particular ontology concepts. The first option is most attractive due to the involvement of only *domain experts*. However, as the underlying knowledge representation language (such as OWL) relies on logical statements, the natural way to design NLIs for knowledge representation is to use a CNL which is in fact yet another formal language and needs to be learnt. The task of designing a CNL for knowledge representation is challenging because not all mathematically clear and logical expressions can easily be translated into English. Yet, CNLs need to be intuitive for domain-experts which are often not logicians. An example from Schwitter et al. [2008][p.7], where the authors compared ACE, Rabbit and SOS, expressing that two things differ from each other would look as follows:

```
OWL: DifferentIndividuals([Individual(Scotland),
                           Individual(England)])
ACE: Scotland is not England.
RAB: England and Scotland are different things.
SOS: Scotland and England are different individuals.
```

While no domain expert would have a problem reading and understanding any of the mentioned sentences (apart from OWL which is too formal), generating them might be problematic. They sound natural in places, but they often seem too explicit, as in the examples above. In the evaluation of Rabbit with domain experts, each generated ontology was different and had to be *corrected* by the knowledge engineers in order to be useful [Denaux et al., 2009]. This indicates that the proposal in Figure 5.4 is too idealistic: *the domain experts might not be able to generate the knowledge in the first-order logic such as OWL, without the help of a knowledge engineer.*

To conclude, customisation of NLIs, irrespective of the tools which are used to accomplish this, significantly affects their performance. While ontologies can ease the process of customisation, building systems which can work with *any* ontology and thus are *portable with no customisation is difficult* – ontologies have different designs and varying quality of lexicalisations. Therefore, building a lexicon from ontologies and using external sources such as WordNet might not be sufficient, and customisation might be necessary in real world applications. The question is, *in what form this customisation poses the least overhead to the users who need to customise it.*

On the other side of the coin are end-users. Even if the system is customised to work very well for the specific domain, the performance is still affected by the way the end-user uses the system. The next chapter is concerned with this topic.

Chapter 6

Usability Enhancement Methods

In the previous chapter, we discussed the portability of NLI systems and its relation to performance (expressed through precision and recall). In this chapter[1], we will assume that the system is ported successfully to the new domain, and the end-users can post questions in the form of Natural Language.

Although NL is intuitive, the simplicity of the interface, which is often a single text box for queries, may cause additional problems for end-users [Stojanovic, 2005b]. Moreover, designing NLI systems is not trivial due to the ambiguities and complexities which arise from the Natural Language itself. One of the ways to approach this problem is to support simple and explicit semantic limitations [Epstein, 1985], i.e. by restricting the supported vocabulary and grammar.

The usability of an NLI system depends on the level of the end-user satisfaction - if the user does not have any difficulties using the system, then it can be considered habitable. A habitable system, as discussed previously in

[1]The content of this chapter is an updated and extended version of Section 4 in *D. Damljanovic, K. Bontcheva: Towards Enhanced Usability of Natural Language Interfaces to Knowledge Bases. In V. Devedzic and D. Gasevic (Eds.), Special issue on Semantic Web and Web 2.0, Annals of Information systems, Springer-Verlag, 2009.* I am grateful to the contribution of K. Bontcheva who read and commented on the initial version of the paper and suggested improvements.

Section 4.1, makes the user aware of the reasons for failures if they happen. By identifying these failures, we can further improve the system and make it more usable. In an attempt to address this problem, many methods have been developed for improving the communication between the user and the system. Figure 6.1 illustrates some of the methods which adapt the system's vocabulary to that of the user (red circle), and those which aim to adapt the user's vocabulary to that of the system (yellow circle).

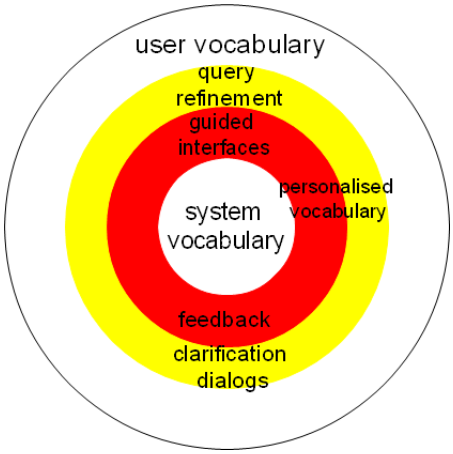

Figure 6.1: Synchronising the user and the system vocabularies

In what follows we will first discuss *language restriction* in Section 6.1, followed by the effect of *feedback* (Section 6.2) and *guiding the user* (Section 6.3) through questions. In addition, as discussed in Section 5.10, the system vocabulary is often extended from external sources (e.g., WordNet). For more personalised systems, this extension can be user-centric, as the user vocabulary can be used for extending the system vocabulary (Section 6.4). Once the user is familiar with the system vocabulary, the opposite adaptation needs to take place, as the user vocabulary needs to be in line with that of the system. Methods for assisting the user in that adaptation are also used to solve the *ambiguity* problem and are discussed in Section 6.5.

6.1 Language Restriction

As we have previously discussed in Section 1.2, a Controlled Natural Language is a subset of a natural language that includes certain vocabulary,

grammar rules and restrictions that have to be followed. The biggest challenge when designing a CNL is restricting the natural language in a way that it remains intuitive and does not require extensive training for the end user. However, applications in industry prove that, actually, CNLs can be learnt and used in practice. For example, AECMA Simplified English [Unwalla, 2004] has been used by the aviation industry since 1986.

Another example is from CPL's evaluation [Clark et al., 2007] – although users had to be very familiar with CPL in order to use it successfully, they did not have problems working with its grammar restrictions. Several failures occurred due to using the language which was not explicit enough for the system (i.e. common-sense facts were not made explicit). The conclusion in Clark et al. [2007] was that the system would benefit from showing the user the derived query interpretation so that any mistakes could be corrected. As is pointed out in Clark et al. [2005, p.510] "*a challenge for languages like CPL is to devise methods so that these corrective strategies are taught to the user at just the right time e.g., through the use of good system feedback and problem-specific on-line help*".

According to Ogden and Bernick [1997], constraining a user to a limited vocabulary and syntax is inappropriate, as users should be free, but the constraints should come from the *task* and the *domain* instead. However, allowing the task and the domain to constrain the language still does not prevent the user from creating ambiguous queries. As natural language itself is ambiguous even in human to human communication, controlled languages have a role to play in reducing the ambiguity. The main drawback of CNLs is their rather steep learning curve. However, according to Zolton-Ford [1984], a limited vocabulary, coupled with a feedback mechanism, means easy training from an end user's point of view.

An alternative approach to restricting the vocabulary is relaxing the queries to support keyword-based in addition to full-blown grammatically correct questions. This allows some flexibility, for example if the user is not familiar with the full expressiveness of the controlled language, he can try using keywords, while for more advanced users there is the option of using full-blown questions. An example of such a system is NLP-Reduce [Kaufmann et al., 2007], which would give the same result for both *capital France* and

what is the capital of France? queries. However, as we will discuss later, allowing users to type in keywords can lead to misconceptions about the system due to expecting the functionality of Information Retrieval engines such as Google, and moreover, expressing the information need precisely through a set of keywords might be difficult.

6.2 Feedback

Showing the user the system's interpretation of the query in a suitably understandable format is called feedback. Several early studies ([Zolton-Ford, 1984],[Slator et al., 1986]) show that after getting feedback, users are becoming more familiar with the system interpretations and the next step is usually that they are trying to imitate the system's feedback language. In other words, returning feedback to the user helps them understand how the system is transforming the queries, therefore motivating them to use the similar formulations and create queries which are *understandable* to the system.

Formulating the query and the user behaviour has long been researched in Information Retrieval (IR): during the search process, the user performs several actions [Stuckenschmidt et al., 2004]:

- poses a query based on an existing information need,
- after retrieved results are shown, decides to either:
  - stop, or
  - reformulate the query in a way which promises to improve the result

This is repeated until the *perfect* answer (relevant documents in the case of IR systems) is found. As this traditional model is adequate only for simple cases, a so-called berry-picking model [Bates, 1989b] has been proposed where users take some of the results and move on to a different topic area. This model assumes that the user starts off with a query on a particular topic and based on the results, he can either explore the result set or re-scope the

search by re-defining the information need and posing a new query. Although different users behave differently during the search process, it has been shown (see [Stuckenschmidt et al., 2004]) that the majority prefer interactive methods, where the system performs the search, gives the feedback to the user and lets him decide about the next steps.

In the evaluation of Querix and three other interfaces for the semantic web [Kaufmann et al., 2006], the system was preferred over all the others because it returned the answer in a form of a sentence, in contrast to the list of answers returned by the other three systems. For example, the question *How many rivers run through Colorado?* was answered by Querix as: *There are 10*, while the other three systems returned a list of rivers and the number of results found. Because of the way Querix replied to the questions, users had the impression that the system really understood them, and trusted the system more.

CNL systems usually implement some form of a feedback. For example, using ACE View in Protégé, typing in *Serbia is a country* the system will show the feedback message: "The sentence was successfully parsed". If the input sentence is changed to *Serbia is a country in Europe* the feedback message will be: "The snippet contains ACE snippet errors". It will then highlight *Europe* which 'confused' the ACE parser, and it will also show the <> sign at the position where the parsing failed. For example, it will show *Serbia is a country <> located in Europe*. However, in order to refine this query and get the answer, the user needs to find out which constructions are supported/understandable by the ACE parser. The user might be able to construct the query in a different way, if he did know how, or if he had more descriptive feedback (for example, whether the above sentence has a problematic grammar not supported by ACE, or if the lexicon is the problem). However, feedback messages are usually automatically generated and are based on very simple rules such as which part of the statement/query is recognised, and which is not.

Another example is CPL: in order to correctly formulate questions using CPL, users need to know "a bag of tricks" [Clark et al., 2007]. That is one of the reasons why an interactive process of question-asking was introduced in CPL. After the user poses the question, their *Advice System* detects CPL

errors and returns reformulation advice. There are 106 different advice messages triggered when the user's question contains grammar rules that are outside the scope of CPL, although correctly interpreted in English; or when the user omits words, such as a unit of measure after a number. In CPL, the feedback does not contain the input text from the user, but rather detects the error and gives advice from a static list of feedback sentences. As Clark et al. [2007] point out, automatic rewording would be very challenging, especially with longer, complex sentences. In addition to the *Advice System*, an *Interpretation Display System* is implemented, which shows the user the system's interpretation of the question. It works so that after posing the question, the system generates a set of English paraphrases and shows them to the user. In addition, it generates a graph where nodes are objects or events from the question, and arcs are relationships between them. If the user detects an error in the graph or English paraphrases, it is possible to rename nodes and arcs, or to reformulate the whole question and inspect the system's interpretation again. According to the evaluation presented in Clark et al. [2007], this graphical representation was well perceived by users.

Although it might be annoying for users, it is not unusual for systems to fail to answer a question, due to an unsupported query syntax, even though the same query could be answered if re-formulated. Adding support for extra linguistic coverage is not always easy due to the need to balance expressiveness with ambiguity. For instance, the evaluation of AquaLog with the KMI ontology [Lopez et al., 2007] shows that 27.53% (19 of 69) of the questions could be handled correctly by AquaLog if re-formulated which means that 65.51% of failures could be avoided. Reformulating in this case entails stating the queries in AquaLog's supported language so that unsupported linguistic failures are avoided, as well as nominal compounds, or unnecessary functional words like *different*, *main*, *most of*.

Closer look at user's queries and behaviour during evaluation of CPL presented in Clark et al. [2007] revealed that users rarely "got it right" the first time. The average number of attempts of reformulating the query by the user, before either getting a satisfactory answer from the computer, or giving up, was 6.3 and 6.6 in physics and chemistry, respectively, while for biology it was 1.5 only; this is related to the questions complexity: the most common

questions in biology where very simple, such as "what is an X?", in contrast to the "story" questions posted in physics, and the algebraic questions posed in chemistry. Further analysis of the frequency of actions taken for reformulating the query, and the types of these actions, showed that the biggest problem for users was to find the right *wording* that enabled the system to answer the question. For example, in chemistry one of the questions was whether a compound is *insoluble*. Users tried several words to express solubility: *soluble*, *dissolve*, *solution*, *insoluble*, until finally hitting on *solubility*, for which the system was able to give the answer.

Summary: By providing the user with the feedback in the form of the system's interpretation of the query, users can learn how to generate queries more efficiently. For example, showing the user which words were understandable and which were not, helps users to familiarise themselves with the system's vocabulary more quickly, and avoid repeating mistakes.

In cases when the system is not able to interpret the query, the system could provide the user with a suggestion of how this query could be reformulated in order to be answered (e.g., by showing examples of supported types of queries adapted for the particular domain).

6.3 Guided Interfaces

Guided interfaces support the user by *suggesting* the queries which are supported by the system. In Bechhofer et al. [1999], relations between concepts are used to assist users by expressing what it is possible to ask about the concept which is typed in — this way only meaningful questions should be posted.

According to Bullock [1999] there is a need for lucidity in information systems – a system should supply the user with an idea as to what is available, and which next steps can be taken. In Bechhofer et al. [1999], the tool for assisting the users in formulating queries is described. The tool is driven by the content of the conceptual model. The tool uses constraints known as *sanctions* which are added to the ontology and which describe the meaningful compositions which can be built. Sanctions are used for lucidity or guidance for creating suggestions. Suggested manipulations are:

- *restriction* – specialising the query by adding more criteria,
- *widening* – removing criteria from a composite query,
- *replacement* – replacing the query by a more specific one, and
- *sibling replacement* – replacing subqueries with sibling concepts.

Hallett [2006] presents a NLIDB that assists users by guiding them through the supported queries. This guided interface automatically infers the set of possible queries that can apply to a given database and generates query frames. *A query frame* is a system-generated query which contains unfilled WYSIWYM anchors. *An anchor* is a part of the WYSIWYM terminology and means a span of text in a partially formulated query, that can be edited by the user to expand a concept. The evaluation results are very encouraging: according to the usability evaluation *"users can learn how to use the interface after a very brief training and succeed in composing queries of quite a high level of complexity"* [Hallett, 2006][p.14]. To evaluate coverage of their system they used 250 questions from GeoBase (also known as the Mooney GeoQuery dataset used in Tang and Mooney [2001] and Popescu et al. [2003]). Their system could generate query frames for 58% of the questions. 42% could not be handled (questions requiring inferences over numerical types, such as *which is the highest point in Alaska* or *what is the combined area of all 50 states?*). However, if they could generate a query frame, the answer was always correct. Although their system is limited because the user cannot post free text queries, a precision of the system of 100% for the questions that were available is encouraging.

A similar approach was applied in an NLI for querying ontologies – Ginseng [Bernstein et al., 2005][2]. This system allows access to knowledge bases in OWL format through NL. The evaluation of Gingseng resulted in 92.8% precision and 98.4% recall, which indicates that, although the user is limited in the way questions can be asked, this is counter-balanced by high performance thanks to the offered support. The evaluation of its descendant GINO [Bernstein and Kaufmann, 2006] with six users proves that the use of guided entry overcomes the habitability problem. The GINO system offers

[2]More details: `http://www.ifi.uzh.ch/ddis/research/semweb/talking-to-the-semantic-web/ginseng/`

guidance to the users as they formulate NL queries step by step, ensuring that only valid queries are posed.

Another option for guiding the user through the domain and available concepts is by using *auto-completion*. Traditional auto-completion is based on matching input strings with a list of the words in a vocabulary, sorted by different criteria e.g., popularity, user preferences, etc. For ontology-based systems, this concept can be extended to the semantic level so that in addition to traditional string similarities, relations between ontology resources are used in order to predict the next valid entry. The proposed *semantic auto-completion* is described by Hyvönen and Mäkelä [2006] and applied in information retrieval, specifically for multi-faceted search. For example, the semantic portal MuseumFinland[3] uses semantic auto-completion on request. The search keywords are matched not only against the actual textual item descriptions, but also against the labels and descriptions of the ontological categories for which they are annotated and organised into view facets. As a result, a new dynamically created facet is shown on user request and it contains all categories whose name or other configurable property values, such as alternative labels, match the keyword. For example, if the user types in *EU countries*, the system would show list of countries in the dynamically generated facet, from which the user can choose.

Summary: With guided interfaces, the user is limited as the number of questions is limited, but the performance is rather high – once the user formulates the query, it is very likely that he will get the correct answer. A more flexible option is the use of semantic auto-completion. Contrary to fully guided interfaces, this method allows for more freedom to the user.

6.4 Extending the Vocabulary

As it has been discussed in Section 5.10, many NLIs use external vocabularies such as WordNet in addition to the domain lexicon (for example, to retrieve synonyms). The user's vocabulary could be a good source for extending the system vocabulary as well.

[3] `www.museeosuomi.fi`

AquaLog [Lopez et al., 2007] is backed by a learning mechanism, so that its performance improves over time, in response to the vocabulary used by the end users. As already discussed in Section 5.3, when porting AquaLog to work with another domain, it is possible to configure its lexicon by defining "pretty names". During runtime, when the system is interpreting user's input ambiguously, it asks the user to help by choosing from several possible interpretations. The user's selection is then saved as a "pretty name" for future disambiguation of the same type. For example, in the evaluation of AquaLog, it was noticed that when referring to the relation *works-for* users choose words such as: *is working*, *collaborates*, *is involved in*. Since the system does not know that *collaborates* can be interpreted as referring to the property *works-for*, it will prompt the user with the available options, and 'learn' the user's choice. In addition to learning a new term, AquaLog records the context in which the term appeared. The context is defined by the name of the ontology, the user information, and the arguments of the question. Arguments of the question are usually the two arguments of the triple, namely two classes or two instances in the ontology connected by a relation.

To evaluate how the Learning Mechanism (LM) affects the overall system performance and the number of user interactions, two experiments are conducted and results are reported in Lopez et al. [2007]. First, AquaLog is trialled with LM deactivated. In the second experiment two iterations are performed. First, the LM is activated at the beginning of the experiment in which the database containing learned concepts is empty. The second iteration is performed over the results obtained from the first iteration.

The results show that using LM improves performance from 37.77% of answered queries to 64.44%. Queries that could not be answered automatically (i.e. required at least one iteration with the user) are quite frequent (35.55%) even if the LM is used. This is because the LM is applied only to relations, not to terms. Overall, the number of queries that required 2 or 3 iterations are dramatically reduced with the use of the LM system which improved the performance even after the first iteration from 37.77% to 40% as it uses the notion of context to find similar but not identically learned queries. This means that LM can help disambiguate the query even if it is the first time

this query is presented to the system.

Summary: Although external sources such as WordNet can enrich the system's domain vocabulary, the user-centric vocabulary can play a significant role in increasing the performance of the system as it has been shown in the evaluation of AquaLog. In addition to maintaining the user's vocabulary, the AquaLog's approach can be extended in several directions:

- *to explore the user-contributed vocabulary which is not personalised, but shared among all users.* For example, if the user A asks *Who works for the University of Sheffield?*, the system can map *The University of Sheffield* to the *Organisation*, and *Who* to a *Person*, but the construction *works for* could be unknown and not similar to any of the existing ontology relations between classes *Person* and *Organisation*. If there are several relations between these concepts, the system can prompt the user (as would be the case with AquaLog) to choose from the list of available options. If the user chooses the relation *employedIn*, the system will remember that *works for* can be related with the relation *employedIn* and would add this to the user-centric vocabulary. Now if user B asks the same question, and there is no data about *works for* construction in his user-centric vocabulary, the vocabulary of the user A could be used to automatically give the answer to the user B, or to rank the *employedIn* relation on the top of all others suggested by the system; a similar feature is discussed in Lopez et al. [2007] as a future work for their learning mechanism where they propose grouping the vocabulary by user profiles.

- *hiding complexities*: recommendations offered to users are usually the names of potential ontology resources, e.g., names of properties, and these sometimes do not sound natural. For example, properties usually consist of at least two words, such as *hasBrother* or *has-brother*. Simple processing of such names can help in deriving more natural words such as *has brother*. However, the way ontology resources are named is definitely not standardised and this feature would have to be customised for each system dependent on the domain;

- *learning could be applied to terms* (classes and instances), not only to

relations;

- *to put users in control*: to allow users to see and modify the created lexicon at any time.

6.5 How to Deal with Ambiguities?

Although controlled natural languages reduce ambiguities to some extent, some issues, specific to the domain knowledge, still remain. For example, if the knowledge base contains two instances of a class *Person* with the same name e.g., *Mary*, the system is not able to predict which one the user is interested in. The way this problem is usually solved is by using one or the combination of the following methods:

1. *automatically resolving ambiguities*: using heuristics and ontology reasoning to implement ranking algorithms;
2. *clarification dialogues*: by involving the user;
3. *query refinement*: in cases when the cause of ambiguity is a vague expression of the user's information need.

6.5.1 Automatically Solving Ambiguities

The E-librarian system [Linckels and Meinel, 2007] uses a *focus function* algorithm in case of ambiguities. A focus function returns the best interpretation for a given word in the context of the complete user question. If more than one best interpretation is found, they are all shown, although the experience with the system revealed that the users generally enter simple questions where the disambiguation is normally successful.

OntoNL is an ontology-based NLI for multimedia semantic repositories [Karanastasi et al., 2007]. This system combines domain knowledge with user profiles, both represented in standards such as MPEG-7 and TV-Anytime to resolve ambiguities and rank results, thus avoiding clarification dialogues. Their system is domain-specific and oriented towards digital libraries.

A large amount of semantic resources available on the Web can be a reasonable source for automatic disambiguation. One example is the IBM's Watson[4] where DBPedia.org (among other resources) is used to calculate the confidence score based on which low scored query interpretations are disregarded. Another example is PowerAqua, where WordNet in combination with the relations found on the Semantic Web is used for disambiguation. Firstly, the *sense-based similarity matcher algorithm* disregards ontology *terms* that are syntactically related, but not semantically equivalent to the query terms. *Semantically equivalent* are those that appear as hypernyms or hyponyms of the given term, or hold an "is-a" relationship with the synset of the term, based on WordNet. Secondly, *relatedness* between ontology terms based on the analysis of the existing taxonomy and relationships between semantic resources is calculated. If the relatedness is low, the query interpretations are disregarded.

As the domain knowledge grows, the task of automatically solving ambiguities becomes more difficult, and often the only way to resolve it is by engaging the user through the clarification dialog.

6.5.2 Clarification Dialogs

In case of ambiguities Querix [Kaufmann et al., 2006] sends them to the user for clarification. In this process users need to disambiguate the sense from the menu with system-provided suggestions, in order to get better retrieval results. For example, if the user enters *population size* and the system cannot decide if the user is interested in the property with name *population density* or *population*, it will ask the user to choose between the two.

Similar to Querix, the AquaLog system [Lopez et al., 2007] relies on clarification dialogues when ambiguity arises. In contrast to Querix, AquaLog is backed by the learning mechanism discussed earlier (see Section 6.4).

In general, clarification dialogues can help users resolve ambiguities, however, if the suggestions provided by the system are not satisfactory, it is possible that the user's need was not expressed precisely enough in the query, which is the main pre-requisite for retrieving relevant information from the knowledge

[4] `http://www.watson.ibm.com`

base (see Figure 6.2). In addition, the query must contain the lexicalisations which are found in the domain knowledge lexicon. This is why, many existing NLI systems, extend their lexicons by using external sources for detecting synonyms: the query posed by the user is very likely to contain words which do not exist in the lexicon, although words with equivalent meaning do.

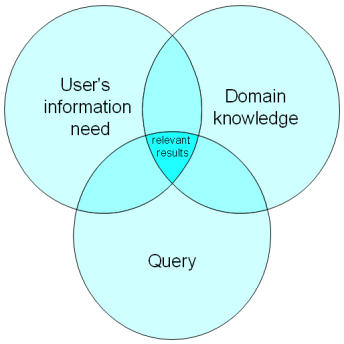

Figure 6.2: Retrieving relevant results

According to Stojanovic [2005b], there is usually a gap between the information need and the query expressing that need, which is caused by "the usage of short queries, whose meaning can be easily misinterpreted". The indicator of this gap, which is called *query ambiguity* [Stojanovic, 2005a], can be reduced by the process of query refinement.

6.5.3 Query Refinement

Changing or refining the query in order to obtain results that are more relevant is called *query refinement*. When refining the query it is important to know the precise information need as well as which part of the query to change/refine [Stojanovic, 2005b]. Refining usually means adding more constraints to the query, until the quality of the results corresponds to the user's expectation.

Librarian Agent [Stojanovic, 2005b] – a system created to replace the human librarian when helping users to find the appropriate books in a library, uses the query refinement technique proposed by Stojanovic [2005a]. Librarian Agent measures query ambiguities with regard to the ontology structure (structure-related ambiguity) and the content of the knowledge base

(content-related ambiguity). Ambiguities are interpreted from the point of view of the *user's need*, and are implicitly induced by analysing the user's behaviour. Modelling a user's need is not trivial especially when users are anonymous as the model of a user's behaviour has to be developed implicitly i.e. by analysing *implicit relevance feedback* whose main purpose is to infer the information need by analysing a user's interaction with the system [Stojanovic, 2005b].

The query refinement process is treated as the process of moving through the *query neighbourhood* in order to decrease its ambiguity regarding the user's need [Stojanovic, 2005b]. Librarian Agent defines the query neighbourhood through identification of the query constraints and the ambiguity for each word. Query neighbourhood includes determining:

1. *A more specific query.* The query is refined so that the set of answers is more specific.
2. *A more generic query.* The query is refined so that the set of answers is bigger.
3. *Equivalent query.* The query is rewritten so that the returned results are the same, but this is initiated for other reasons e.g., optimising the execution time.
4. *Similar queries.* The query is refined so that its results are partially overlapped with the initial query.

The approach is evaluated with 20 questions, which cannot be expressed precisely using the defined ontology vocabulary, but the answers are contained in the information repository, e.g., *find researchers with diverse experiences about the semantic web*. The goal of the evaluation was to see how the effectiveness of the ontology-based querying is affected, when the query process is enhanced with the presented refinement facility. Six computer science students with little or almost no knowledge about domain ontologies, and with no knowledge of the system, were asked to retrieve resources for 10 questions in one session, using the two retrieval methods (with and without the refinement). Users were asked to explicitly confirm when they got relevant results.

For each search four measures have been considered: success, quality, number of queries, and search time. Results revealed that *success* and the *quality of the session* were significantly higher (57/85.7%; 0.6/0.9), while the *number of queries* and the *search time per session* was significantly lower for the system with query refinement switched on (10.3/5.2; 2023/1203s). Stojanovic [2005b] concludes that if the system can discover and measure ambiguities in a query and support the user in resolving these ambiguities efficiently, the precision and recall of the retrieval process will increase.

Summary: In some cases it is not convenient for users to control the output either because they are not interested in doing so, or the system might have enough data to efficiently solve ambiguities automatically. However, this is strongly related to the domain and the system functionality. The more specific the domain and the simpler the system, the more feasible automatic ambiguity resolution is. For more generic domains, a better solution would be engaging the user to resolve ambiguities through clarification dialogs. In cases of imprecisely expressed information need, query refinement is likely to be a good solution. However, it is important to observe users, their actions and behaviour during the process of refinement.

6.6 Summary and Discussion

Design of habitable NLIs and the choice of the methods which should be used rely mainly on the targeted *domain*:

- *Closed-domain NLIs* usually work with one or several ontologies covering a narrow domain
- *Open-domain NLIs* work with a set of ontologies covering various domains, ideally those available and published on the Web

Closed-domain NLIs might benefit from **guided interfaces**, which usually have good performance, however the queries are fully controlled by the system. This means that the user does not have the freedom to enter queries of any length and form. A more flexible way of guiding the user is using **auto-completion** and this can also be applied to open-domain NLIs.

Another important factor when addressing habitability is the *supported language*: an NLI which supports a CNL usually has good performance, however the user has to learn the CNL. If the NLIs support more flexible language, the performance is usually degraded. One of the reasons is *ambiguity* which arises due to one of the following reasons:

- *The undefined information need*: flexible language usually means supporting keyword-based queries which are often not enough to express the need precisely. In this case **query refinement** could be used to derive similar queries, more specific and more generic ones. Ontologies play a significant role in predicting the query refinement process e.g., by defining a set of similar queries.
- *The broad domain*: for closed-domain NLIs, it is possible to resolve ambiguities automatically by calculating the rankings of the query interpretations based on the ontology reasoning. For open-domain NLIs, although a ranking mechanism is advisable, it might be challenging to resolve ambiguities automatically and the user might need to be engaged with **clarification dialogues** to choose between system provided options. The challenge in this case is generating relevant, but not too long of a list of options for the user.

Methods which can increase the habitability of NLIs irrespective of the domain and the supported language are:

Feedback. Providing the feedback to the user by showing the system's interpretation of a query, the user can learn how to generate queries efficiently. Moreover, an NLI system should *make the user aware* of the type of a failure when it happens, by showing which habitability domain was affected:

- *conceptual failure*: knowledge is not available in the system;
- *lexical failure*: the user should use different words when asking the question;
- *functional failure*: the user might need to split the query into several more simple ones;

- *syntactic failure*: the user should be encouraged to reformulate the question as the grammar might not be supported by the system, but an alternative way to ask the same question and get a result, exists.

The real challenge here is how to model feedback so that it does not pose too much overhead to the user.

Extending vocabulary. Although external sources such as WordNet can enrich the system vocabulary, as well as the lexicon which is created individually for each domain, the *user-contributed* vocabulary can play a significant role in increasing the performance of the system over time. In contrast to the *personalised vocabulary* which observes the user as an individual it would be interesting to simplify this approach, and *share* vocabulary among different users. In addition to maintaining the user-contributed vocabulary, this approach should allow users to see and modify the created lexicon at any time.

From the discussion in this and the previous chapter, the biggest challenges when building NLIs to ontologies can be summarised through the following requirements:

1. *Portability* with minimal customisation.
2. *Ambiguity:* unambiguous transformation from NL into a formal query.
3. *Expressiveness/robustness:* avoiding the use of a controlled language, but allowing users to enter queries of any length and form.
4. *Minimum training* for the user and keeping the supported language intuitive.
5. Assisting the user in the process of query construction.
6. Allowing users to control the output by providing a mechanism to follow the system's transformations from the input (query) to the output (result), so that they can disagree on the system's decisions and refine the query at certain points.

7. *Hiding complexities* of the queried knowledge structure: showing results without imposing users to the underlying complexities of the structured knowledge.

In the next chapters, we present how we have addressed these challenges through the design of two systems:

- QuestIO (Question-based Interface to Ontologies) builds the domain lexicon automatically from the semantic resources, and tries to automatically interpret the user's query based on internal ranking mechanisms which rely on ontology reasoning.
- FREyA (Feedback, Refinement and Extended vocabularY Aggregation) is an interactive system which explores:

    1. *Feedback*, which refers to showing the user query interpretations. First experiments are presented in Chapter 8, where the approach of QuestIO is extended with an interactive method of showing the *query interpretations* to the user and allowing them to choose the correct one. This approach is evaluated with users, and the modifications which are adapted are presented in Chapter 9.
    2. *Refinement* in FREyA (described in Chapter 9) refers to resolving ambiguities which arise due to broad domain coverage. Ambiguities are resolved by engaging the user with clarification dialogs.
    3. *Extended vocabulary* is described in Chapter 9, where in addition to the automatically generating lexicon (as in QuestIO), the user's vocabulary and synonym detection coupled with a learning mechanism is used to improve the performance of the system.

Part III

Building Natural Language Interfaces to Conceptual Models

Chapter 7

QuestIO

In this chapter[1] we present Question-based Interface to Ontologies, and its evaluation.

The Question-based Interface to Ontologies (QuestIO) system[2] translates

[1] The topic of this thesis originated as a part of my work within the TAO (Transitioning Applications to Ontologies) project. The methodology used in TAO is to be published in *H. Wang, D. Damljanovic, T. Payne, N. Gibbins, and K. Bontcheva: Transition of Legacy Systems to Semantically Enabled Applications: TAO Method and Tools. Semantic Web Journal, IOS Press, 2011.*, where my main contribution is writing about ontology learning tools in Section 2.1.2 which goes beyond the scope of this thesis, and the implementation and writing about semantic annotation and querying ontologies using Natural Language in Section 2.1.3 – that section is a summarized version of Sections 7.1 and 7.2 in this chapter. Section 3 in the paper is by large my contribution which includes evaluation described in Sections 7.4.1 and 7.5.6 in this chapter. The rest of the paper is written by the first author, while the other authors read and provided suggestions for improvements of the work.

[2] The initial idea about work described in this thesis was presented publicly for the first time in *D. Damljanovic: Natural Language Queries for Enhanced Knowledge Access, Summer School on Multimedia Semantics Analysis, Annotation, Retrieval and Applications (SSMS'07) , Glasgow, UK, July 15-21, 2007.* and included the description of *CLOnE QL*. In 2008, *CLOnE QL* was renamed to QuestIO before it was published in

- *V. Tablan, D. Damljanovic, K. Bontcheva: A Natural Language Query Interface to Structured Information. In Proceedings of the 5h European Semantic Web Conference (ESWC'08), Tenerife, Spain, June, 2008.* and
- *D. Damljanovic, V. Tablan, K. Bontcheva: A Text-based Query Interface to OWL Ontologies. In: 6th Language Resources and Evaluation Conference (LREC'08), Marrakech, Morocco, ELRA, May, 2008.*.

My contribution for the former paper is writing the initial paper draft. The evaluation was performed by K. Bontcheva and V. Tablan with my assistance and is not included in this thesis. In terms of implementation of the described system, I had initial guidance from V. Tablan, especially when developing the first version of the OntoRoot Gazetteer. He also contributed the backtracking algorithm which became part of QuestIO. With regard to

Natural Language queries to SeRQL (Sesame RDF Query Language) queries[3], which are then executed against the given ontology/knowledge base in order to return results to the user. In this chapter, we give details of QuestIO's design, implementation, and evaluation. First, we present the initialisation of the system, which is performed automatically from the ontology (Section 7.1). Details of the runtime processing of the query is described in Section 7.2, followed by the language coverage supported by QuestIO in Section 7.3. Section 7.4.1 presents the evaluation of correctness and the language coverage, Section 7.4.2 presents the evaluation of portability and scalability, while in Section 7.5 results from the user-centric evaluation are discussed.

7.1 Building the Domain Lexicon

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

- names of all ontology resources i.e. fragment identifiers[4], 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

the latter paper, my contribution was writing it in full along with the implementation of the system and the evaluation section. V. Tablan and K. Bontcheva read and commented on the pre-final version of the paper. Sections 7.1, 7.2, 7.3, 7.4.1, and 7.4.2 are updated versions of the latter paper.

[3] At the time of the initial implementation of QuestIO, we were using GATE ontology-based components which worked with Sesame 1.x and OWLIM 2.8.4. where SPARQL was not supported.

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

space. For example, `Project_Name` or `Project-Name` would become `Project Name`;

- any name that is written in *camelCase* style is split into its constituent words, so that `ProjectName` becomes `Project Name`.

Each item from this list is analysed separately by the Onto Root Application (see Figure 7.1). 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).

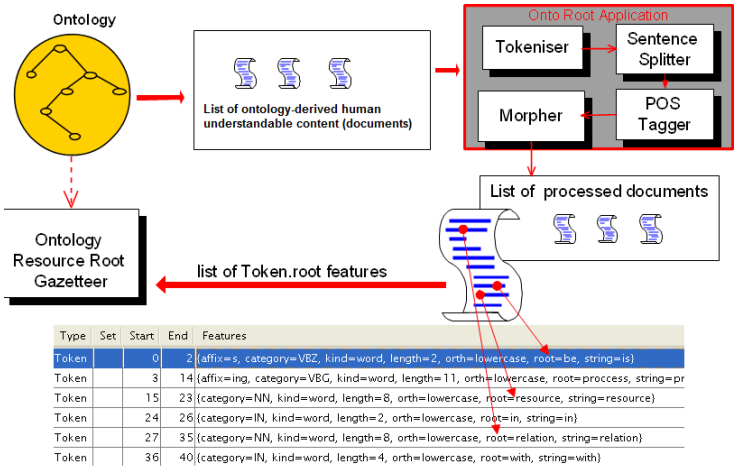

Figure 7.1: Building the domain lexicon from the ontology

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 the following strings:

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

Each of the items is then analysed separately and the results will 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 processing. 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 knowledge repository: the more human understandable descriptions attached to the semantic resources, the richer the lexicon, and hence better the coverage of the system. However, too many identical descriptions (e.g. exactly named entities of different type such as people who are named as locations, etc.) might cause low performance due to the high risk of ambiguity. In other words, we can expect a reasonable performance only with a very specific domains and manually crafted ontologies where ambiguities are not common. Even narrow domains can be problematic for QuestIO if the data contains a large amount of duplicate names. For example, in the domain of music a group called *The Who* with the label *Who* might be misleading for questions starting with *Who* such as in *Who are the members of the Who?*, in which case, the gazetteer will mark both WH-phrase *Who* and the proper noun *The Who* as referring to the same group, whereas the first annotation should ideally be disregarded. In general, the performance is expected to be better for semantic resources where the TBox is larger in comparison to the ABox. This is because the TBox defines concepts and relations, and in that context the ambiguity is avoided if the conceptualisation is carefully designed. The ABox contains individuals that can be named ambiguously, and this can be expected to degrade QuestIO's performance, due to fact that it performs a shallow text processing and heavily relies on the lexicalisations attached to the semantic resources.

7.2 Query Processing

QuestIO is an Information Extraction application, based on GATE, which:

- takes a free text query and an ontology as an input,
- transforms it to the set of queries expressed in a formal language, e.g. SeRQL, and
- returns a set of results returned after executing these queries against the given ontology.

Key components for the query interpretation and analysis are shown in Figure 7.2.

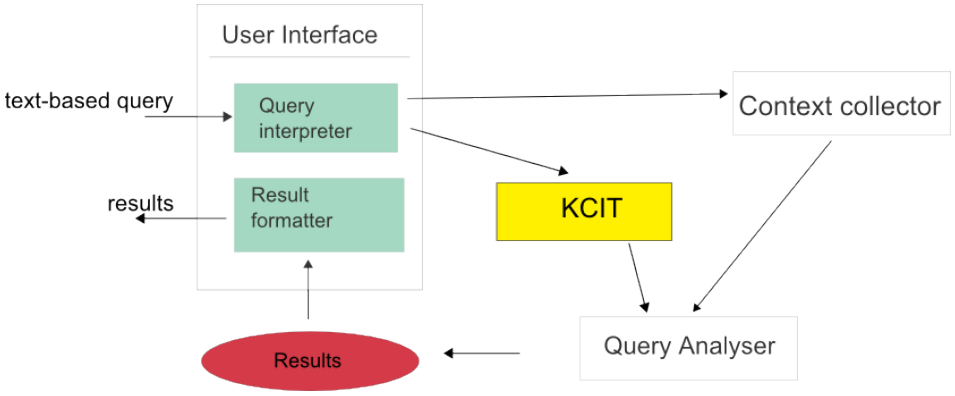

Figure 7.2: The QuestIO component diagram

7.2.1 Query Interpretation

Each user query is interpreted using the *Query Interpreter* in the *User Interface*. It is then analysed by two components, each of which represent a separate GATE pipeline application. Firstly, the Key Concept Identification Tool (KCIT) identifies key concepts inside the query (instances, classes, properties or property values from the ontology) by performing the *ontology-based lookup*. We process the query with the same language processing resources we used when extracting lemmas in the previous phase (Section 7.1),

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.* Additionally, the *Context Collector* would extract the text between all identified key concepts (i.e., chunks), which is in this case *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 next, combines the key concepts with key phrases, keywords, and chunks, in order to infer any potential relations that are defined between these concepts inside the ontology.

7.2.2 Query Analysis

When all relevant data are collected, the *Query Analyser (QA)* performs the following steps (Figure 7.3):

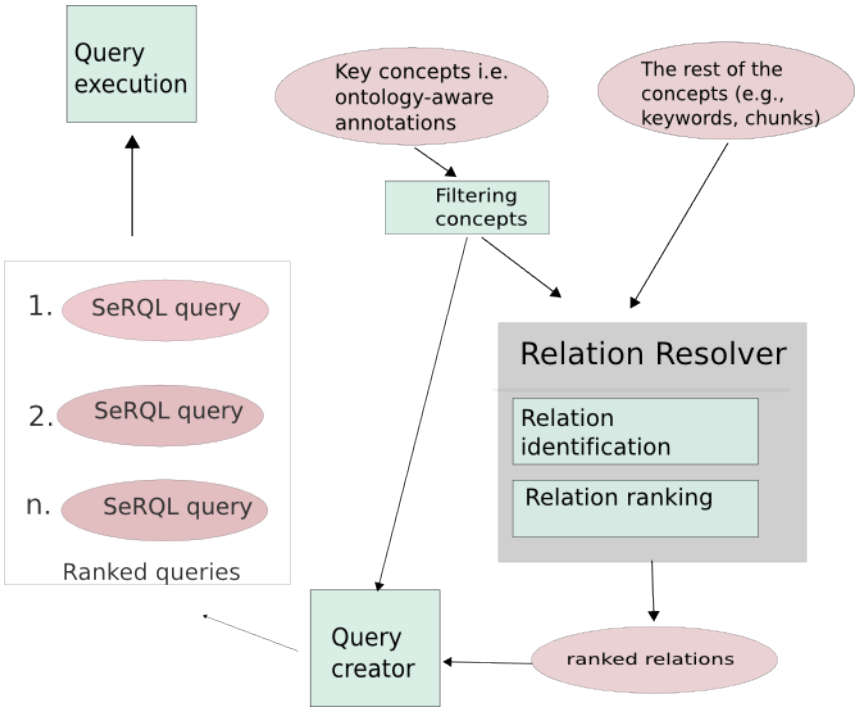

Figure 7.3: The Query Analyser module

1. *Filtering the identified key concepts.* With human language it is possible to use the same expression in different context and express totally different meanings (see e.g. 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 label *ANNIE POS Tagger* inside the GATE knowledge base[5]. The GATE knowledge base contains instances of classes and relations between them based on the GATE domain ontology[6]. (Note: as part of our evaluation, derived from the TAO project, we use data *about* GATE such as system documentation, mailing list and so on. This is not to be confused with our use *of* GATE to process the queries. See Section 7.4.1.) *ANNIE POS Tagger* refers to an instance of type *POSTagger*, which has label

[5] `http://gate.ac.uk/ns/gate-kb`
[6] `http://gate.ac.uk/ns/gate-ontology`

POS Tagger. If a query contains the text *ANNIE POS Tagger*, two 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.

2. *Identifying relations between key concepts.* This step includes identification of defined ontology relations (properties) between identified key concepts. These relations are very important as they add descriptions to the concepts and define their behaviour by adding rules and constraints. They are retrieved through ontology-based reasoning (we used the OWLIM[7] repository and its TRREE engine).

3. *Ranking potential relations.* Retrieved relations are then scored using a combination of three factors. One of them is based on string similarity and is called a *similarity score*. The other two relevant factors for scoring the properties are more complex and are based on the property's 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 (simScore) 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

[7] `http://ontotext.com/owlim/`

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 (specScore) reflects the position of the property in comparison to other existing properties in the ontology hierarchy. The intuition behind this score is that 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, a property *hasBrother* could be defined as a subproperty of the property *hasSibling* thus being more specific. The higher score is given to the more specific properties (see Figure 7.5). The specificity score is calculated as follows:

- All properties are arranged in two columns, where the first column is a property, and the second column is its direct super-property. To illustrate this with an example, let us assume that an ontology has six properties defined as illustrated in Table 7.1.

Property	**Direct super-property**
p1	n/a
p2	p1
p3	n/a
p4	n/a
p5	p4
p6	p5

Table 7.1: Properties and their direct super-properties as defined in an ontology

- For each property, its distance from the furthermost super-property is calculated. Following our example, the distance is as shown in Table 7.2.
- The specificity score is calculated by dividing the calculated distance for each property by the maximum distance among

Property	Distance
p1	0
p2	1
p3	0
p4	0
p5	1
p6	2

Table 7.2: Distance of each property from its furthermost super-property

all properties on the ontology level. For the example in Table 7.2, the maximum distance is 2, which appears between p6 and p4. Hence, the specificity scores are as shown in Table 7.3

Property	Specificity score (normalised distance score)
p1	0
p2	0.5
p3	0
p4	0
p5	0.5
p6	1

Table 7.3: Specificity scores

As we can see from the example, the minimum specificity score is assigned to the properties that have zero super-properties defined (p1, p3, and p4), while the maximum specificity score is assigned to the property with the largest number of super-properties defined (p6).

The Distance score (distanceScore) 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 at the same level of the property hierarchy, and the latter has a more specific domain which is `Man` while the domain for the former is `Person`. If we assume that the class `Man` is defined as a

sub-class of `Person` in the ontology, *hasBrother* will be assigned a higher value on the distance score, because properties with more specific domain and ranges are assigned a higher distance score. In essence, the distance score is an average of the *specificity scores* of all domain and range classes defined for the property. Hence, it is calculated as follows:

- For each property, find the domain and range classes.
- For each domain and range class calculate the *specificity score* following the same principles described above for calculating **Specificity scores** for properties, where the only difference is that we look at the super classes, instead of super properties.
- The distance score is the average of all specificity scores found.

The final score (FC) for each property is a weighted sum of the three measures and is calculated as shown in Equation 7.1:

$$FC = 3 * simScore + 1 * specScore + 1 * distanceScore \tag{7.1}$$

These metrics are ontology-motivated and are largely comparable to those used in the AquaLog system and many others.

4. *Creating SeRQL queries.* When all potential relations are scored and ranked, a formal SeRQL query 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. However, before generating the formal query, the elements are organized into a special format:

```
CLASS OR INSTANCE - [PROPERTY - CLASS OR INSTANCE]
```

where the part marked between '[' and ']' can be repeated N times, where N is the number of key concepts in the query subtracted by 1. The query interpretation with the highest score is then processed to generate the SeRQL query.

For example, if the query is *What are the countries located in Europe?*, the key concepts are *countries*, and *Europe*. The first three query interpretations after ranking relevant properties are:

```
COUNTRY locatedIn EUROPE (SCORE: 3.27)
COUNTRY hasOldName EUROPE (SCORE: 0.79)
COUNTRY hasMainAlias EUROPE (SCORE: 0.79)
...
```

The first interpretation has a very high score (3.27) due to *similarity score* for *locatedIn*, see Table 7.4. The *specificity score* is lowest (0.0) for this property because there is no defined super-property for *locatedIn* in the property taxonomy. However, both *hasOldName* and *hasMainAlias* are subproperties of *hasAlias*, and hence have a higher specificity score. The similar trend exists with regard to the *distance score*. This is due to the specificity scores of the domain and range classes for these properties. Namely, all properties have ENTITY class defined as domain, while the range for *locatedIn* is higher in hierarchy in comparison to the range of the other two properties and hence the lower distance score, see Figure 7.4.

Property	*locatedIn*	*hasOldName*	*hasMainAlias*
Domain Class	ENTITY	ENTITY	ENTITY
Range Class	LOCATION	ALIAS	ALIAS
Specificity Score	0.0	0.25	0.25
Similarity Score	1.0	0.0	0.0
Distance Score	0.27	0.54	0.54
Total score	3.27	0.79	0.79

Table 7.4: Calculating individual scores for the candidate properties

The SeRQL query generated accordingly from the first ranked interpretation is as follows:

```
SELECT c1, p2, i3

FROM
{c1} rdf:type
```

Figure 7.4: Position of ENTITY, LOCATION, and ALIAS classes within the PROTON ontology. ENTITY is most generic as it does not have any super-classes (owl:Thing excluded), followed by ALIAS that has one super-class LEXICAL RESOURCE, followed by LOCATION which is most specific as it has two super-classes (OBJECT and ENTITY)

```
{<http://proton.semanticweb.org/2005/04/protonu#Country>},
{c1} p2 {i3},
{i3} rdf:type
{<http://proton.semanticweb.org/2005/04/protonu#Continent>}

WHERE
p2=<http://proton.semanticweb.org/2005/04/protont#locatedIn>
AND
i3=<http://www.ontotext.com/kim/2005/04/wkb#Continent_T.4>
```

The dynamic creation of formal queries makes QuestIO flexible, yet easily extendable towards any other formal query language e.g., SPARQL.

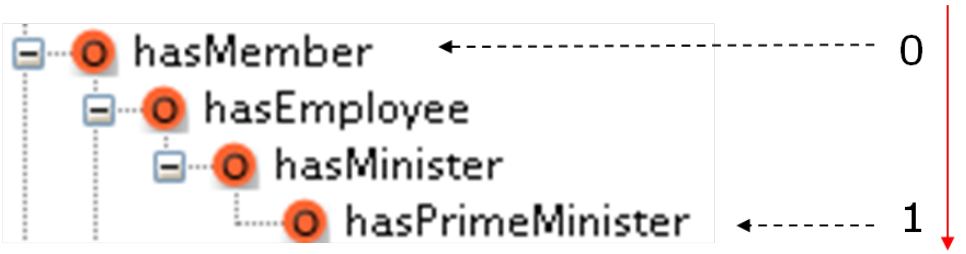

Figure 7.5: Specificity score for properties

7.3 Coverage

Query processing in QuestIO starts by identifying key concepts in the query based on the lexicon extracted as described in Section 7.1. It performs exact string matching between the lemmas of the extracted lexicalisations and the lemmas found in the query. This improves robustness of the system so that all morphological inflections such as *city* and *cities* in the query are identified, no matter which of the two forms exist in the repository. However, the system does not extend the vocabulary by any external resources such as WordNet and solely relies on the lexicalisations attached to the semantic resources.

As QuestIO works by recognising key concepts inside the query before performing other disambiguations and interpretations, the number of concepts in the query is not limited. As long as there are relevant relations between the concepts in the ontology, the required formal query can be created and the results returned. An example is shown in Figure 7.6 where in the given query three concepts are identified when run 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.

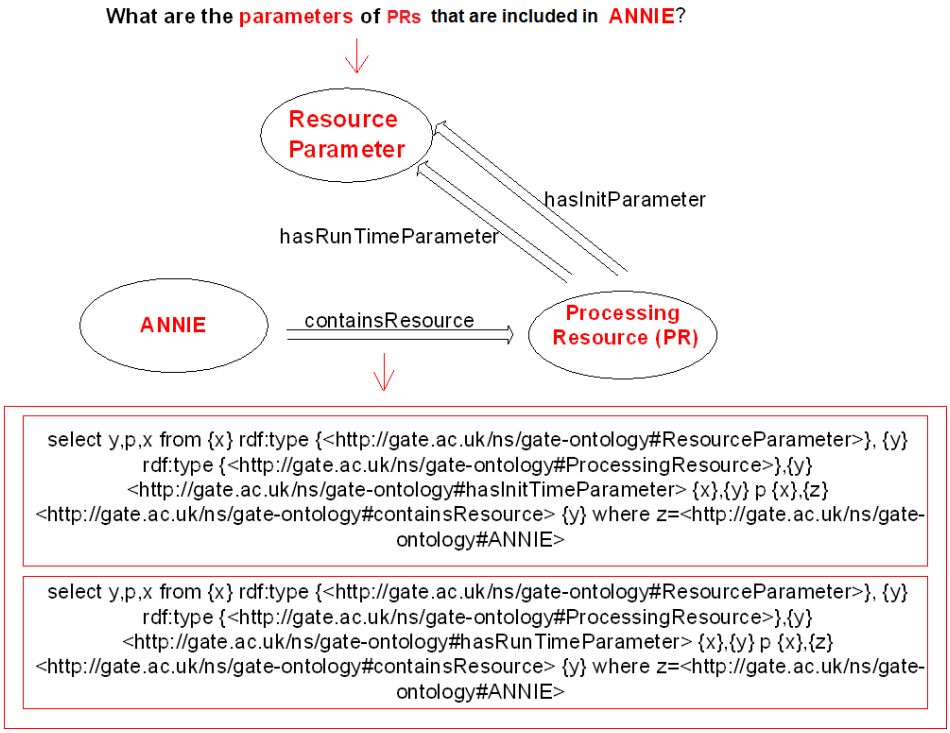

Figure 7.6: Supporting relative clauses with QuestIO

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 a 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, regardless of the first query being in the form of a fully-fledged question, and the latter more similar to a concept-based search 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 key concepts identified 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 does not affect 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, QuestIO supports queries including conjunction and disjunction (see Figure 7.7). These types of queries are processed so that first, 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.

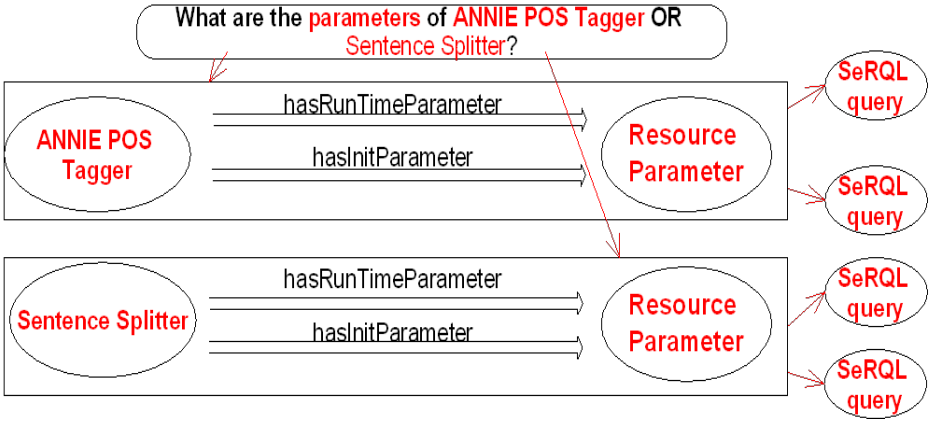

Figure 7.7: Supporting queries expressing conjunction/disjunction with QuestIO

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, and SeRQL queries are created accordingly.

To illustrate the flexibility of the QuestIO's supported syntax we give an example in Figure 7.8. For three different input queries expressed in the Natural Language or using incomplete queries, the result will be the same. The *inverse property* visible at the bottom of the figure where the results are shown, are indicating that the *hasCapital* property has *French Republic* as a domain, and *Paris* as a range; however, due to the flattening effect of the graph, we show results this way.

7.4 Qualitative and Quantitative Evaluation

In this section we present two kinds of evaluation performed with QuestIO. The first one is *comparative*, demonstrating the advantages/disadvantages

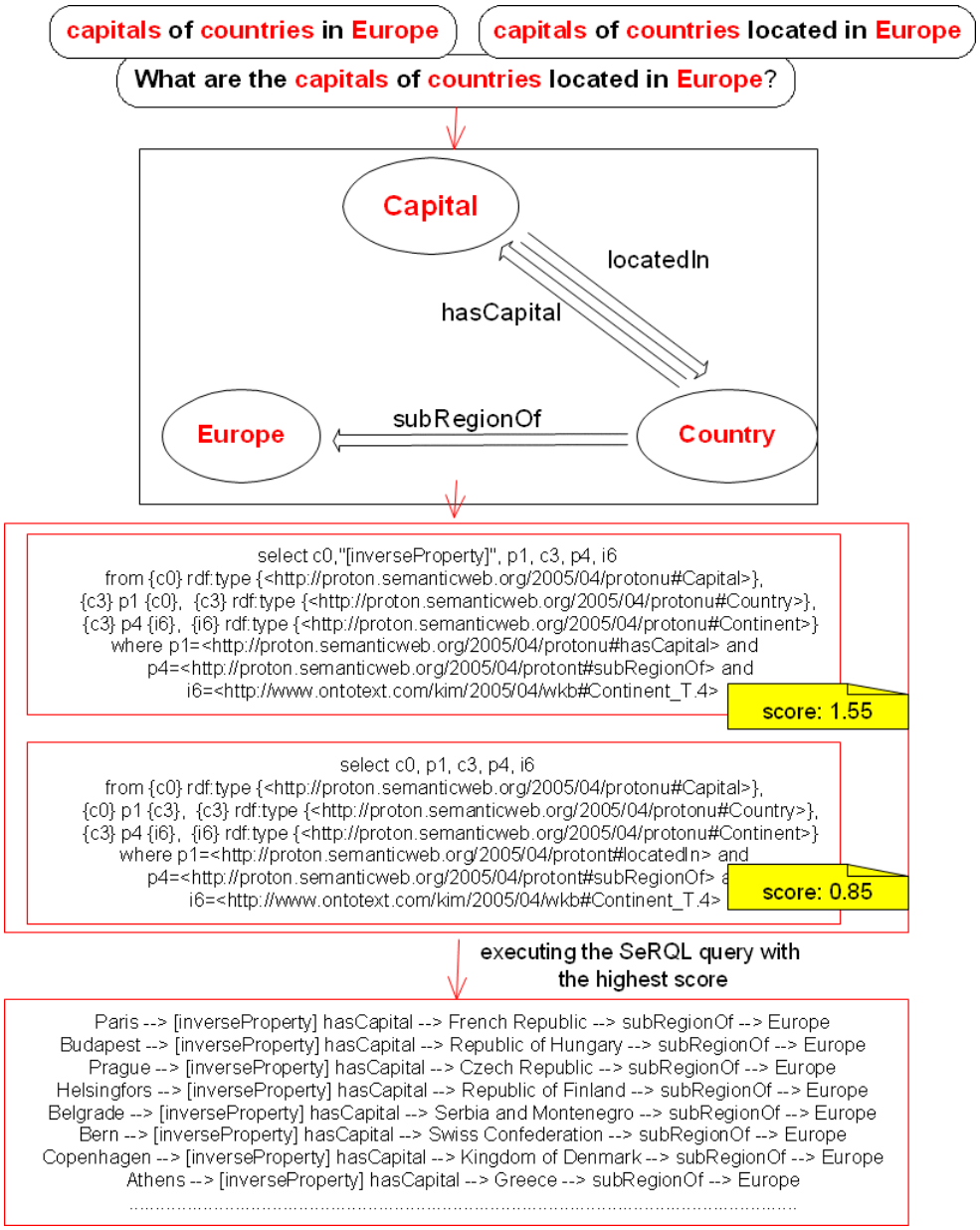

Figure 7.8: Expressiveness: QuestIO returns the same result for three different variations of the input query

of QuestIO's language coverage compared to that of AquaLog [Lopez and Motta, 2004, Lopez et al., 2007]. 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, thus demonstrating how the size of the knowledge base affects the query execution time.

7.4.1 Correctness and Coverage

We evaluate *correctness* using *precision* and *recall* measures (see Section 4 for definitions). We calculate these measures for the two systems: QuestIO and AquaLog using the same evaluation conditions. We chose the AquaLog system for two reasons. Firstly, our main intention with this comparative evaluation is to assess the coverage of the language supported by QuestIO, in comparison to a system with a deeper linguistic analysis. AquaLog satisfies this requirement. Secondly, as QuestIO does not require any customisation when porting from one system to another, the requirement for the system to be tested against was also to be able to automatically load the desired ontology without any additional configuration.

Dataset

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[8] project and is available from `http://gate.ac.uk/ns/gate-kb`. This ontology encodes the component model of GATE, the available plugins, the types of modules included in each of the plugins, the parameters for the different modules, and the like. The resulting ontology contains 42 classes, 23 object properties and 594 instances.

Firstly, the questions are filtered so that those enquiring about information that is not in the ontology, are excluded (14 out of 36 questions). The reason for excluding these questions is that they would not be answered correctly due to the lack of the knowledge, neither the relevant SPARQL queries could be generated. Hence, it is our view that it would be inappropriate to report

[8] `http://www.tao-project.eu`

the results of the system based on them. On the other hand, if the new knowledge would have been added to the ontology, the system might return the correct answer. However, this we can not judge without running the experiment with the updated knowledge base. The remaining 22 questions are used in the experiment.

Results

Results are categorised as follows:

- *correctly* answered;
- *correctly answered after reformulation*: when the original question is re-formulated the returned answer was correct;
- *partially correct*: the generated queries missed out one of the required constraints, so the answer was less precise (these answers were not included when calculating the overall precision and recall values);
- *failed* queries: when either no query is generated or the generated query is not correct.

Further, we divided these 22 questions into two groups:

1. All questions that were malformed or are not supported by AquaLog [Lopez and Motta, 2004, Lopez et al., 2007], among which there were
   - one conjunction query *What are the run parameters of POS Tagger and Sentence splitter?*
   - one query with brackets *Does GATE have a coreference resolution component (PR)?*
   - one query starting with *How many...*
   - three queries not in the form of a full-blown question, for example *I cannot get WordNet plugin to work.*
2. Full-blown correctly structured questions (16 queries)

Regarding group number 1, out of the 6 questions QuestIO was able to correctly answer four queries, one question was answered partially, and one failed.

Regarding group number 2, all questions were executed using the AquaLog system, and then using QuestIO. The results are shown in Table 7.5. QuestIO seems to perform better than AquaLog, if we consider only correctly answered questions when calculating precision and recall (64.28% vs.45.45%, and 56.25% vs. 31.25% respectively). 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 (resulting in increased precision from 45.45% to 72.72% and recall from 31.25% to 50%). For example, *What are the values of the POS Tagger parameters?* was correctly answered by AquaLog when reformulated to *What are the parameters of the POS Tagger?*, whereas both versions of the query were handled correctly by QuestIO.

Table 7.5: Results of running the same set of queries with QuestIO and AquaLog: *c. correct - conditionally correct (correct after reformulated), p. correct - partially correct*

	QuestIO	**AquaLog**
correct	9 (56.25%)	5 (31.25%)
c. correct	0	3 (18.75%)
p. correct	5 (31.25%)	3 (18.75%)
failed	2 (12.5%)	5 (31.25%)
precision	64.28%	45.45%/ 72.72%
recall	56.25%	31.25%/50%

If we consider conditionally correct answers in the results, AquaLog outperforms QuestIO in terms of precision, while QuestIO outperforms AquaLog in terms of recall. This is due to the fact that if a question is formulated following the AquaLog's supported language, the system seems to be more precise and for the questions which it can not answer correctly it fails more often than QuestIO, while QuestIO returns partially correct answers more often than it fails. This highlights the advantages and disadvantages of the languages supported by the two systems. AquaLog analyses grammar of the question more carefully while requiring the user to know what kind of questions are supported. QuestIO relies on a shallow language processing

and the pattern-matching while not requiring a strict adherence to syntax. In that regard, AquaLog seems to be more suitable for a question answering system which would be used for searching the knowledge base to find precise facts, while QuestIO seems to be better for browsing through the available knowledge and looking into whether different concepts are related to each other.

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 too long and complicated. 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.

Moreover, the AquaLog system has a better interface than QuestIO. For example, in cases when the result is an ontology instance only, it is possible to examine all assigned properties for this instance. In QuestIO, it is only possible to see the name of the instance, and therefore the user has to browse the ontology itself in order to find more details (see the lower part of the previously discussed Figure 7.8 which demonstrates how QuestIO renders results). Additionally, in the case of disambiguation, AquaLog will prompt the user with a dialogue, whereas QuestIO would automatically derive the result which is ranked best, or in case of several equal scores, it would return all of them without requiring any input from the user.

7.4.2 Portability and Scalability

To test *scalability*, in another experiment we trialled QuestIO with two different knowledge bases of different sizes, one being a subset of the other. We prepared a set of queries that return identical results when executed against the two knowledge bases. This is because the goal was to test how the size of the dataset influences the *execution time* of the query.

The smaller dataset is the Travel Guides Knowledge Base (KB) that contains instances and relations between them from the Travel Guides (TG) Ontology[9]. The TG ontology is an extension of the PROTON ontology[10]

[9] `http://goodoldai.org/ns/tgproton.owl`
[10] `http://proton.semanticweb.org`

and contains data about tourism destinations and tourist preferences (see [Damljanovic and Devedzic, 2008] for more details).

The core of the Travel Guides Knowledge Base contains geographical data such as those about cities, countries and continents [Damljanovic and Devedzic, 2009]. This core was extracted from the KIM KB [Popov et al., 2004] which contains general data, specifically about organisations, people, locations, and has about 40 times more resources than the Travel Guides KB. The size of both knowledge bases in shown in Table 7.6.

During this experiment, the two knowledge bases did not need to be changed or customised to work with the QuestIO system, thus demonstrating *portability*. The set of queries chosen were of different level of complexity, where the *complexity* is directly proportional to the number of identified key concepts. The experiments were run on Ubuntu 9.10, on a computer with dual Intel(R) Core (TM) 2 CPU 6400@2.13 GHz with 8G of memory. We have also repeated all experiments five times and here we report the average numbers.

As shown in Table 7.6, the initialisation time of QuestIO was much longer when used with KIM KB. However, this step is performed only once.

Table 7.6: Knowledge Base Sizes

	TG KB	KIM KB
Classes	364	335
Object Properties	120	90
Datatype Properties	47	43
Instances	2816	122885
Total size (C + P + I)	3347	123353
Initialisation time	11.68 seconds	318.4 seconds

The execution times were between 2.85 and 59.09 times (average: 13.7) longer when executed with KIM KB. Still, most of the queries were executed within a few seconds, excluding some exceptional cases. These are visible from Figure 7.9.

The query *is London capital of any country?* took longest 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.

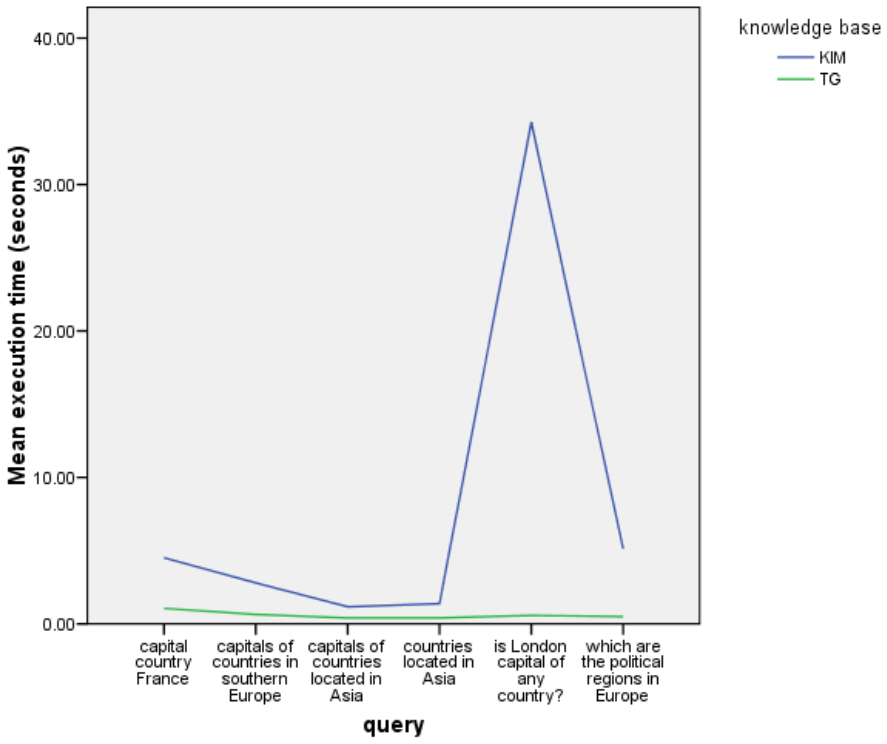

Figure 7.9: The average execution time across five runs, using the TG and KIM knowledge bases

Table 7.7: Execution times for running the same set of queries with QuestIO. Shown times are in seconds.

	TG	**KIM**
Queries	**Execution time**	
countries located in Asia	0.41	1.38
capitals of countries located in Asia	0.41	1.17
which are the political regions in Europe	0.50	5.3
is London capital of any country?	0.58	34.27
capitals of countries in southern Europe	0.65	2.8
capital country France	1.06	4.52
Average execution time	0.60	8.21

7.5 User-centric Evaluation

In order to test usability of QuestIO, we have developed a prototype for access to documentation about GATE software and then conducted a user-centric task-based evaluation[11].

7.5.1 QuestIO Prototype

The prototype interface had one text box for a query and a button. Each page always had a link to browsing the ontology, so that the users who are more comfortable with the structured format could use it (see Figure 7.10). After the user submits a query, the results are shown in a *document pane* (the pane where the results are URLs of documents mentioning concepts from the query), and a *refinement pane* which is used to either refine the set of returned documents in the document pane, or to provide an answer.

[11]The QuestIO prototype described in Section 7.5 is published in *D. Damljanovic, K. Bontcheva: Enhanced Semantic Access to Software Artefacts. In Workshop on Semantic Web Enabled Software Engineering (SWESE'08) held in conjunction with ISWC'08, Karlsruhe, Germany, October, 2008.*. My contribution was the full implementation of the system and writing the paper. K. Bontcheva provided comments on the pre-final version of the paper.

Section 4 in *H. Wang, D. Damljanovic and J. Sun: Enhanced Semantic Access to Formal Software Models. In the Proceedings of the 12th International Conference on Formal Engineering Methods, Shanghai, China, November 16 - 19, 2010.* is largely based on the OntoRoot Gazetteer and KCIT tool described in Sections 7.1 and 7.2 in this chapter. My other contribution to the paper is the description of the ontology in Section 3.2 which goes beyond the scope of this thesis. H. Wang and J. Sun contributed the remaining sections.

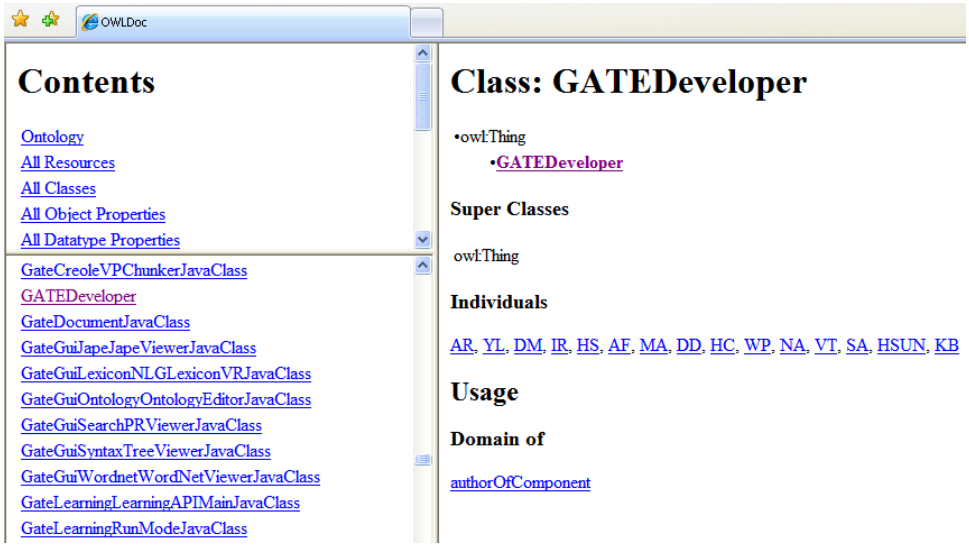

Figure 7.10: Browsing ontology to find GATE developers

Figure 7.11 illustrates *refinement pane* that shows the answer to the user's question which is expressed in Natural Language. However, if he selects one of the named POS taggers and clicks *Refine*, the system will show only the documents with the selected type of the POS tagger (see Figure 7.12).

Another example, for a question which is not expressed as a full-blown Natural Language question, but rather as its fragment, is shown in Figure 7.13. The document pane lists all documents which contain the term *GATE developer* based on the user's query. However, the *refinement pane* lists the names of the developers as they appear in the ontology (see Figure 7.14), and the user can now select one of the names and the list of documents would be refined (see Figure 7.15).

Due to scalability issues with QuestIO working with SeRQL and OWLIM 2.8 (which was relying on Sesame 1.x), we have extended QuestIO in order to be able to translate NL to SPARQL, as the SPARQL implementation was more optimised and the answers were returned several times faster in comparison to SeRQL. This significantly improved the performance of QuestIO with the more-than-a-million-triples dataset, which had to return an answer in subseconds when working with real users. Hence, in the experiments, we used the version of QuestIO which worked with OWLIM 2.8.4, Sesame 1.2

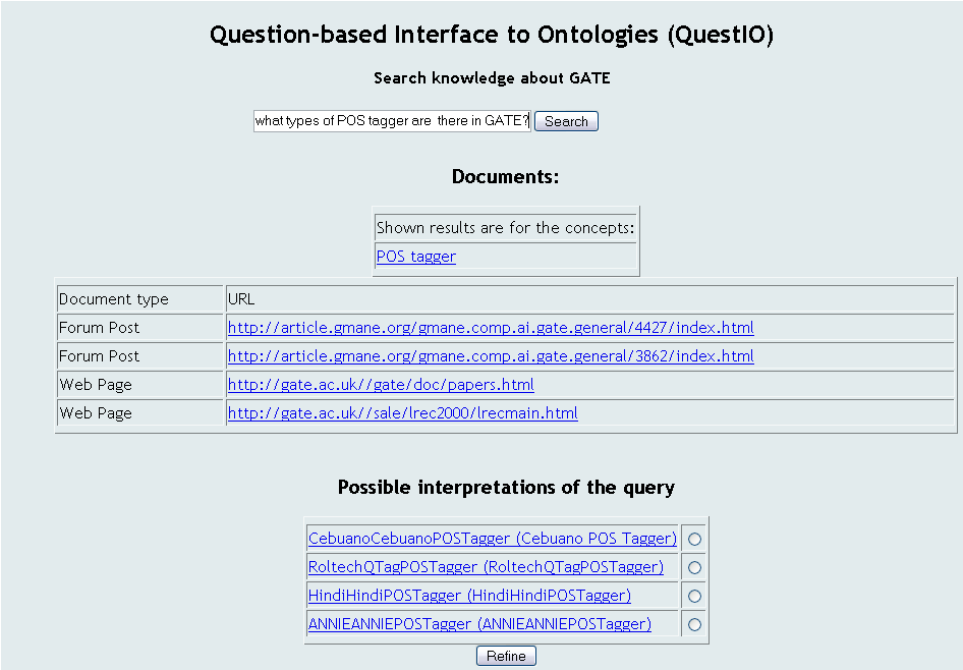

Figure 7.11: Using the QuestIO prototype to find the answer to the query *What types of POS Tagger are there in GATE?*. The *document pane* lists documents about *POS Tagger*, while the *refinement pane* lists the answer to the question which can also be used to find more specific documents

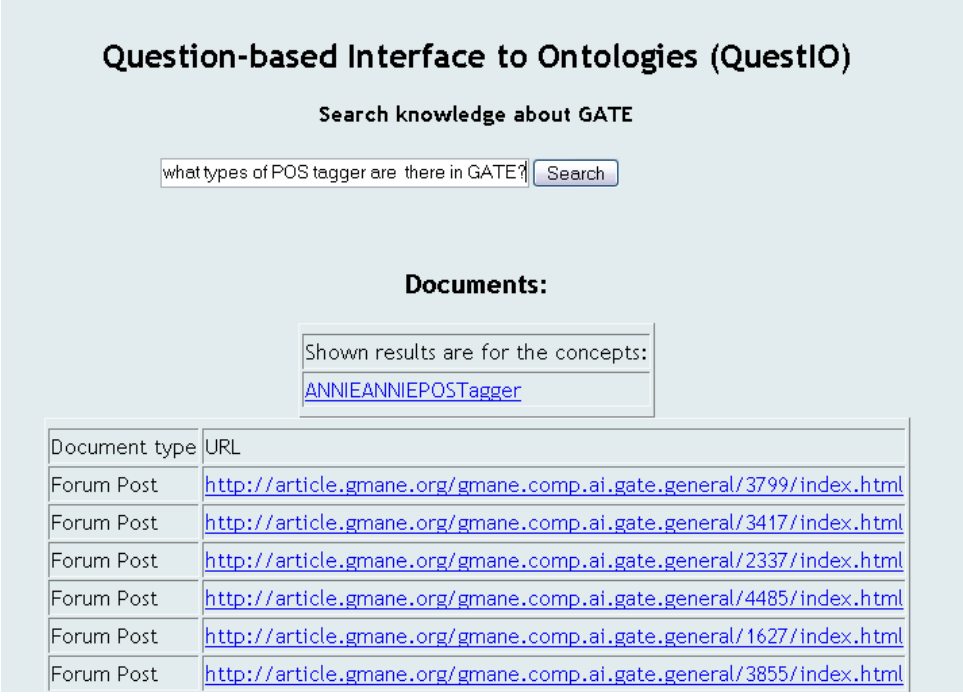

Figure 7.12: Documents about ANNIE POS Tagger

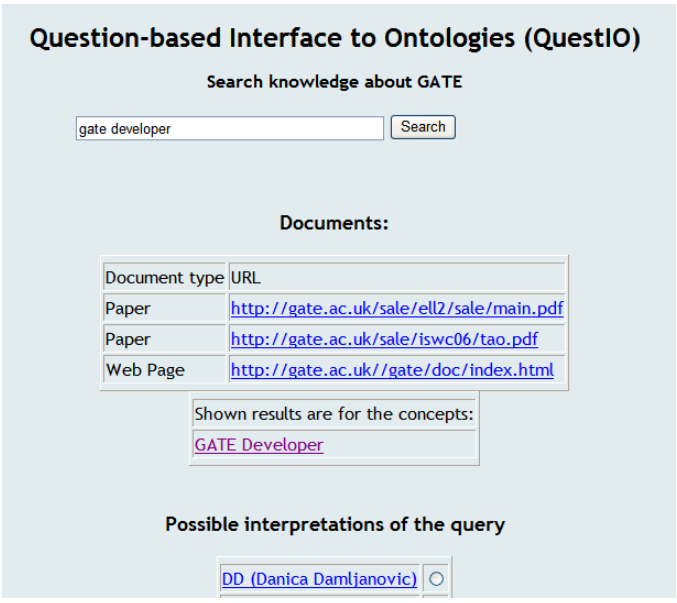

Figure 7.13: Searching for GATE developers using QuestIO

Figure 7.14: The refinement pane showing the list of GATE developers

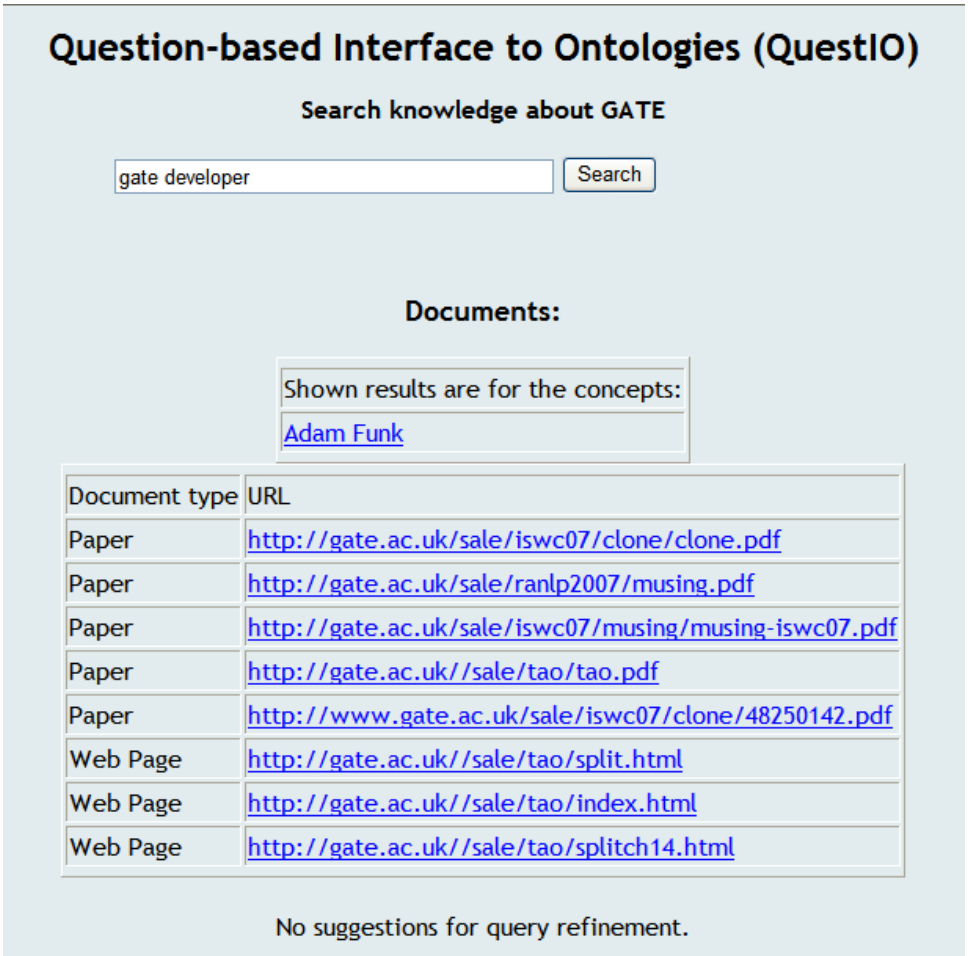

Figure 7.15: Refined results after selecting *Adam Funk* from the list of developers

and SPARQL.

7.5.2 Dataset

We have used the GATE domain ontology which describes concepts about the GATE software, modules, components, and developers (see Section 7.4.1 for more details). This ontology has been used to annotate existing software documentation and code about GATE such as source code, forum posts, Web pages linked from `http://gate.ac.uk`, and any documentation that was available. Annotations are exported into OWL statements, based on an Information-Extraction ontology[12]. By exporting information about annotations and documents in which these occurred, we could perform not only concept-based search which would discover relations available in the GATE ontology (between the GATE components such as a GATE PR and a parameter for example), but also the documents which contain annotations referring to certain GATE concepts. For example, we could find *documents* which mention *POS Tagger*. Table 7.8 describes the size of the dataset.

number of annotated documents related to GATE	10 070
number of generated annotations	183 127
number of statements in the GATE ontology	3948
number of Information Extraction statements	1 138 847
total number of statements	**1 142 795**

Table 7.8: Size of the dataset

7.5.3 Evaluation Scope

The aim of this qualitative evaluation was twofold. Firstly, we wanted to test the usability of QuestIO. Secondly, we wanted to validate whether software developers, who are often not experienced in Semantic Web technologies and formalisms, are able to easily find all information relevant to their tasks by using QuestIO. This gives us an insight into the feasibility of using such an interface in comparison to the baseline which in our case were traditional ways of search e.g, Google, at least in the case of technologically literate subjects.

[12]We have used the Proton ontology: `http://proton.semanticweb.org/`

7.5.4 Experimental Setup

We carried out a *complete counterbalanced*[13] *repeated measures*,[14] *task-based* evaluation design (also called a within-subjects design), i.e., the same users interact with QuestIO and also use their current working practices and tools, in order to complete a given set of tasks. With only two conditions in our case (baseline and QuestIO) one half of the subjects was asked to accomplish each task first using QuestIO, and the other half was asked to do it using traditional ways of searching before doing it using QuestIO. As 12 subjects were involved in the experiment, this enabled us to cover each possible order 6 times.

As part of the experiment we collected relevant background data (e.g., experience with GATE, familiarity with semantic web concepts) by asking participants to fill in a *multiple choice pre-test*. Participants then had to perform four tasks using both QuestIO and their usual working methods, while we recorded their sessions. After each task they were asked to fill-in a questionnaire assessing various features of the QuestIO prototype. At the end, they had to fill in an overall user satisfaction survey and reply to questions related to the interface and the query language. Questionnaires used in this experiment are available in Appendix A.

Training

Before the experiment, each participant watched two videos:

- a video introducing the aim and purpose of the QuestIO prototype, mentioning its motivation and main objectives (1 min 52secs)[15]

[13] In complete counterbalancing each of the possible orderings of the experimental conditions is equally represented. If k is the number of conditions, k! is the number of orderings (see Wuensch [2009]). The reason for using counterbalancing is to minimise the sequence effects – results might be contaminated by practice effects e.g. subjects get better at the task as time passes, fatigue effects e.g., subjects get tired as time passes.

[14] *"In this design all subjects appear in both experimental conditions, so halving the number of subjects needed."* [Preece et al., 1994, p. 647]

[15] `http://www.tao-project.eu/researchanddevelopment/demosanddownloads/movies/gate-case-study-with-refs/gate-case-study-with-refs.html`

- a video explaining the language supported by QuestIO, and also the prototype interface (5 min 37 secs)[16]

After watching the training videos and before being given their tasks, a half of subjects were shown a short introduction:

> *The goal of this prototype is to enable a single point of access to all knowledge contained in GATE software documentation and code: forum posts, source code, source documentation, Web pages and publications accessible from the GATE website. You will be asked to perform several tasks using first this prototype and then the tools you use on a daily basis, e.g. the GATE support page, Google, Sourceforge.*

Another half received a similar introduction, with the difference that they were asked to use the tools they usually use on a daily basis, and then to use the prototype.

Software

In order to collect and analyse the user interactions, we used the Morae software[17] which comprises of:

- *Morae Recorder*: to set up the study and record the user,
- *Morae Observer*: to observe the user while performing the tasks, and
- *Morae Manager*: to analyse results.

What to measure?

As this is primarily a usability study, we measured:

- *efficiency*: time spent to complete the tasks using the two approaches – baseline and QuestIO;

[16] `http://www.tao-project.eu/researchanddevelopment/demosanddownloads/movies/prototype-tutorial/prototype-tutorial.html`

[17] `http://www.techsmith.com/morae.asp`

- *effectiveness*: the percentage of completed tasks using the two approaches;
- *user satisfaction*: a SUS questionnaire as a standard satisfaction measure [Brooke, 1996] to test usability of QuestIO.

In order to measure more subjective opinions after finishing each task, we asked each participant to fill in a *questionnaire* reporting about their experience with the prototype. As argued by Nielsen [1994], subjective satisfaction needs to be measured since it is an important usability attribute. We asked all subjects the following questions:

- Were the results returned by QuestIO relevant?
- Did they find the refinement pane helpful?
- Was ontology browsing helpful?

We have also asked them to rank the overall experience with the QuestIO prototype in comparison to their every day working practices, and also to give their opinion on whether it was easy to formulate the queries required by QuestIO.

As a set of answers the subject were offered a pre-defined set based on the Likert scale [Likert, 1932] ranging from 1 to 5 (where 1 is Strongly Disagree and 5 is Strongly Agree).

Sample

As the dataset was related to the GATE software, we asked the members of the GATE team in Sheffield to participate in our experiment. We are aware that this choice of evaluation participants could raise questions over their objectivity, however our choice was limited due to the requirement that all our users had knowledge of GATE. On the other hand, proximity, high motivation and expertise of the group meant that we were able to obtain a rich set of evaluation results (see Section 7.5.6) which influenced our decisions related to the second part of our work (Chapters 8 and 9).

Most of the subjects belonged to both, or one of these groups:

- *GATE developers*: a group of people who are actively working (or used to work) on maintaining GATE and developing new GATE software components.
- *GATE users*: a group of people who are using GATE to perform language processing tasks such as syntactic parsing.

7.5.5 Tasks

A key point when designing the tasks was to make them similar to everyday tasks carried out by GATE developers and users. We have also verbalised them in a way that, if copy-pasted as such, the results will not be found. The tasks were as follows:

1. Find out what POS (Part-Of-Speech) taggers exist in GATE.
    - *Our assumptions*: this task is designed to test the effectiveness of the semantically enabled prototype; answers are available both on the Web, and through the prototype, but the idea is to test how efficiently this task can be performed using the two approaches. We would also like to see whether the subjects struggle with the query language of QuestIO and whether those that are familiar with the ontologies would turn to browse the ontology instead of using the text-based interface.
2. Imagine that you are a GATE developer who needs to extend the Cebuano Gazetteer. Your task is to find out the names of all its runtime parameters.
    - *Our assumptions*: this task is designed to resemble those performed on a daily basis by GATE developers. More importantly, it is not possible to find the correct answer on the Web as it is not available even in the source code: the developers have to know the specific place to look for the configuration files, which contain the answer. However, this knowledge is available in the ontology.
3. Find which forum posts are related to the Learning PR.

- *Our assumptions*: this task is designed to emphasis the benefits of concept-based search in comparison to the keyword based search currently available through search engines such as Google. While it is possible to complete this task using traditional search engines and also `sourceforge.net`, we hope that the subjects will get more relevant results quicker, through using the QuestIO interface. Also, we would like to see if the users will use the *refinement pane* in this task, because *learning* is ambiguous in the ontology and the pane can be used to filter out the irrelevant results.

4. Think of any task that you would like to perform using this prototype. For example, find documents which you have written. Try using the refinement pane.

    - *Our assumptions*: with this task we test whether the participants have learned how to use the prototype and whether they can formulate text-based queries without any difficulties; in addition, we want to collect a new set of questions which can be used to test and extend QuestIO's capabilities.

7.5.6 Results

Subjects and their background

With the pre-tasks questionnaire we wanted to assess the background knowledge of subjects and get an insight on how much they can be considered GATE domain experts, but also how much they know about the semantic web, ontologies, semantic search and ontology editors. We have also asked them about traditional ways of searching about problems related to GATE: where they search for help and how often. This gives us insight into the feasibility of the QuestIO prototype which gathers all knowledge about GATE at one place. Figure 7.16 shows the distribution of their answers. Each subject had a choice to select all that apply, not only one from the list of the options. Most of the subjects reported that they use the GATE support page (`http://gate.ac.uk/support.html`) as the starting point when

trying to learn about GATE or solve a particular problem, but as many of them reported that they use mailing list and the source code. We can see that 6 out of 8 available methods are indeed using different locations on the Web where subjects search for information about GATE. The lower part of Figure 7.16 shows how often the subjects reported to be in need to search for various GATE-related information. This gives us insight into suitability of subjects for our study. More than half of our subjects are in need to search for information about GATE at least once per week, making them highly relevant for our study.

Figure 7.17 shows how often the subjects are asked about various GATE components. This is important assessment about their GATE expertise, as it is already shown in Figure 7.16 that in 33% of the cases the subjects reported that they usually ask the member of the GATE team when enquiring information about GATE.

Finally, Figure 7.18 shows their experience of using GATE expressed in years. The most common experience was either *5 years or less* and *10 years or less*, which represents the half of the participants. 16.67% of the participants have never used GATE before.

We have calculated overall GATE expertise based on the linear combination of these three assessments: how often the subjects are asked about GATE (never: 0 points, rarely: 1, 1-2 days per week: 2, 3-5 days per week: 3 points), for how many years they have been using GATE (never used GATE: 0 points, rarely: 1 point, less than 2 years: 2, less than 5 years: 3, less than 10 years: 4, less than 14 years: 5) and also whether they consider themselves as GATE users (1 point), GATE developers (2 points), both (3 points) or neither (0 points). In the scalar representation of the latter, the GATE User was considered to be less of an expert than GATE developer; this might not be true outside of this evaluation however, we are primarily concerned with finding and searching GATE components which is something GATE developers do more often than GATE users. Figure 7.19 shows the results.

In order to assess their knowledge about semantic web technologies we have asked them about their familiarity with the terms semantic web, ontologies, semantic search and also whether they have ever used the ontology editors.

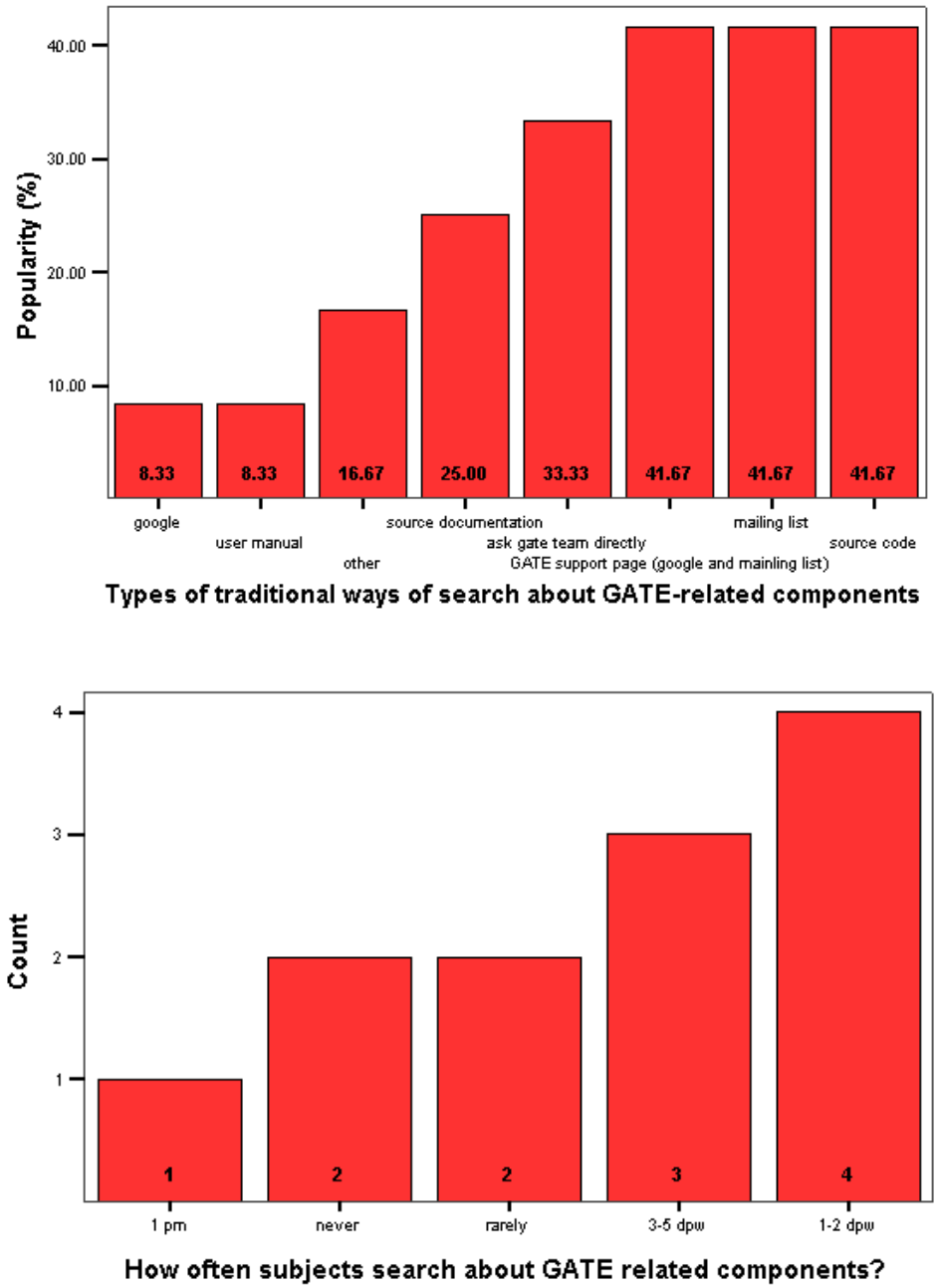

Figure 7.16: Traditional ways of searching about GATE components and frequency of search as reported by 12 subjects: 1 pm (per month), 1-2 dpw (days per week), 3-5 dpw (days per week)

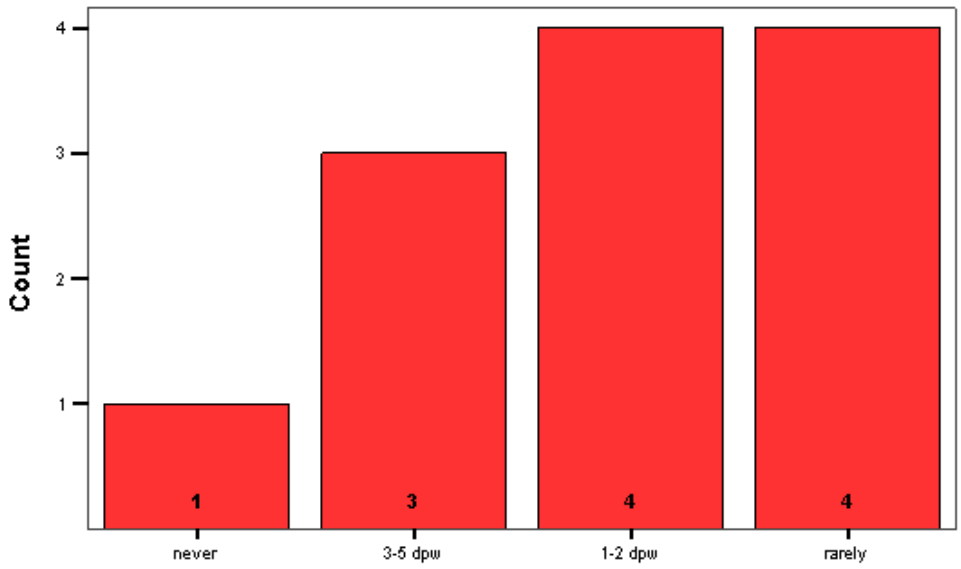

Figure 7.17: Frequency of *being asked* about the GATE-related components, as reported by 12 subjects: 1 pm (per month), 1-2 dpw (days per week), 3-5 dpw (days per week)

The normalised results are shown in Figure 7.20[18]. From this figure we can derive that majority of the subjects were unfamiliar with the semantic web related terms as the most common answer was familiarity 0 (the mode), while the median and mean were 16.67 and 25.69 respectively.

[18] Answers were chosen from a 5-point Likert scale, see Appendix A. The final results are calculated as *score minus one* so that 1 (Strongly Disagree) scored 0 points, 2 scored 1, and so on, where 5 (Strongly Agree) scored 4. We then normalised the score on the scale from 0 to 100, similar to the way SUS Scores are normalized as described by Brooke [1996]. There are many discussions in literature on whether Likert scale should be interpreted as interval or ordinal data. While the opinions are mixed, it seems to be dependent on the design of the experiment, and indeed on what is being measured by the scale. See Jamieson [2004] for more details. Throughout this thesis we interpreted Likert scale using both ways, depending on the circumstances and the goal of the experiment in question.

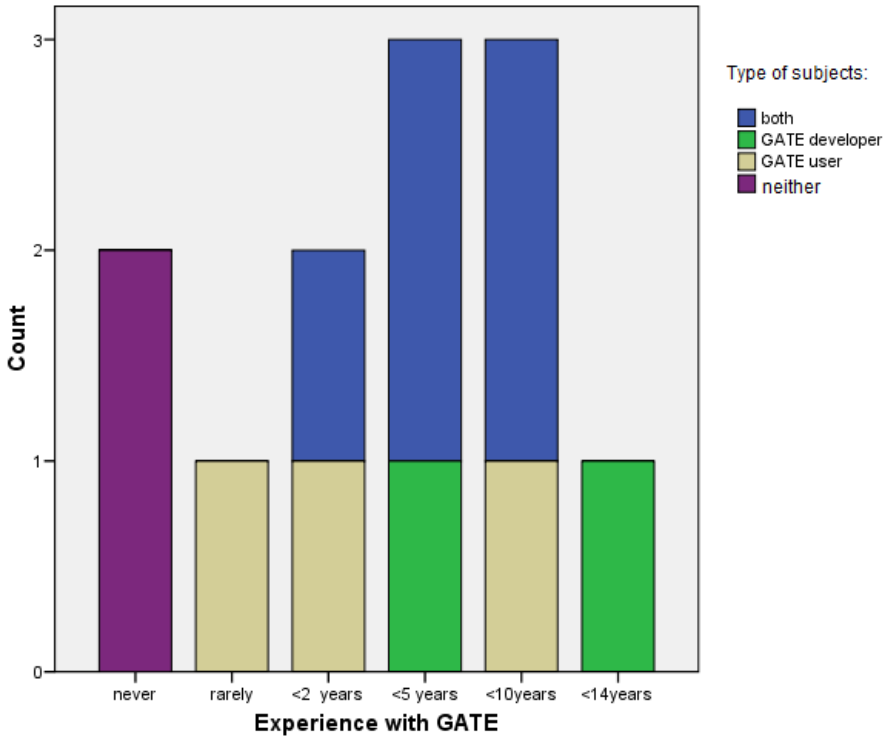

Figure 7.18: Experience in using GATE

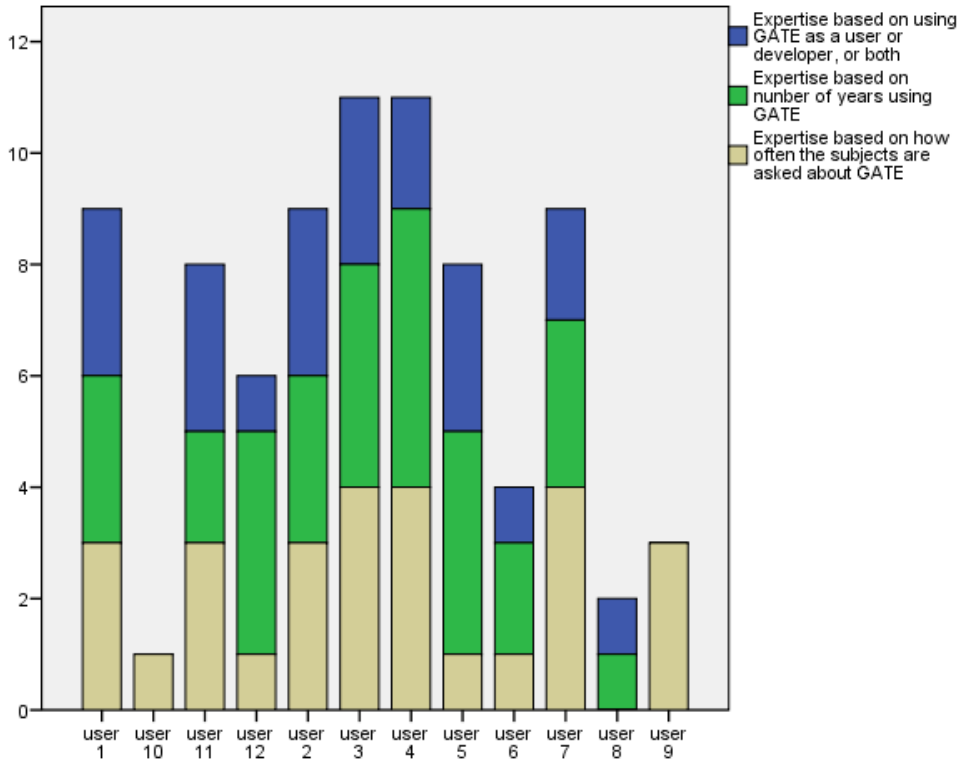

Figure 7.19: GATE Expertise expressed through the scalar value (0 – never used GATE, 11 – the most experienced GATE expert)

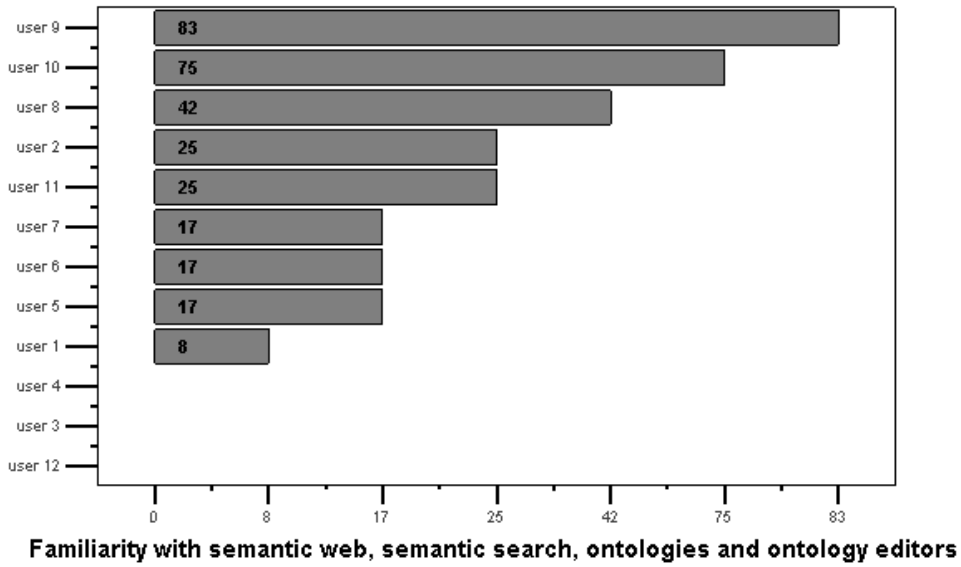

Figure 7.20: Expertise in semantic web technologies, familiarity expressed in the range from 0 (minimum) to 100 (maximum)

Efficiency

To measure efficiency, we measured the time it took each participant to finish each task. We used one-way repeated-measures ANOVA to test the significance of our results.

Figure 7.21 shows the *average time* spent per task for all participants. From this diagram it is visible that for most tasks, the subjects finished tasks considerably faster with QuestIO. On average, it took 46.61% longer to finish each task using baseline (traditional ways of search) in comparison to using QuestIO (107.1375 seconds vs. 157.075 seconds). The only exception is task 3 where participants spent slightly more time to finish it with the QuestIO prototype. When looking at the results more closely, we noticed that when performing task number 3 using the QuestIO prototype, the participants spent majority of the time clicking on the resulting URLs in order to decide whether the documents were relevant or not. When using a traditional search method, they could easily determine this because they were displayed not only with the URL, but also with short snippets from the content.

In order to test the statistical significance of this difference we used repeated-measures one-way ANOVA with two levels. This method assumes the normal distribution of the dependent variable (average time per task in our case) for all factors, hence we first performed the tests of normality for both groups using Shapiro-Wilk test with 95% confidence interval[19].

This test suggested that both groups do not have a normal distribution ($p < 0.001$ for both groups) and hence we performed the data transformation using Ln function. The Shapiro-Wilk test on the transformed data reveals that the normal distribution exists ($p = 0.353$ for baseline and $p = 0.987$ for QuestIO indicating that the latter data have almost perfect normal distribution). On the transformed data, we proceed with the one-way ANOVA with 2 levels to further investigate our findings. We set up our null hypothesis as follows: there is no difference in the efficiency of the baseline and QuestIO approach. If we find that this is true the assumption is that the difference in efficiency

[19] Another commonly-used test is Kolmogorov-Smirnov. Both methods test whether one distribution is significantly different from a normal distribution. Shapiro-Wilk test is recommended when the sample size is between 3 and 2000 and the Kolmogorov-Smirnov test if the sample size is greater than 2000. See Phillips [1996] for more details.

happened due to chance.

The ANOVA shows that we can reject the null hypothesis, and conclude with a high level of confidence ($F(1, 11) = 9.5, p = 0.001$) that the subjects were significantly slower when using baseline (157.08 seconds) in comparison to using QuestIO (107.14 seconds).

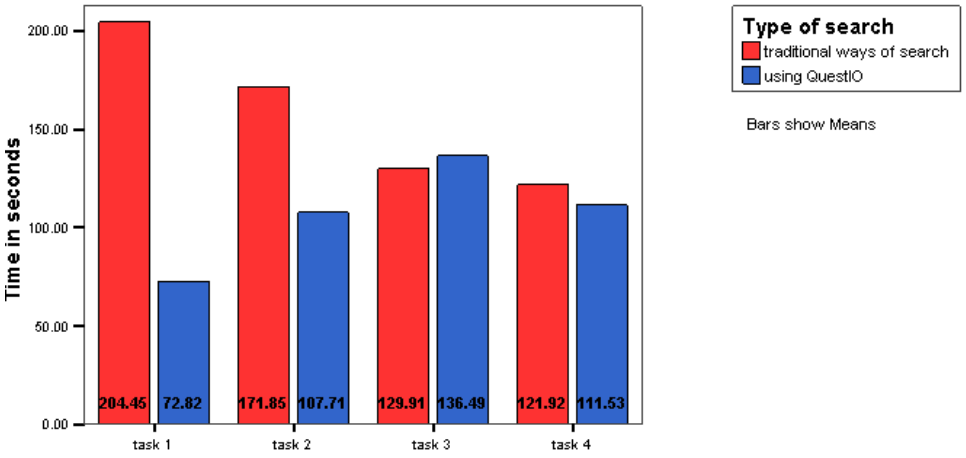

Figure 7.21: Average time per task

Effectiveness

Effectiveness indicates how successfully the tasks were finished using both systems. We observed each user and graded task success as:

- task completed with ease (0),
- completed with difficulty (1),
- failed to complete (2).

Figure 7.22 shows the difficulty per task based on the success rate for all participants. Two extreme cases were task 1, which was finished successfully with ease by all participants when using QuestIO, and task 2 which was not completed by any of the participants, when using alternative ways of search. This is in line with our expectations. Overall, the success rate for performing tasks using QuestIO was 0.355 in comparison to 0.895 using baseline, on the

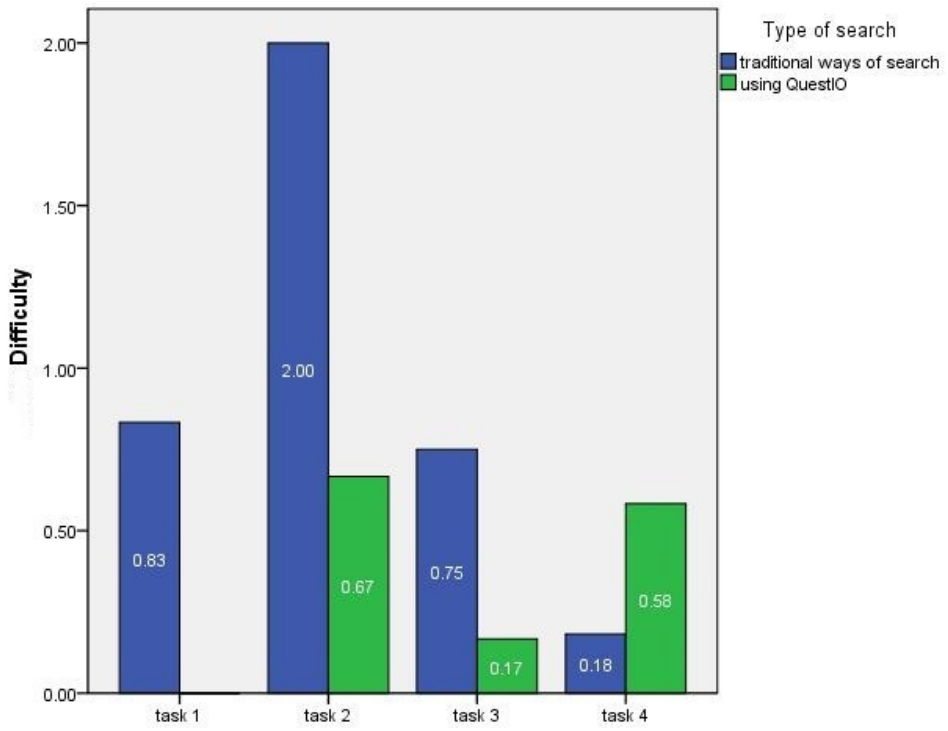

Figure 7.22: Task difficulty based on the average success rate per task: task completed with ease (0), completed with difficulty (1), failed to complete (2)

scale from 0 to 2. This difference is highly significant according to the Friedman test with $\chi^2 = 12$ and $p = 0.001$[20]. This indicates that subjects found it easier to finish tasks using QuestIO, in comparison to the baseline.

User satisfaction

We chose the SUS questionnaire as our principal measure of software usability because it is the *de facto* standard. This questionnaire is developed according to the proper techniques based on the Likert scale [Brooke, 1996].

Furthermore, researchers at Fidelity Investments carried out a comparative study of SUS, three other published usability questionnaires and an internal questionnaire used at Fidelity, over a population of 123 subjects, to deter-

[20]The Friedman test is the non-parametric alternative to the one-way ANOVA with repeated measures. It is used to test for differences between groups when the dependent variable being measured (success rate in our case) is ordinal.

mine the sample sizes required to obtain consistent, accurate results. They found that SUS produced the most reliable results across all sample sizes; they noted a jump in accuracy to 75% at a sample size of 8, but recommended a sample of at least 12–14 subjects [Tullis and Stetson, 2004]. Consequently, for our evaluation, we recruited 12 participants.

As a reference for interpreting the results, SUS scores range from 0 (very little satisfaction) to 100 (very high satisfaction) [Bailey, 2006], and scores from 60 to 70 are considered average. Total mean SUS score in our evaluation was 69.38, which is almost equal to the most common value (the mode) and also median which both were 70 (see Figure 7.23). SUS scores ranged from the minimum of 52.5 to the maximum of 85. Moreover, as our SUS scores were almost perfectly normally distributed (Shapiro-Wilk coefficient=0.974, p=0.949), we can estimate that 95% of the SUS scores will fall in the range from 63.47 to 75.28. This is a satisfactory result.

Spearman test showed that the SUS result was not influenced neither by the GATE expertise of our subjects (Spearman correlation coefficient = -0.134, p-value=0.679, for the relation between GATE expertise and the SUS score), neither by their knowledge of semantic web technologies (Spearman correlation coefficient= 0.014 for the relation between semantic web knowledge and the SUS score, p-value=0.965).

Subjective measures of user satisfaction

Whereas above results are based on our judgment from observing the subjects, we were also interested into a subjective insight as perceived by the subjects themselves. To assess that we asked them five questions, first of which was whether *it was easier to perform tasks using the QuestIO prototype in comparison to the alternative ways of search.* The overall results were in favour of our prototype, with 18.8% disagreement to this statement, 54.1% agreement, and 27.1% neutral. Figure 7.24 shows the detailed distribution of results, grouped by task. For tasks 3, 2 and 1 (sorted by % of agreement) the subjects considered QuestIO easier in comparison to the traditional ways of search, whereas for task 4, they voted in favour of the latter.

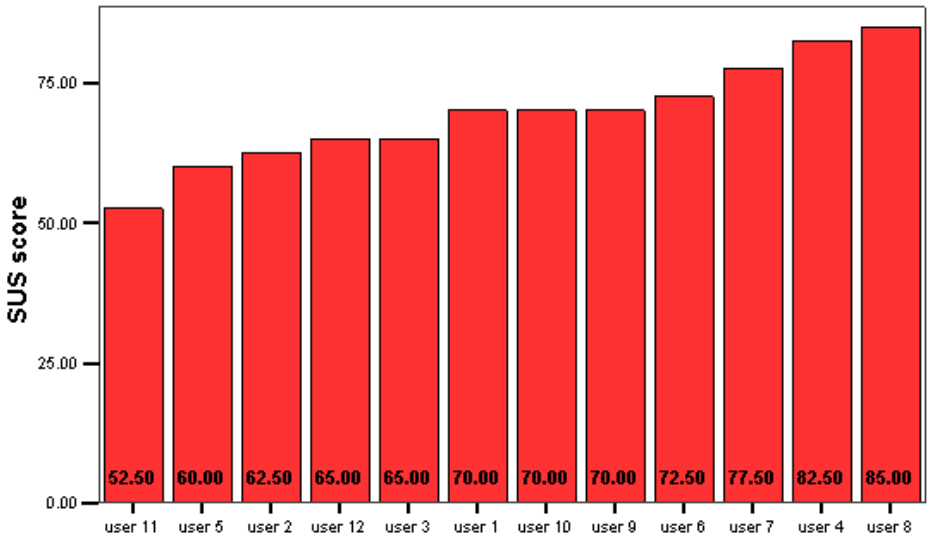

Figure 7.23: The SUS score by participant

Users also reported that:

> *The prototype was slightly easier, in that it listed them all (except rasp, which it missed) on a single page with no false positives.* (user 2, task 1)

but also

> *I couldn't find the answer using the standard approaches.* (user 1, task 2)

In order to get a subjective measure of result relevance, we asked subjects after each task *whether they found the results returned by QuestIO relevant.* Across all tasks, 60.4% of subjects agreed, 10.5% disagreed, and 29.2% were neutral. Detailed distribution is shown in Figure 7.25. It is interesting to note that while the first three tasks had no recorded disagreements of subjects, the task 4 seemed problematic and 41.7% subjects disagreed to the statement that the results returned by QuestIO were relevant. Another extreme is Task 2 which has the highest level of agreement (83.3%) in comparison to the other tasks. Let us recall that Task 2 was the most difficult

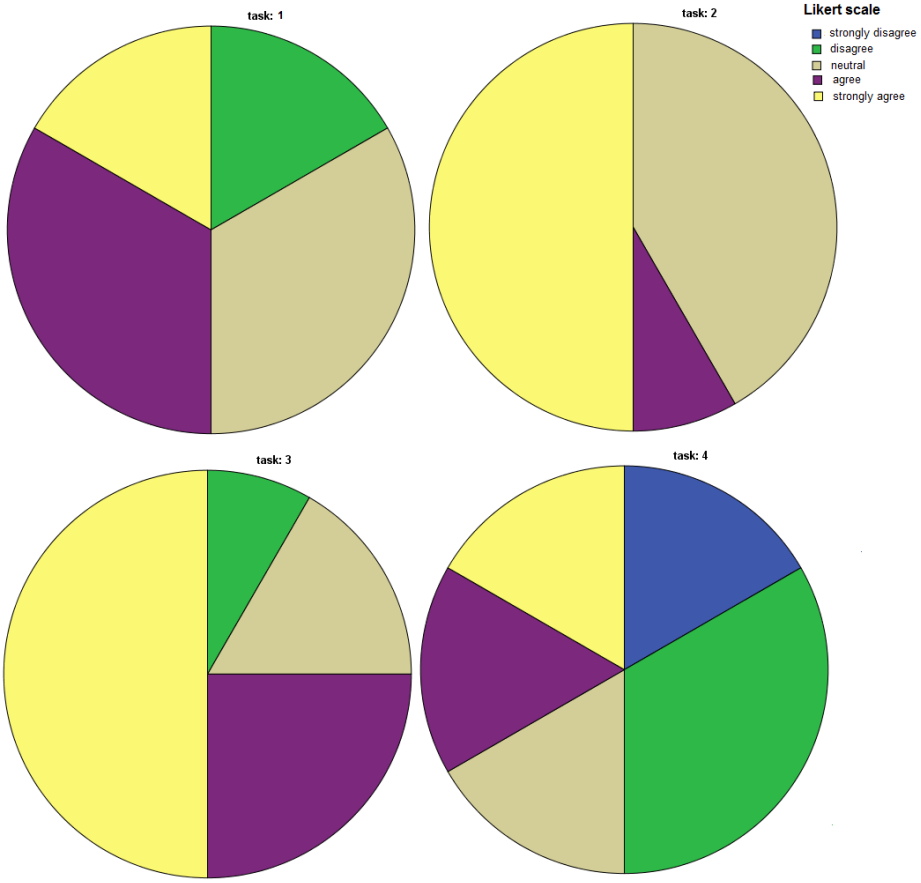

Figure 7.24: Agreement of subjects to the statement that *It was easier to finish the task using the prototype in comparison to the traditional ways of search.*

task for all participants as shown in Figure 7.22. A closer analysis revealed that the queries for Task 4 were very similar to those they would have typed into Google, some of which were not related to GATE components at all, although they were related to GATE. For example, queries such as *GATE web site*, or *GATE projects*. This demonstrates that we failed to explain well to the subjects what kind of knowledge is available, and also that the prototype is not a replacement of a general purpose search engine, firing off queries against the entire Web and its documents, but rather containing the knowledge about GATE as a software.

In addition to choosing an answer from the Likert scale, the participants

could give specific comments for each question. These comments reveal that different users had different experience using QuestIO prototype. For example, a user reported that:

> *The QuestIO results do not offer any summary or snippet, which makes it difficult to assess their relevance.* (user 12)

while another user reported:

> *It's useful to have the various sources all available to search via a single interface.* (user 4)

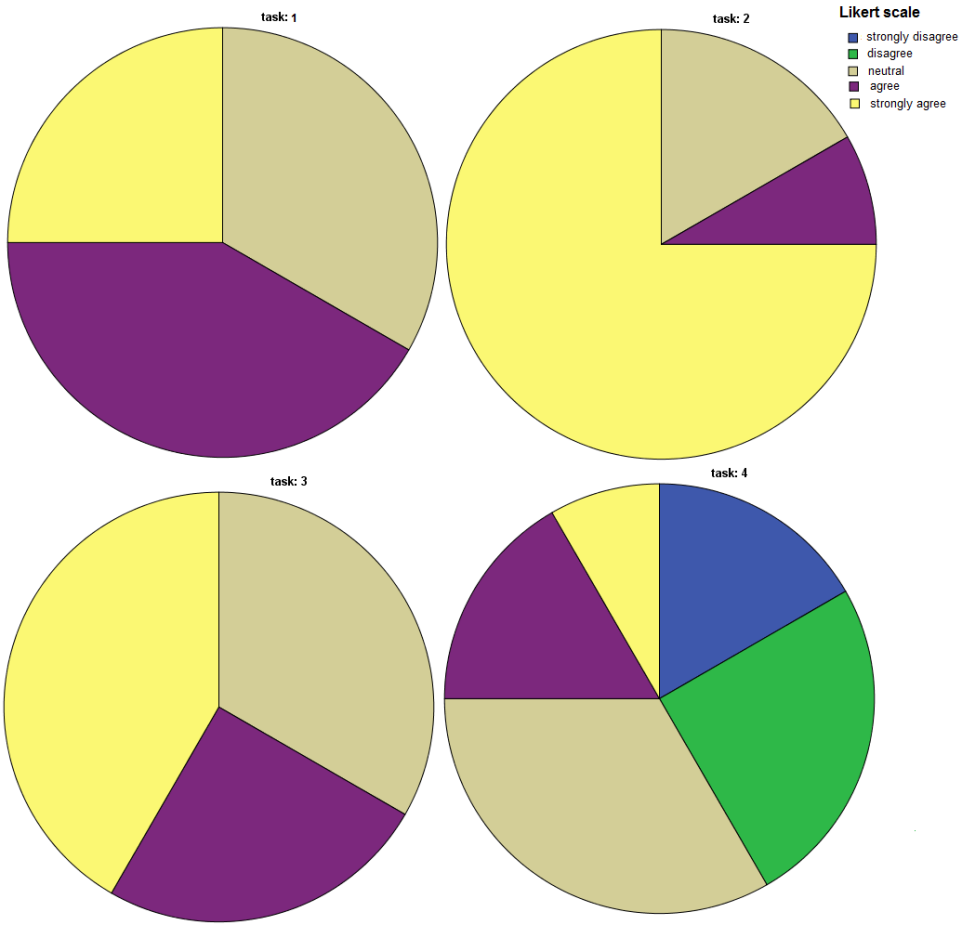

Figure 7.25: Relevance of results returned by QuestIO

We asked all participants *whether the possibility to browse the ontology in the prototype was helpful*. 8.3% disagreed, 54.2% agreed, and 37.5% were neutral reporting that they did not need to use it. Figure 7.26 shows the detailed distribution of results. For tasks 2, 3 and 1 (in decreasing order according to agreement) this option was rated most helpful, while again, task 4 had the highest percent of disagreement in comparison to others (25% in comparison to 0% for Tasks 2 and 3, and 8.3% for Task 1).

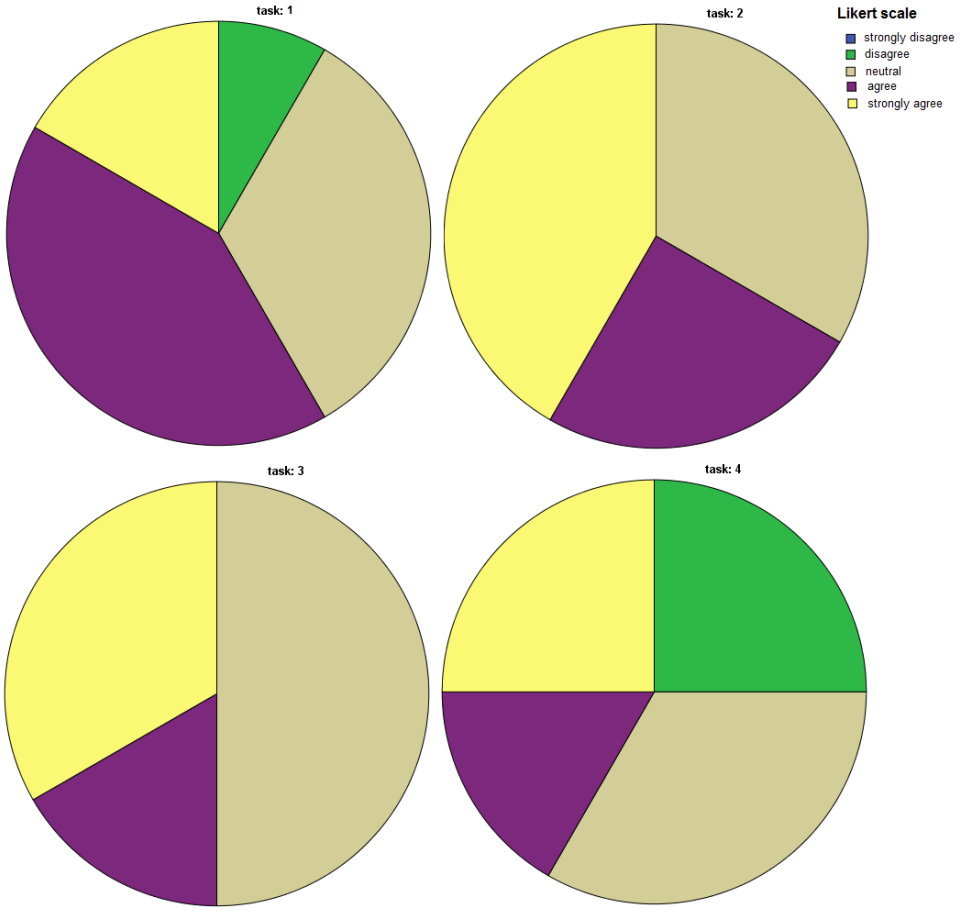

Figure 7.26: Was the option to browse the ontology helpful?

In the participants's own words, when searching for *developer of Morphological Analyser* the system could not find any results, so the user browsed the ontology and afterwards reported:

It explained why I couldn't search for Niraj and Morpher to-

> *gether because there wasn't a relationship defined there.* (user 12, task 4)

We asked all participants *whether the refinement pane was helpful.* The refinement pane showed the answers for Tasks 1 and 2 (given that participants formulated the query which could be parsed successfully), while for the Task 3 it was used for the refinement of the results, as the task was to find documents about specific GATE component (Learning PR). Overall, 58.4% agreed, 6.3% disagreed, 35.4% reported that they neither agreed nor disagreed.

Figure 7.27 shows that for tasks 1, 2 and 4 the disagreement was equal (8.3%) while for task 3, all subjects eather agreed that the refinement pane was helpful or were neutral. Interestingly, agreement for the first 3 tasks was similar (75%, 58.3% and 66.7% for tasks 1, 2 and 3 respectively), while at the same time much higher than the agreement for the task 4 (33.3%).

However, most of the comments expressed positive attitude overall:

> *It seemed to narrow down the results to the right thing* (user 1, task 3).

or

> *Great! the refinement pane suggested 'LearningBatchLearningPR', which was the right choice* (user 12, task 3).

We asked all participants *whether it was easy to formulate queries using QuestIO.* 68.7% agreed, 4.2% disagreed, 27.1% neither agreed nor disagreed. Figure 7.28 shows the detailed distribution of results, where it is visible that the easiest was to formulate the query for task number 3 (100% agreement), which is in-line with one of the comments:

> *same keyword used with the mailing list - and it works!*, (user 7, task 3)

The next easiest was Task 1 (83.3% agreement), followed by Task 2 (50% agreement), and finally Task 4 (41.7% agreement) with which subjects struggled as they report:

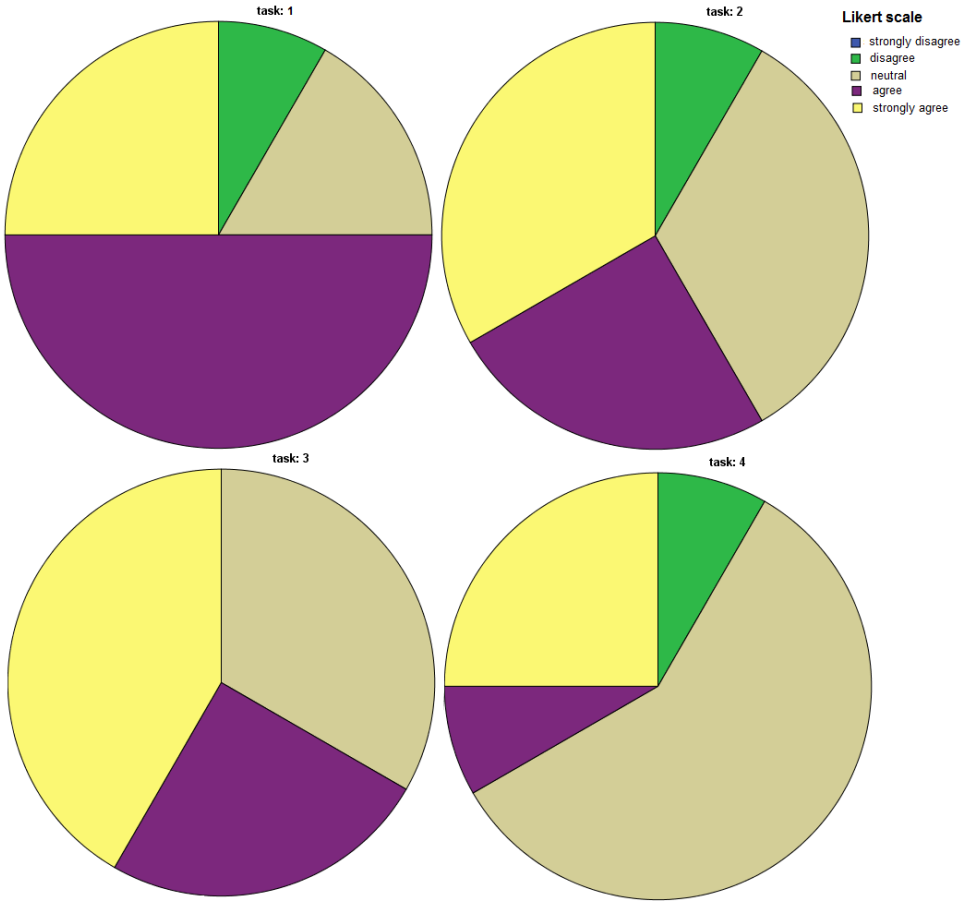

Figure 7.27: Was the refinement pane helpful?

> *Wanted to search for PRs implemented by Niraj. Couldn't do it or didn't know how to do it*, (user 7, task 4) .

We have also asked the participants for any suggestions and ideas on the interface improvements. Some of the suggestions were as follows:

- "*When displaying the results (document links) you need to provide a summary or a lead paragraph or a list of keywords so that the user knows whether a document is relevant*".
- "*The answer was in the 'refine' section at the bottom of the page; maybe put this closer to the top?*"

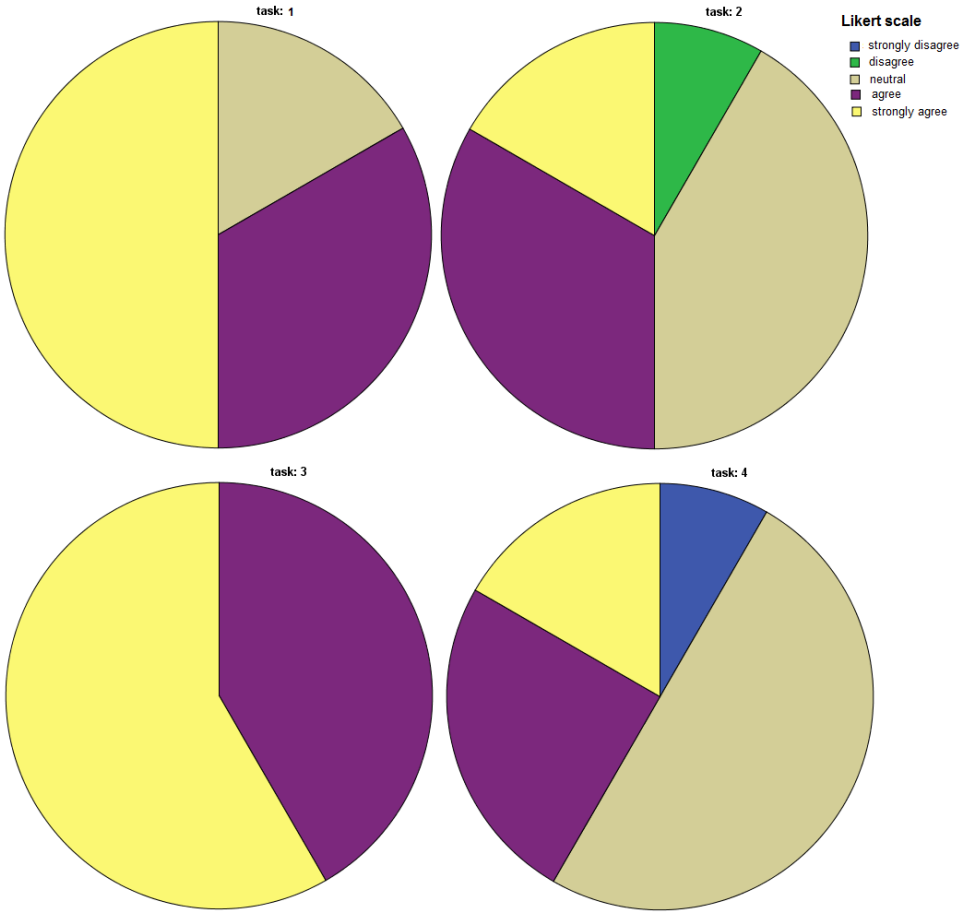

Figure 7.28: Was it easy to formulate the query using QuestIO?

- "*Yes, definitely don't display everything (papers, pages, posts) but only resources that are specified.*"
- "*Also navigation, ordering, ranking and grouping of results should be implemented. Highlighting the terms is not supported, too, which I find very useful.*"

The evaluation results show that majority of participants (68.7%) found the QuestIO language easy, and in fact 54.1% found using QuestIO easier than traditional ways of search. The reason for this might be the possibility to find all information at one place, rather than searching several locations on the Web. Through measuring the relevance of the results (60.4%) we proved that

the SPARQL queries generated based on the user's queries were sufficiently accurate. Despite the fact that 58.4% subjects agreed that the refinement pane was helpful, the comments they left indicate that our design decision to use refinement pane both for showing the results of the query (Tasks 1 and 2), and refining the documents to contain results of the query (Task 3) was confusing. Although our subjects were not highly experienced with semantic web technologies, the possibility to browse ontology was rated as helpful in 54.2% of cases. While the five tested features of QuestIO received quite a positive feedback in general, we must note that, without exception, Task 4 received the most negative results. Many of the comments left by our subjects indicate that it was difficult for them to understand the scope of the knowledge available in the prototype, and also, in cases of failures, to understand why they happened. Similar trend is revealed through the effectivness evaluation, where subjects were generally more successfull for all tasks on average, the only exception being task 4.

In addition, many subjects were frustrated by the fact that they had to browse through documents in order to judge their relevance and also by other user interface issues listed above, most of which were related to the subjects' familiarity with search engines and the way these show results. The prototype was taken as a competitor of Google, which is not what our intention was, however, due to the comparison and also the fact that we asked the subjects to perform identical tasks using the prototype and any other page on the Internet, lead them to such thinking. Nevertheless, this evaluation encouraged us to continue work on the text-based interface, however, we have revisited our requirements based on the analysis just shown, before we did any further work.

7.6 Summary and Discussion

In this chapter we described a Question-based Interface to Ontologies which automatically derives the answer by translating the Natural Language or keyword-based question into the SeRQL/SPARQL, and returns the answer to the user after executing the formal query against a given ontology. We evaluated QuestIO in several aspects, and here we present the brief summary

and the lessons learned from those evaluations which assessed the main features of QuestIO which are the *flexibility of the supported language* and *the portability without customisation*. We also presented the GATE case study and the user-centric evaluation which assessed the *usability* of QuestIO, the difficulty of the supported language, and several other features.

Flexibility of the supported language We have evaluated *expressiveness* of the supported language using the software engineering domain with 36 questions from the GATE mailing list, and then repeated the experiment with AquaLog. QuestIO's query language showed advantages in comparison to the AquaLog's, thus resulting in higher precision and recall for the same set of questions. The main difference in the two supported languages is that while QuestIO does more shallow language processing in comparison to AquaLog, it supports not only grammatically correct questions, but also question fragments, and ill-formed ones.

The language supported by QuestIO proved to be very *robust* during the user-centric evaluation, based on the number of different queries which have been formulated in order to successfully finish the same tasks. To give an example, in order to complete Task 2, users typed in:

- "cebuano gazetter parameters"
- "What are the runtime parameters of cebuano gazetteer?"
- "what are the parameters of cebuano gazetteer?"
- "Cebuano gazetteer runtime parameters"
- "Runtime parameters of cebuano gazetteer"
- "Cebuano runtime parameters"
- "Cebuano gazeteer": this example includes the spelling error, due to which the system failed to return the correct answer; it has happened twice for two different users, where one of them immediately typed the same query into Google and based on the 'Did you mean' functionality which offered the correct spelling, figured out that he made a mistake – corrected it and got correct results, while the other one gave up on

the query and went to browse the ontology. Nevertheless, he has also found the correct answer.

Moreover, our findings showed that encouraging subjects to use keyword-based input was in places misleading. Set of keywords for search engines is a bag of words which would be searched against the available documents using boolean operators. Set of keywords as an input for a Natural Language Interface is a set of words with omitted prepositions or WH-phrases for example – these keywords are not independent, the assumption is that the user is interested in existing *relations* between them. These two concepts are quite different, and it might be better way to encourage users to use *Natural Language* questions, even if not fully grammatically correct and with omitted words, instead of encouraging them to use *keyword-based* queries.

However, the flexibility of the supported language has a trade-off which was highlighted during the user-centric evaluation. Namely, the concepts which appear in the query should be in line with the **knowledge structure**. An interesting example of the system's failure was the query *Cebuano parameters*. System recognized *Cebuano* as a plugin, and also *parameters*, but it could not connect these two as there is no direct relation in the ontology. The system also failed to provide a useful feedback so the user decided to browse the ontology in order to figure out what the problem was and find the answer. As shown in Figure 7.29, the system needed to find a *Processing Resource* which is the *connecting* node in between the two nodes which appeared in the query; this emphasises the fact that **translation from the NL query to the SPARQL query which will give the correct answer must take the structure of the knowledge into account – and this is what makes the design on NLIs to any structured data extremely hard and expensive**. The system could search for the *connecting* nodes in between concepts in the query (and not only the direct relations), but this is very time consuming and in addition will generate a lots of noise which might be very difficult to filter. Moreover, there might be more than one connecting node in between the two concepts, and this complicates the issue even more.

Scalability and portability We have assessed QuestIO's scalability by trialling it with the two ontologies of different sizes one being the subset of

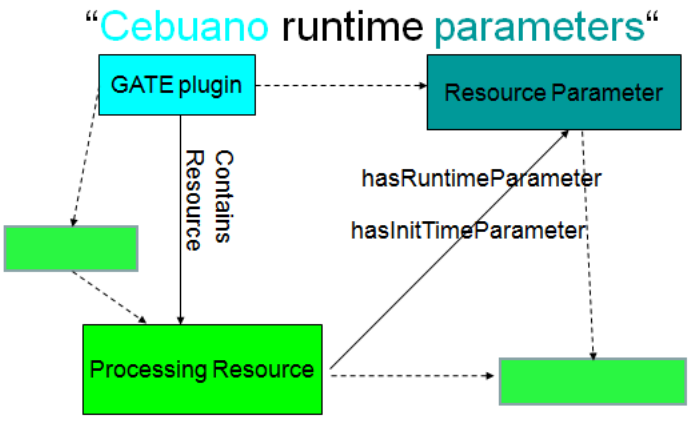

Figure 7.29: Finding parameters of the Cebuano gazetteer: the importance of the data structure

the other. While the quality of results was not affected, the initialisation time and the execution time were significantly longer for the larger ontology. More specifically:

- The ranking mechanism which relies on the properties and string similarity caused extremely low performance (long execution time) for some queries.
- Best ranking based on the assumptions for one type of ontology design might not stand for the other, and therefore automatically returning the answer in all cases might not be preferable.
- Using SPARQL as an alternative to SeRQL is faster for version 2.8.4 of OWLIM which relies on Sesame 1.x.

Usability With regard to the user-centric evaluation, QuestIO had quite a good performance when evaluated on the set of predefined questions and also with questions for which the answer was in the ontology. However, for undefined tasks (where subjects had freedom to type in any query they were interested in) users often were not satisfied with the results, and the reasons can be summarized through the following cases:

Lexical failures *Tokenizer* vs. *Tokeniser*: the system did not support spelling variations; but neither it could recognize the misspelled words

such as *Gazetteer* vs. *Gazeteer*; also, *horacio saggion articles* failed due to the system not being able to find relation between *publications* and *articles*. While *publication* was in the lexicon, *articles* was not and thus the system returned no answer.

Conceptual failures The most common types were:

- **Missing concepts**: the system failed to return any results due to no knowledge about concepts, for example *Projects about GATE*, *GATE web site*; although the ontology is about GATE components, even *GATE* was not recognized by the system as such a concept does not exist in the ontology (however, GATE plugins, GATE PRs, etc. do exist); similar was for *JAPE*, which was not recognized by the prototype and resulted in some very experienced GATE users be very frustrated due to the fact that the system about GATE *must* know about JAPE.
- **Missing relations between concepts**: for example, for the query *Developer of Tokeniser* the system knows about *Developers* such as Adam Funk, Niraj Aswani, Hamish Cunningham, etc. however, there was no relation defined between developers and the components they developed. Therefore the system failed to find them but also to give a useful message to the user.
- **Missing concepts and relations**: this case is the combination of the previous two, such as in the query *Author of morphological analyser* where the system did not know about *Author*; the user reformulated the query into *Developer of morphological analyser* but still no results were found this time due to no relation between *Developer* and *morphological analyser*.

While lexical failures are easier to address (e.g. using some spelling algorithm in combination with WordNet) than conceptual failures, both remain challenging. For example, in the last mentioned example, the system does have knowledge about question terms in the ontology, but it cannot find relations between them and therefore does not give any answer. In an ideal case, the system should communicate the relevant message to the user indicating whether the answer does not exist, or the query needs to be reformulated.

With regard to the lexical failures, using WordNet would help find the *articles* and *publications* as synonyms for example, however, for very specific domains such as the GATE case study, it would fail to connect *author* with *developer*.

To address these challenges we developed a second NLI system called FREyA, to which we now turn.

Chapter 8

Towards Better Usability with FREyA: Part I

In this chapter[1] we present the FREyA system which is named after **F**eedback, **R**efinement and **E**xtended Vocabular**y** **A**ggregation. Our intention with FREyA is to combine these usability enhancement methods (discussed in Chapter 6) in order to improve the performance of NLIs to ontologies and address the challenges discussed in Section 7.6:

Conceptual failures: At first, we have taken QuestIO one level up by including the user into the loop when interpreting questions. We experiment with *feedback* by showing the user all *query interpretations* and the system's rankings, so that the user can influence the answer by choosing the correct interpretation. This approach is evaluated with users, details are presented in this chapter. Further on, we move towards *concept interpretation* where we use *clarification dialogs* for resolving *ambiguities* through *query refinement* – details of this work are presented in Chapter 9.

Lexical failures: we have used the user-interaction methods to extend the system's lexicon from the *user's vocabulary*. In addition, we experiment with handling the spelling-errors and synonym detection.

[1] An updated version of of this chapter is submitted as a research article to the Journal of Web Semantics.

Addressing lexical failures is detailed in Chapter 9, where in addition, we address the following:

- **Deeper grammar analysis** The identification of the the question's semantic meaning is improved (in comparison to QuestIO) by combining the syntactic analysis with the ontology-based lookup.
- **What to show** The goal is to provide a concise answer to the user's question. The result of the SPARQL query returned by QuestIO is a subgraph (represented by a table) which ideally contains the answer to the user's question. However, it is not trivial to derive one single answer from this subgraph, and showing the whole graph to the user might be overwhelming (as pointed out by users in the evaluation described in Section 7.5).

8.1 Feedback

As we discussed in Chapter 6, *feedback* increases the user's confidence and in the case of failures, helps the user understand which habitability domain is affected (see details about habitability domains in Section 4.1). In our initial design of FREyA, we modelled *the system's interpretation of the query* based on two important aspects:

- *The answer is found*: feedback can make the user more confident that the answer is indeed correct and also it can make the user familiarise himself with the queried knowledge structure.
- *The answer is **not** found*: feedback should be used to make the user aware of the reasons for no answer. This is more complex case as it is sometimes hard to identify which habitability domain is affected:
  - The answer is not found because the system could not parse the question (the lexical or syntactic failure, the question *could* be answered if reformulated).
  - The answer is not found because the system could not find the information about the required concepts (the conceptual failure, the question *could not* be answered if reformulated).

– The answer was not found although the system successfully parsed the question. This case is probably the most complex because it might mean that the answer is negative, but also that the information is missing.

8.1.1 Hiding Complexities

One challenge when modelling feedback is showing the system's interpretation having in mind that NLIs are intended to be used by the users not necessarily familiar with ontologies. NLIs to ontologies usually translate a natural language query into some intermediate interpretation which is a set of triples, which is then translated into a formal query such as SPARQL. Hence, the most natural way, from the point of view of the system's developer, would be showing either triples or the SPARQL query. However, as our intention is to develop methods which are suitable for casual users as well as for semantic web experts, in our initial design we want to simplify the system's interpretation, and hide complexities as much as possible. Therefore, we take the following decisions:

- show labels instead of URIs;
- show the linear list of elements (instead of triples) in order in which they appear in the question;
- show relations between the elements by rendering a tree-like structure.

8.1.2 Identified Context and Tree-based View

Implementing these decisions resulted in the Web interface which looks as in Figure 8.1. After the user posts a question, the system first generates the table with two columns: *Identified context* which shows the linear list of elements (recognised concepts and relations between them as found in the ontology), and *Our score*, which shows the score based on which the *Identified contexts* are ranked. The system automatically selects the first option, and the results are rendered using the *tree-based view*.

The user has the option of selecting any of the *Identified contexts* by clicking on the radio button in the desired row. The results will be rendered on click.

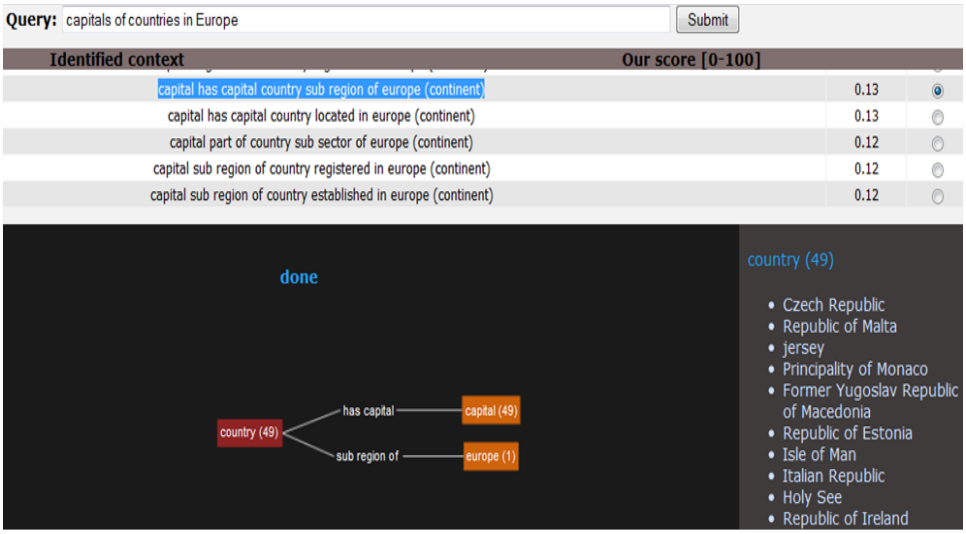

Figure 8.1: The FREyA interface

Further on, the user can explore the tree-based view by selecting its nodes, for example *Country* in Figure 8.1, and the instances will be shown in the right pane.

In the case when the system recognises concepts in a query, but does not find any results, the concepts will be shown in the *Identified context*, and on selection the message reading *No relation found within this context* is displayed in the area for the tree-based view (see Figure 8.2).

8.1.3 Linearised List of Concepts

We have mentioned previously that the order of the recognised concepts follows the order in which they appear in the question. This is to ensure the user is not confused with the output, and also to try and 'translate' the natural language question into the set of recognised concepts. However, due to the presence of properties in each query interpretation (as properties are crucial to get the correct answer), this can lead to the 'not so natural' effect, see for example Figure 8.3. The *Identified context* is shown including *has run time parameter* relation. The interpretation as such is not understandable without additional explanation to the user – users must be trained to understand the role of the property in between the recognised concepts. The

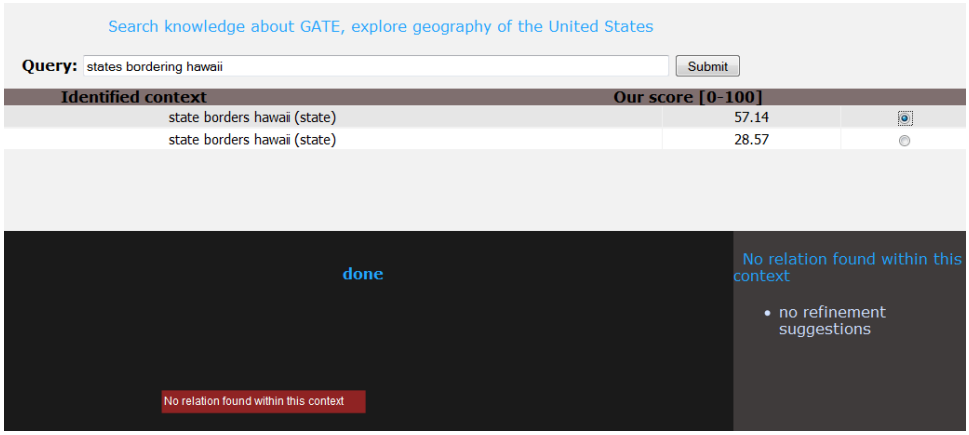

Figure 8.2: The FREyA interface: showing results for *states bordering Hawaii*

other option which we could consider is to reverse the order and show the interpretation to read:

```
rasp parser (language analyzer) has run time parameters resource parameter
```

However, for more complex queries, this approach would require modelling triples. For example, if we look back at the example in Figure 8.1, the first interpretation reads:

```
capital has capital country sub region of europe (continent)
```

To make this interpretation more natural, we would have to show:

```
country has capital capital;
country sub region of europe (continent)
```

However, this makes it harder to follow which question term refers to which ontology concept, and from where the relations were derived. Therefore, we stay with the linearised representation, but decide to model the tree-like view (see the lower left part of the Figure 8.3), so that it is indeed clear to the user that according to the knowledge structure 'the rasp parser has run time parameters...' and not the other way around.

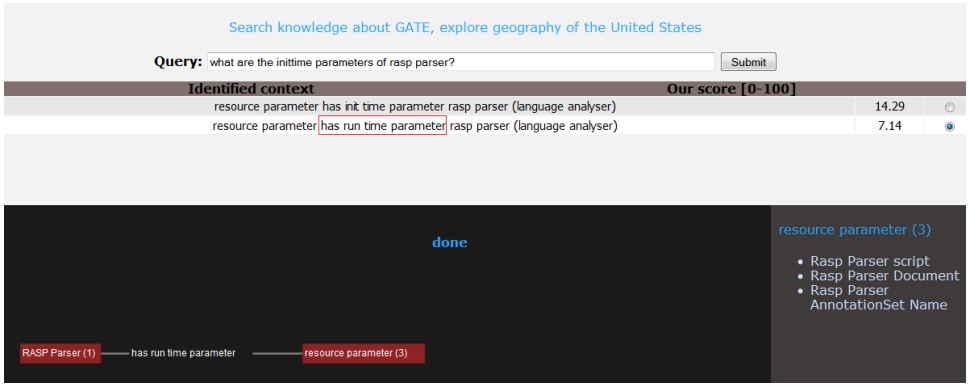

Figure 8.3: The FREyA interface: showing results for *runtime parameters of rasp parser*

When no results are found, the user will see the message *No relations found within this context*, see Figure 8.4.

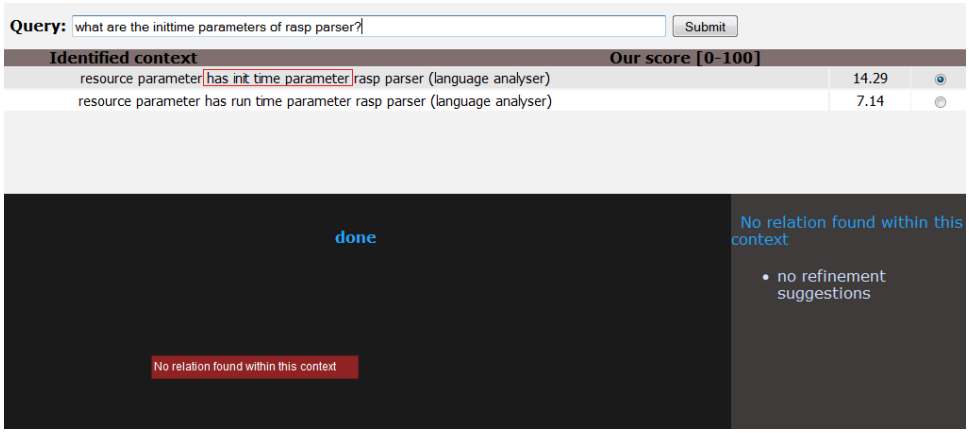

Figure 8.4: The FREyA interface: showing results for *init parameters of rasp parser*

The tree-like structure shows relations between the concepts which are *grouped* so that for example, one node is rendered for each class, but not for all instances. Instances are shown in the right pane when the user clicks on the node.

8.2 Evaluation

During the GATE Summer School in July 2009, we organised a task-based evaluation with the participants in order to test feedback. The evaluation was a part of the lecture on using GATE to build Natural Language Interfaces to Ontologies.

8.2.1 Evaluation Scope

Our intention was to see whether users could make the correct conclusions based on the system's feedback, and therefore:

- reformulate the query in order to get better results,
- terminate the task based on the conclusion that
  - there is no answer (answer is negative), or
  - the knowledge was not available in the system.

In addition, we wanted to assess the efficiency, effectiveness and user satisfaction of FREyA in comparison to QuestIO.

8.2.2 Experimental Setup

Training

The participants listened the 20 minutes talk about Natural Language Interface to Ontologies, where they were given a short overview of how the system works and they had also the chance to familiarise themselves with the separate components in GATE in order to understand various Processing Resources which are used for building QuestIO and FREyA. They were given a five minutes demo on how to use the Web-based interface[2].

[2]Slides are available from `http://gate.ac.uk/sale/talks/gate-course-july09/slides-pdf/questio.pdf`

What to measure?

At the beginning of the experiment, we asked participants to complete the questionnaire about their background (age, profession, knowledge of semantic technologies). Next, they are asked to complete four tasks, and after each task, they had to answer several questions:

- whether they could finish the tasks successfully: based on their answer, we measured *effectiveness*;
- whether the feedback was helpful or not;
- whether it was easy to formulate the queries for the task or not.

As a set of answers the subjects were offered a pre-defined set, with an option to add additional comments in free-text. The full task list and the questionnaires given to the participants can be found in Appendix B.

After finishing all tasks, subjects were asked to complete the SUS questionnaire as a standard user satisfaction measure.

In addition, we measured the time each user spent on each task, and also the number of queries they have used.

8.2.3 Dataset

We have initialised FREyA with two domain ontologies. The first one is the same as in the evaluation described in Section 7.5 covering GATE components, while the second one is the Mooney GeoQuery ontology covering the United States geography[3].

8.2.4 Tasks

As previously mentioned, subjects were asked to perform four tasks. For each task they had the opportunity to read the one related to the GATE domain, and the other one related to the United States geography, before they decide

[3]The Mooney geography dataset is available from `http://www.ifi.uzh.ch/ddis/research/talking-to-the-semantic-web/owl-test-data/`

which one to perform. If they were not confident in their knowledge about GATE, we hoped they would choose the task related to the US geography. The task pairs covering two domains were of the same complexity.

We will next list the tasks and describe what were our incentives for them.

Task 1:

- Task 1a: Find part of speech taggers which exist in GATE. Find out which parameters exist for the POS Tagger of your choice.
- Task 1b: Find mountains which exist in the United States. Find out in which state is the mountain of your choice located.

Our assumptions: This task contains two parts, each of which was a separate task in the previous evaluation of QuestIO. The intention is to compare efficiency and effectiveness of subjects with those in the previous study. This way we are testing the effect of the new usability features which exist in FREyA, but were not part of QuestIO.

Task 2:

- Task 2a: Imagine that you are a GATE developer who needs to extend the RASP Parser. Your task is to find out the names of init parameters.
- Task 2b: Find out which states border hawaii.

Our assumptions: In this task, our goal was to see if the system provided enough feedback to the user when the answer to the question was negative. Both ontologies contain the knowledge about the concepts in question, however, the lack of relations between the concepts indicates that there is no answer. What we are interested in here is whether the system's interpretation of the query together with the message *No relations found within the context* can be clear to the user, and if he can make the decision with confidence based on this.

Task 3:

- Task 3a: What are the parameters of the PRs which are included in the same plugin as the Morpher?

- Task 3b: Which rivers flow through the state in which the mountain harvard is located?

Our assumptions: With this complex task we wanted to see whether the users are able to complete it based on the feedback the system provides. The task is complex in terms that it requires formulation of at least two queries to get the answer. The second query needs to be formulated based on the answer and the feedback returned for the first query. The subjects need to figure out that there is knowledge about what they are searching for in the system, but the query they are likely to type in first is too complex and therefore, they need to reformulate it (and not give up concluding there is no answer).

Task 4:

- Try exploring the knowledge available in the system. Either search for various components of GATE such as PRs, plugins, LRs, VRs, or explore the United States geography by inquiring about: cities, states, rivers, mountains, highways, etc. Then ask some questions in order to connect these concepts such as 'which states border georgia?' or 'which rivers flow through states which border california'. Input as many queries as you like.

Our assumptions: with this task we test whether participants have understood what types of tasks can be performed using FREyA. Alternatively, we could compare the results of this task with the ones from the previous study. In addition, this gives us the opportunity to collect a new set of questions which can be used to test and extend FREyA's capabilities.

8.2.5 Participants

Participants were all outside Sheffield University, and were not known to us before they registered to attend the GATE Summer School. They were almost evenly distributed across researchers, software developers and students, and also across gender.

We measured their expertise in ontologies, ontology editors and SPARQL, using the Likert scale, and the expertise is calculated as a linear combination

of these three, and then normalised on the scale from 0 to 100 (similar to how the SUS score is calculated). With the most common value (the mode) of 50, median=58.33, and the range from 0 to 100, we conclude that their knowledge of the semantic web technologies was neither basic, nor advanced (see Figure 8.5), although with a mean of 60.8 and a very high variation (22.98) which is more distributed towards the higher values, it is leaning more towards the advanced level.

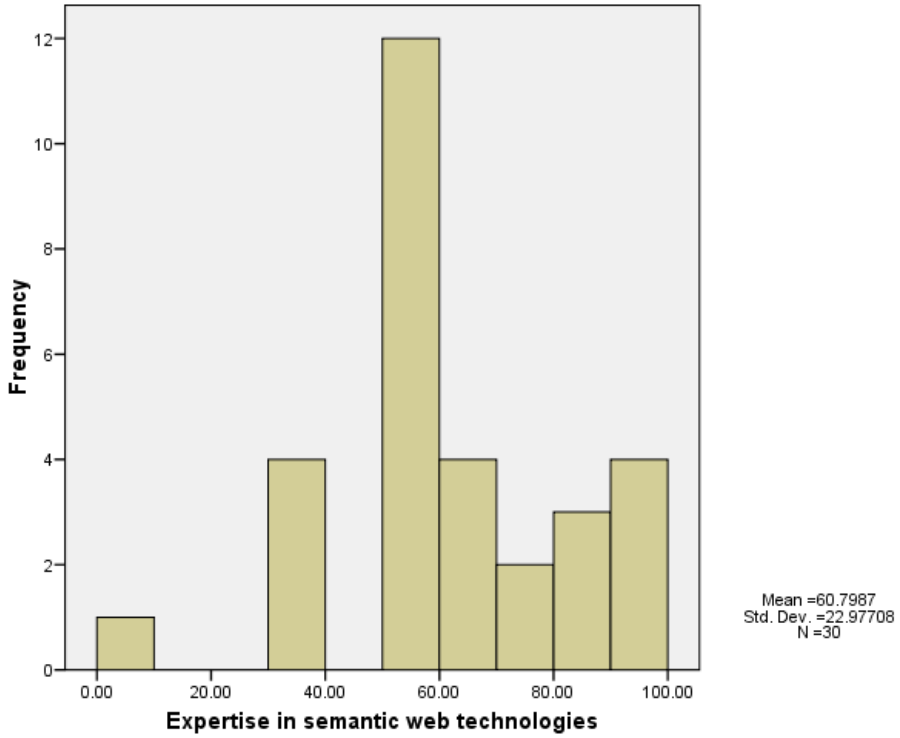

Figure 8.5: The expertise of subjects in using ontologies, ontology editors and SPARQL

8.2.6 Results

While the number of participants at the GATE Summer School was 50, the participation in the evaluation was on the voluntary basis, and many have not completed all required tasks and also all questionnaires. Therefore, we have disregarded all incomplete records, which resulted in the recorded data

for 30 participants having completed the background questionnaire and at least first three tasks. 11 out of these 30 participants finished Task 4, while 19 completed the SUS questionnaire. However, all of them have previously finished at least three tasks and therefore we can make conclusions about the user satisfaction based on these records.

Effectiveness

Figure 8.6 illustrates the task difficulty per task, based on the average value of the success rate across all participants. Task 1 was the easiest, while Task 3 was the most difficult to finish.

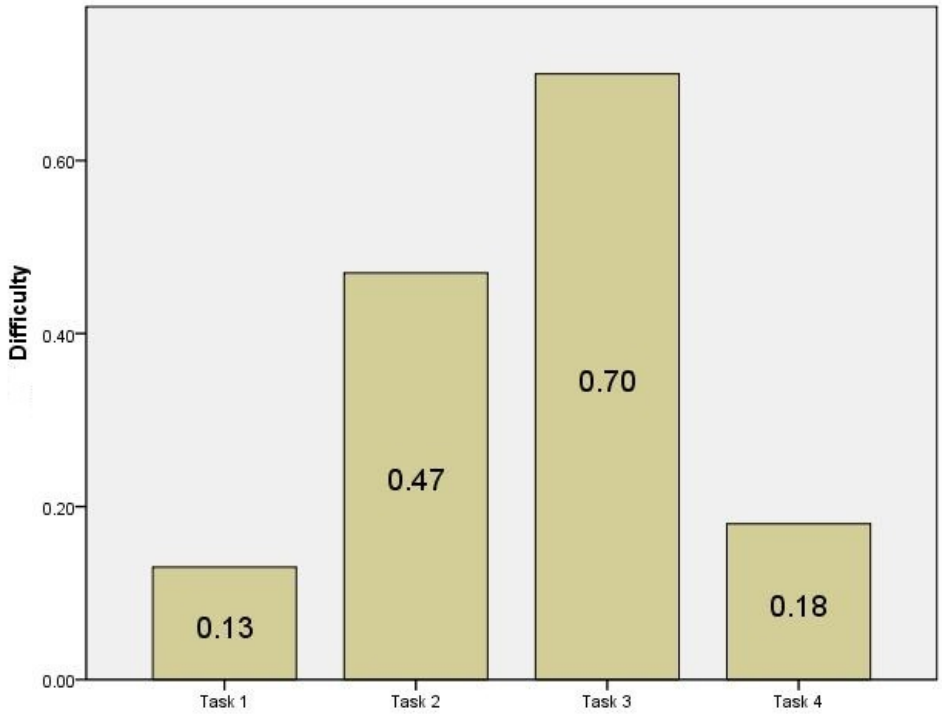

Figure 8.6: Task difficulty based on the success rate per task: finished with ease (0), finished with difficulty (1), not finished (2)

However, looking into the distribution of different success rates within each task, as shown in Figure 8.7, Task 2 had the most failures (23.33% participants did not finish the task). Task 1 was completed successfully by all participants, with only four subjects reporting difficulty. Interestingly, if

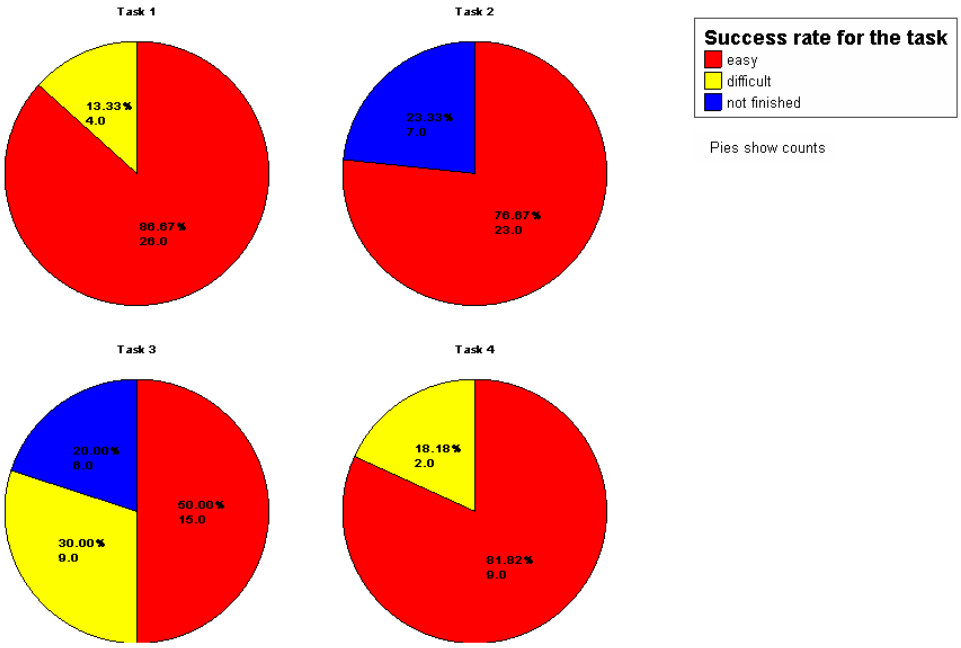

Figure 8.7: Frequency of different success rates per task

participants managed to finish Task 2 successfully, they have not experienced any difficulties. Task 3 had not been finished in 20% of the cases. In comparison to Task 2, this is slightly better, however, a large portion of those who completed task 3 reported difficulty to do so. Task 4 was finished by only 11 participants, majority of which reported that they have finished it with ease.

User Satisfaction

With regard to the SUS score, the overall mean was 66.97. This is slightly lower than the SUS score of 69.37 for the QuestIO user evaluation presented in Section 7.4. The mode and the median were equal to the one in the previous evaluation (70). The range was much bigger starting at the minimum of 25, and spreading until the maximum of 95 (in comparison to 52.5 minimum and 85 maximum in the evaluation of QuestIO). Overall, this is a good result.

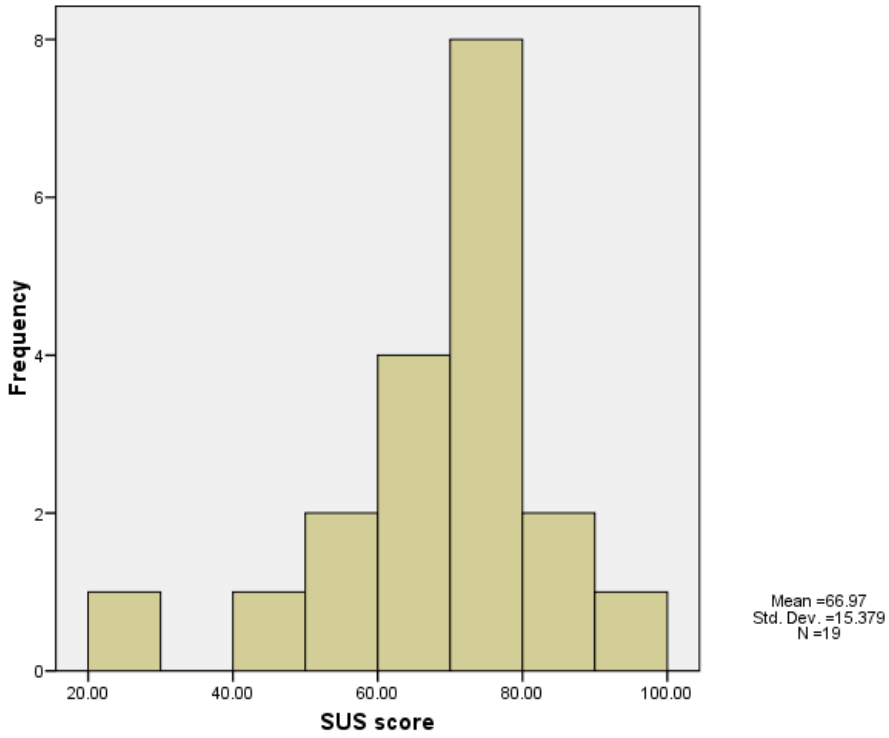

Figure 8.8: The distribution of SUS scores for 19 participants who completed the questionnaire

Comparison with QuestIO

Task 1 was intended to test the difference between effectiveness and efficiency of FREyA in comparison to QuestIO. In other words, this test evaluates whether the feedback in FREyA improves usability of the system.

Preparing data As in the FREyA evaluation, the first task was equivalent to the two tasks (Task 1 and 2) from the QuestIO evaluation, we first merged the results of these two into one. For effectiveness, in case that the success score differed for the two tasks in the previous study, the higher one was picked as the representative. For example, if one of the tasks was marked as *task completed with ease (0)*, and the other *failed to complete (2)*, the overall assigned score was *failed to complete (2)*. For efficiency, measured through the time spent on the task, we summarized the time for Tasks 1 and 2 into one value.

Effectiveness We test the significance of the difference in effectiveness using Chi-Square test of independence. Our null hypothesis is that there is no relation between the system used (independent variable) and effectiveness measured through the success rate (dependent variable). With p=0.01 and $\chi^2 = 8.313$ we can reject the null hypothesis, leading us to the conclusion that the difference in effectiveness in using the two systems (0.67 for QuestIO, 0.13 for FREyA) is significant. This indicates that the new usability features that exist in FREyA and do not in QuestIO, had a positive impact on effectiveness.

Efficiency With regard to the efficiency of the two systems, although the overall result differs (180.5 seconds on average for Tasks 1 and 2 for QuestIO, 155.27 seconds for FREyA), 2-tailed independent t-test reveals that this difference is not significant (t=0.188, p=0.852 with equal variances assumed, and t=0.287), and thus we retain the null hypothesis and conclude that there is no relation between the system used (independent variable) and efficiency measured through the time spent on task (dependent variable). This indicates that new usability features of FREyA did not have a significant influence on how quickly subjects could finish tasks.

Subjective Measures of User Satisfaction

In order to test the subjects' subjective perception of the specific features, we asked them about *Identified context* and *Query formulation*.

Identified context Figure 8.9 shows the distribution of the subjects' subjective judgment on the *Identified context*. The exception is task 2 for which we did not ask subjects about *Identified context* explicitly, but instead we asked them whether it was clear that *there were no states (or no parameters for the GATE domain)*.

With regard to the first three tasks, the *Identified context* was most useful for the easiest task (task 1), and the least useful for Task 3. It is surprising

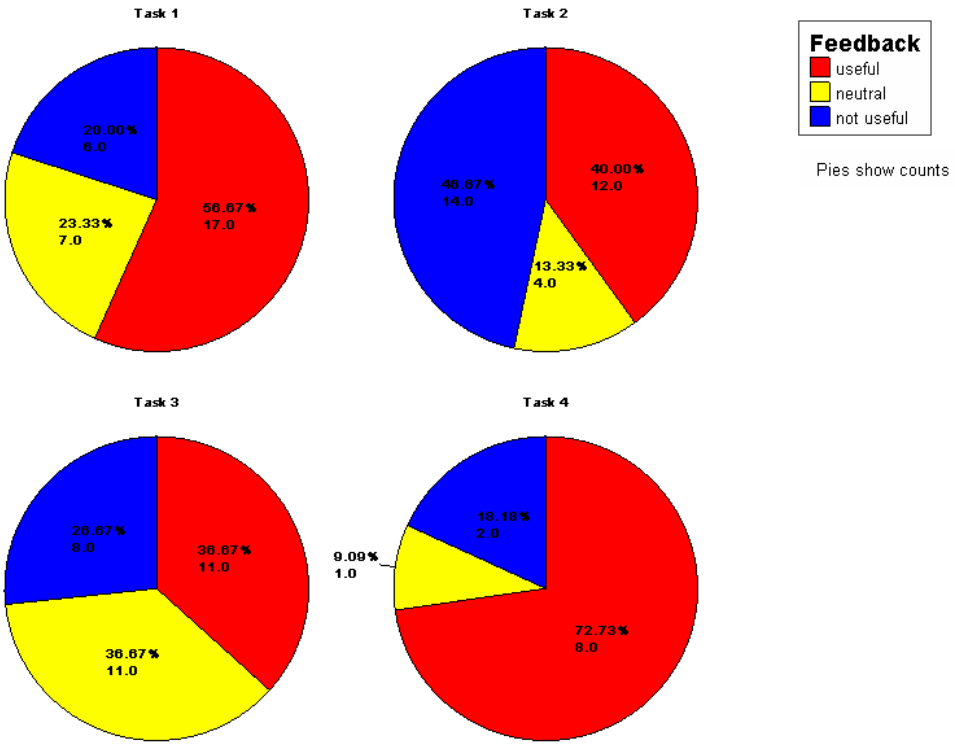

Figure 8.9: Clarity of feedback, all tasks

that the large percent of subjects found the *Identified context* confusing or neutral when doing task 1, although all of them successfully finished the task. Namely, six subjects who found the *Identified context* confusing when doing task 1, reported that *several of the generated examples were confusing*

or nonsensical e.g. 'state – is mountain of – rainer'; the reason for this was that the system showed the recognised elements of the query in order in which they appeared in the query; as in the ontology, the actual relation *is mountain of* has *state* as a range, and *mountain(rainer)* as a domain, the natural way of showing this to the user would be: *rainer – is mountain of– state*. However, this kind of interpretation is a step towards showing triples to the end-user, and for more complex queries, these would need to be multiplied. As we have previously discussed (see Section 8.1), with this initial prototype of FREyA, the intention was to mark question terms as recognised without going deeply into the complexities of ontology structure.

With regard to the subjects who failed to complete task 2, this happened due to three reasons:

- Two out of seven (28.57%) said the *System provided **confusing** output so they could not figure out what to do.*
- Two out of seven (28.57%) said the *System provided **no** output so they could not figure out what to do.*
- Three out of seven (42.86%) could not find any information about 'Hawaii' due to the system failing to recognize 'Hawaii' with uppercase.

The last group can be classified as the system failure, and therefore we conclude that the remaining 57.14% failures happened due to the users struggling to understand the feedback. In other words, four subjects failed to finish task 2 due to not being able to understand the system's feedback.

Looking at the results of 23 participants who claimed that they finished task 2 with ease (see Figure 8.10):

- For 8 out of 23 (34.78%) it was **not** clear that there are no bordering states/no init time parameters for Rasp parser for that specific task, but they could successfully finish the task by looking at the results for some other queries. For example, some of them said that *they determined that the system meant that there were no bordering states by querying another state with others bordering it.*

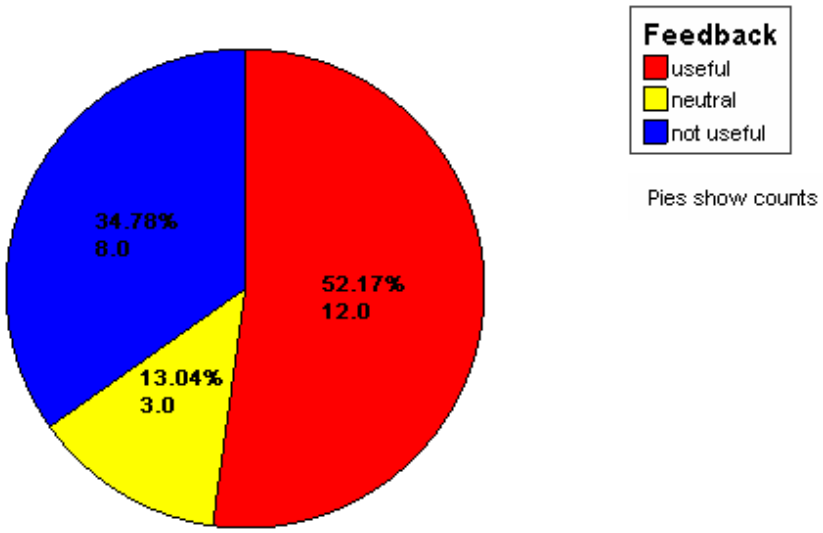

Figure 8.10: Clarity of feedback for Task 2 considering only the participants who have finished the task with ease

- 3 out of 23 (13.04%) experienced **the system failure** in not recognizing Hawaii spelled with an upper case; they have managed to reformulate the query in order to finish the task successfully.
- For 12 out of 23 (52.17%) participants, the feedback shown by the system was clear enough to draw conclusions that there is no answer.

Overall, 12 subjects struggled to understand the system's feedback, 4 out of which have not found the alternative way to solve the task. This forms 40% of all subjects, which is quite a high number. Only 40% of subjects found the feedback messages useful, even though 76.67% of them reported that they *finished the task with ease*.

From task 2, we conclude that the *Identified context* coupled with the message *No relation found within identified context* was not useful even though 76.67% of subjects have found the way to complete the task successfully, usually by trying similar queries for which a result existed.

Looking into the details for the six subjects (20%) who failed to complete

task 3, one of the subjects stated that *"system provided confusing output: couldn't manage to find out how to formulate the query; tried several ones by refinement"*: when investigating further the queries of this user, we found out that he tried 18 different queries, most of which gave some results, however, they were either too generic (e.g., *PRs*), or too specific and long, and also very similar to the wording of the actual task, for example *"creole plugin prs parameters that are the same as the parameters of gate morphological analyser"*.

Query formulation Figure 8.11 illustrates how easy was to formulate the queries for the tasks. Subjects struggled most with task 3: many of them have tried to input the exact wording of the task and then, since the system showed the recognised concepts but no answer, 80% tried to reformulate the query and successfully finished it, while 20% gave up (tasks not finished).

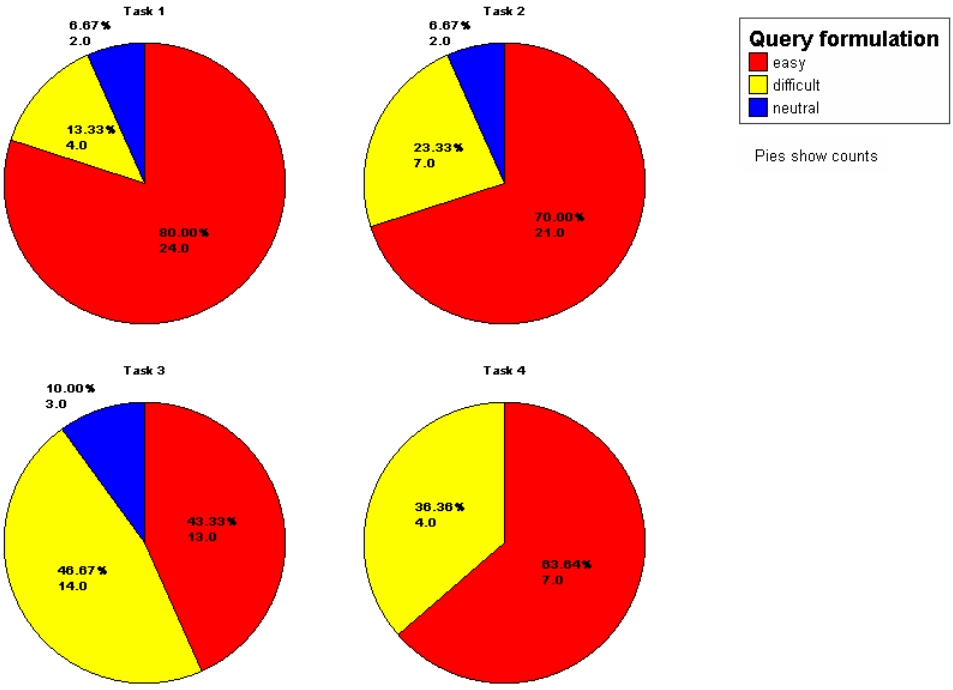

Figure 8.11: Query formulation per task

Table 8.1 illustrates the average number of queries across all subjects, which were used for finishing the first three tasks.

The lowest number of queries was for task 2. However, it seems that for both

Task	Avg. #queries	Avg. #queries (successfully finished tasks)	Avg. #queries (tasks not finished)
1	7.27	7.27	n/a
2	4.10	3.17	7.14
3	6.03	5.5	8.17

Table 8.1: Number of queries per task across all subjects

tasks 2 and 3, the subjects who have finished the task successfully, used less queries. The logs reveal that majority of subjects who failed to complete the task, could have finished it even after the first query, given that they could understand the system's messages.

One of the subjects stated that *[FREyA] is a nice tool but can easily be fake i.e. try 'state mountains in the States' or 'state apple, monkeys, bananas, mountains in the USA'*. This is an interesting observation which is indeed true. The system does not use any predefined rules or syntax which would help it rule out the sentences such as this one. However, this is left to the user: our intention is to make the user aware of the available knowledge; if, for the given query, the user gets the wrong answer, he will at least know the reason for that. In an ideal case, the tool would indicate that *state* at the beginning of the query is recognised as *geo:State*, and the user, knowing this is not true, needs to reformulate the query (i.e. use similar words such as 'give me' or 'show' or 'list' instead of 'state' at the beginning of the query).

Comparison with QuestIO As we have assessed the query formulation in the evaluation of QuestIO, we can now compare the results of the two studies. Here we consider only the tasks that are repeated across the two studies. In other words, we compare the perception of the query formulation for Tasks 1 and 2 in the QuestIO evaluation against the perception of the query formulation for Task 1 in the FREyA evaluation. In addition, we compare the perception of the query formulation for the undefined task (Task 4) for both studies.

While the query language used in the two studies does not differ, the comparison will reveal whether there is any effect of some other factors (such as user-interaction features of FREyA and the feedback) that influence the perception of the query formulation. Our null hypothesis is that there is no

relation between the system used, and the difficulty of the query language. We use non-parametric Fisher Exact Test to assess this[4].

		Query language		
		easy	neutral	difficult
QuestIO	defined task	66.7%	29.2%	4.2%
FREyA	defined task	80%	6.7%	13.3%
QuestIO	undefined task	41.7%	50%	8.3%
FREyA	undefined task	63.6%	0%	36.4%

Table 8.2: Query formulation as perceived by subjects in the two studies

Table 8.2 shows the distribution of the answers, indicating that there were more positive answers to the perception of the query language in the FREyA evaluation, in comparison to the evaluation of QuestIO for both defined and undefined tasks. For defined tasks, Fisher's Exact test reveals that this difference is not significant (F=5.255, p=0.071) hence we retain the null hypothesis that there is no difference in how the two groups of subjects perceived the query formulation for defined tasks. For the undefined task, this difference is significant (F=8.016, p=0.015) indicating that subjects had impression that the FREyA's query language was easier than the one required by QuestIO. This might indicate that the new usability features of FREyA had a positive effect on the user's perception of the query language and helped boost the user's experience.

8.2.7 Summary and Discussion

In this chapter we presented the initial implementation of the FREyA system with the special emphasis on *feedback*, and the task-based evaluation with 30 subjects from the GATE Summer School, which was conducted with the aim to assess the new usability features of FREyA from the *end-user*'s point of view, and make comparison, where appropriate, with QuestIO. While we mention the term *end-user* quite frequently in this thesis, an interesting question is certainly who are they? Are they expected to have background in semantic technologies or can we assume that FREyA can be used by

[4]Fisher exact test is the exact version of Chi-square which is usually used for testing 2-by-2 tables, particularly for small samples. As Chi-square is an approximation, it is not as trustworthy as the exact test on the data with expected counts less than 5.

casual users of any background? FREyA attempts to render the feedback in a user-friendly manner and its eventual goal is to make the vast amount of structured information available to the casual users. However, based on the evaluation presented in this chapter, we can only make claims about the population represented by our sample which largely included computational linguists, computer scientists, and software developers, who were familiar with semantic web technologies even if not on the advanced level. With regard to the *application developers*, the system was not customised prior to this evaluation. However, it is important to highlight that the system was initialised using two completely different domains, one about the GATE software and the other one about the United States geography. Due to the huge diversity between the two domains, there were no problems caused by ambiguities, neither there was a need to combine the knowledge of the two for any of the questions.

The *feedback* was provided using two elements:

- The *Identified context* table showing all *query interpretations* to the user, where each interpretation is a linear combination of the concepts and relations between them. The order of the recognised concepts followed the order in which they appeared in the question.
- The *tree-based view* showing the concepts and their relations in the tree view, for any selected *identified context*.

We tested the effectiveness, efficiency, and also user satisfaction using the System Usability Scale. We also compared effectiveness, and efficiency with the results from the previous study to assess whether feedback makes any significant difference. In addition, we assessed subjective measures of user satisfaction through gathering the user's opinion about *Identified context* and *Query formulation*.

Task 1 was intended for a comparison of effectiveness and efficiency of FREyA with the results of the previous study with QuestIO. Task 2 was intended to test whether subjects could understand that the answer to their question was negative. Task 3 was testing whether subjects could make correct conclusions about query reformulation based on feedback. The wording of the task was such that if used as a query the answer would have not been

retrieved unless the query was reformulated or split into at least two queries. Task 4 was important only in the context of collecting new queries, and also for a comparison with the previous evaluation where appropriate.

All subjects completed Task 1 although four of them did so with difficulty. This result is significantly better ($p = 0.01$) than the result for the same task in the previous study of QuestIO, indicating that the new usability features had a positive effect on effectiveness. However, although the subjects finished this task faster than in the previous study, this difference is not significant ($p >= 0.776$).

With regard to the *Identified context*, it was not well received even for task 1 which was the easiest. For tasks 2 and 3, the *Identified context* was not key to success. Instead of understanding that there were no relations within the identified context as it was stated by the system, the subjects ended up reformulating the queries many times, and trying similar queries in order to finish the task. The average number of queries per task for the tasks completed successfully is much lower than for those that were not completed, indicating that the subjects who did not understand the system's messages, believed that they need to reformulate the query, which is what they did many times until giving up at the end. This is specifically the case for tasks 2 and 3.

Overall, our conclusion is that:

- Feedback had a positive impact on the overall effectiveness of the system, but no significant effect on efficiency.
- Feedback had a positive impact on the subject's perception of the difficulty of query formulation.
- The *Identified context* showing the linearised list of concepts was not well accepted.
- The tree-based structure especially its interactive feature was well accepted.
- Showing that the system knows about certain concepts, but cannot find any relation between them was not clear.

- For complex queries, feedback was often helpful in terms that subjects could figure out that they need to reformulate the query in order to get the correct answer.

Based on the findings from this evaluation, we moved towards the *concept-based* interpretation, in contrast to the *query-based* one which was represented by *Identified context*. This enabled us to address some of the issues revealed during the evaluation just presented, and also to address challenges highlighted in the QuestIO evaluation (Chapter 7). These challenges are the base for the improved version of FREyA – its design, implementation and evaluation are detailed in the next Chapter 9.

Chapter 9

Towards Better Usability with FREyA: Part II

In this chapter[1], we detail FREyA which goes one step further in exploring the usability enhancement methods, by combining *feedback* with *clarification dialogs* with the aim to improve the performance of NLIs to ontologies. This

[1] My initial ideas on FREyA are summarised in *D. Damljanovic, M. Agatonovic, H. Cunningham: Usability of Natural Language Interfaces for Querying Ontologies, Workshop on Controlled Natural Language (CNL'09), Marettimo Island, Italy, June 08-10, 2009.*. M. Agatonovic and H. Cunningham provided useful comments and improvements on the initial proposal. In *D. Damljanovic, M. Agatonovic, H. Cunningham: Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In Proceedings of the 7th Extended Semantic Web Conference (ESWC'10), Springer Verlag, Heraklion, Greece, May 31-June 3, 2010.* I wrote about FREyA which is largely based on the content in this chapter. While I implemented the system, I had lots of discussions with M. Agatonovic whose comments and suggestions improved the system and also the final version of the paper. H. Cunningham commented on an earlier version of this chapter, which was then included in the paper. Similar contribution of both co-authors is applicable to *D. Damljanovic, M. Agatonovic, H. Cunningham: Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In Proceedings of the 7th Language Resources and Evaluation Conference (LREC'10), ELRA 2010, La Valletta, Malta, May 17-23, 2010.* – Sections 9.2 and 9.7.4 are an updated version of this paper. Section 9.4 is an updated version of *D. Damljanovic. Towards Portable Controlled Natural Languages for Querying Ontologies. In Rosner, M., Fuchs, N., eds.: Proceedings of the 2nd Workshop on Controlled Natural Language. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, Marettimo Island, Sicily, September 13-15, 2010.* Sections 9.6 and 9.7.5 are an updated version of *D. Damljanovic, M. Agatonovic, H. Cunningham: FREyA: an Interactive Way of Querying Linked Data using Natural Language. In: Proceedings of 1st Workshop on Question Answering over Linked Data (QALD-1), Collocated with the 8th Extended Semantic Web Conference (ESWC 2011). Heraklion, Greece, June 2011.*

work builds up on the experience from the evaluation of QuestIO described in Chapter 7 and also on the evaluation of the initial implementation of feedback described in Chapter 8. This leads to the system which, in contrast to QuestIO, and the first version of FREyA which works with *query-based interpretations*, moves towards *concept-based interpretations*, in an attempt to:

- Improve recall by **addressing expressiveness**: generating the dialog whenever an "unknown" term appears in a question. The initial domain-lexicon is extracted from the ontology, as in QuestIO (see Section 7.1), and enriched by WordNet [Fellbaum, 1998]. "Unknown" terms are those that are identified as candidates to be linked to the ontology concepts. These terms are chosen based on the analysis of the syntactic parse tree, however this analysis does not require strict adherence to syntax and works on ill-formed and incomplete questions as well as on the grammatically correct ones (Section 9.4).
- Improve precision by **resolving ambiguities more effectively**: generating the dialog whenever a question term refers to more than one ontology concept i.e. for any ambiguities that cannot be solved automatically.

The important part of the dialog are suggestions, which are found through ontology reasoning. The system then learns from the user's selections, and improves its performance over time (Section 9.5). The complete workflow starting with the Natural Language question and ending with the answer is detailed next in Section 9.1.

In addition, while in QuestIO the results are shown to the user as a set of triples returned by SPARQL, in FREyA two important aspects are explored:

- *identification of the answer type* (Section 9.2), which is then used for
- *showing the concise answer to the user* (Section 9.3).

With regard to portability, FREyA allows different modes to be used with different datasets, so that the learning model can be built during the training phase, and used later (Section 9.6).

The system is evaluated using the Mooney GeoQuery dataset previously used in the evaluation of feedback, and also using the MusicBrainz and DBPedia datasets which are part of the Linked Open Data cloud. The results are presented in Section 9.7.

9.1 FREyA Workflow

Figure 9.1 shows the workflow which starts with the Natural Language question (or its fragment), and ends when the answer is found. Each step in the workflow is explained in details in the following sections.

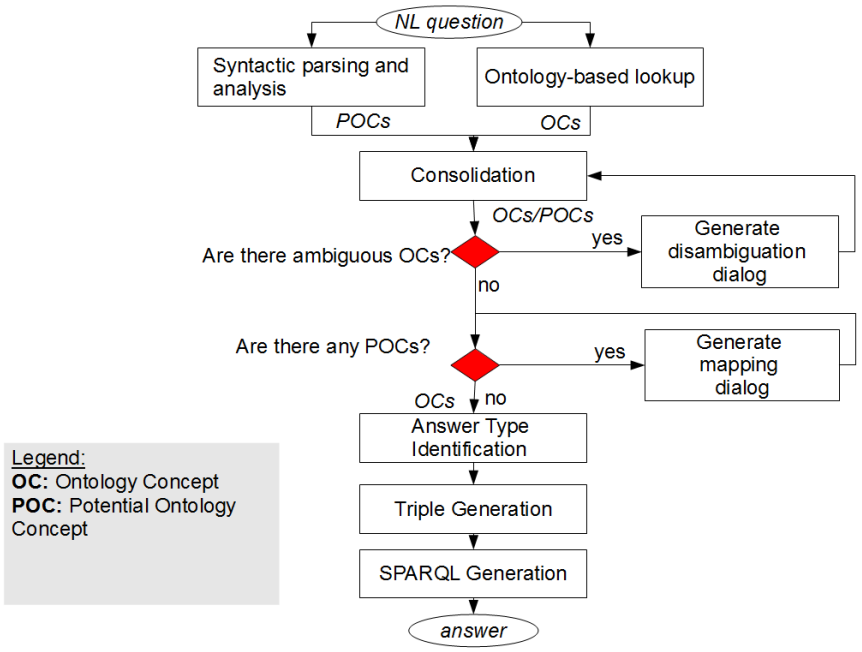

Figure 9.1: FREyA Workflow

9.1.1 Ontology-based Lookup

Ontology-based Lookup links question terms to logical forms in the ontology which we call Ontology Concepts (OCs) without considering any context or

grammar used in the question. Ontology Concepts refer to *instances/individuals*, *classes*, *properties*, or *datatype property values* such as string literals. The lookup is performed against the domain-lexicon extracted using the same principles detailed in Section 7.1. However, the OntoRoot Gazetteer used in QuestIO is memory-based, and hence is not always suitable for large-scale datasets as initialisation of the domain lexicon is time-consuming. The lexicon is derived from the semantic repository by executing a set of SPARQL queries. Moreover, the data can be distributed over various types of servers, which often allow access through SPARQL endpoints. However, depending on the repository which is used underneath, some SPARQL queries can be highly unoptimised and slow.

One optimisation which we had to perform when attempting to load the DBpedia dataset from the FactForge server[2] with FREyA, is *using a more scalable gazetteer for ontology-based lookup.* OntoRoot Gazetteer which loads the lexicon into memory could not load the DBpedia lexicon with 20G RAM, and we had to use a more scalable gazetteer called Large Knowledge Gazetteer (LKB) developed by Ontotext.

Initial version of LKB (distributed with GATE 5.1) loaded the lexicon by executing one SPARQL query and was matching the exact text from the query with the lexicalisations from the ontology as they are returned by SPARQL. OntoRoot Gazetteer is more flexible and robust than LKB, as it does a significant post-processing of the ontology lexicalisations returned by SPARQL, before it adds them to the gazetteer list. Therefore, we collaborated with Ontotext in order to merge the features of the OntoRoot and LKB gazetteers and enable loading the lexicon from the large datasets through more than one SPARQL query, and also to be able to post-process the lexicalisations returned by SPARQL in order to improve the effectiveness of the lookup.

The initial SPARQL query which is used for the DBpedia experiment is shown in Appendix C. The queries are not optimised and it would be worth to explore the alternatives that would help to achieve the same result more efficiently. For this experiment, we did not have control over factForge.net, which already included inferred triples which had to be filtered out to avoid duplicates – hence the SPARQL was more complicated than the one that

[2] `www.factforge.net`

would be as effective if the repository contained explicit statements only.

By default, the system assumes that *rdfs:label* property is used to name Ontology Concepts. However, for ontologies which use different naming conventions (such as using the special class *Alias* in PROTON previously mentioned, or using *dc:title* inside the MusicBrainz dataset, see Section 9.7.5), it is possible to predefine which properties are used for names. This will enable the system to make the distinction between making a *datatype property value element* and an *instance element*. This distinction is important as different *elements* are used differently during the *Triple Generation* and *SPARQL generation* steps.

To give an example using the MusicBrainz dataset, for the query *When did Kurt Cobain die? Kurt Cobain* would be linked to four Ontology Concepts (all individuals), one referring to *mm:Artist*, one referring to *mm:Album*, and the other two referring to *mm:Track*[3], because it is matched with the string literal of the property *dc:title* of these four URIs which is *Kurt Cobain*. However, if the question were *Did Kurt Cobain die on April 8, 1994?*, the *April 8, 1994* would be annotated as a *datatype property value element* related to the individual *Kurt Cobain*, as it is matched with the value of the property *mm:endDate*.

The ontology-based lookup relies on the human understandable lexicalisations of ontology resources and therefore, the quality of produced annotations depends directly on them. However, it is not always the case that ontology resources are followed by human understandable lexicalisations (e.g., labels). This is especially the case for properties. In addition, Natural Language is so expressive that words like *total*, *smallest*, *higher than* or *how many* cannot be understood without grammar analysis, neither they can be encoded into the relevant structure without additional processing. Some formal languages such as SPARQL have not even supported some of these structures until recently (e.g., it was not possible to do COUNT in SPARQL before version 1.1[4]). In order to capture the semantic meaning of such and similar constructions, we analyse its grammar and then translate certain question terms

[3]For clarity of presentation, we use prefix *mm:* instead of *http://musicbrainz.org/mm/mm-2.1#* and *dc:* instead of *http://purl.org/dc/elements/1.1/title* in the examples.

[4]`http://www.w3.org/TR/sparql11-query/`

into the relevant operations with ontology concepts (e.g., superlative might mean applying maximum or minimum function to the datatype property value).

9.1.2 Syntactic Parsing and Analysis

The syntactic parsing and analysis generates a parse using the Stanford Parser [Klein and Manning, 2002] and then uses several heuristic rules in order to identify *Potential Ontology Concepts (POCs)*. POCs refer to question terms/phrases which can but not necessarily have to be linked to Ontology Concepts. Although POCs are chosen based on the grammar analysis, the strict adherence to syntax is not required and the algorithm works on ill-formed questions and question fragments as well as on the grammatically correct ones. For example, noun phrases (NP), nouns (NN*), verbs (VB*), or WH-phrases such as *Where*, *Who*, *When*, *How many* are expected to be found by our *POC Identification algorithm*. This algorithm is based on the identification of *prepreterminals* and *preterminals* in the parsed tree, and also on their part-of-speech tags. A node is a *prepreterminal* if all its children are *preterminals*. A *preterminal* is defined to be a node with one child which is itself a leaf. The *POC Identification algorithm* is configurable in a sense that it can be set to ignore, or consider specific part-of-speech tags. The high-level pseudo-code looks as follows:

```
1. find all X where X is a prepreterminal
2. if X is NP or NN* then POC = X
3. if POC contains ADJP or ADVP
   then split POC into several POCs
4. find all Y where Y is preterminal AND Y is not
   identified as POC at step 2
5. if Y is in the list of POS tags to consider AND
      Y is not in the list of POS tags to ignore then
      POC = Y
```

9.1.3 Consolidation

The consolidation algorithm aims to merge the output of the *Ontology-based lookup* and the *Syntactic parsing and analysis* by mapping the identified POCs into OCs. While this algorithm attempts to perform this step automatically, it is possible that it requires attention from the user. This is the case when there are ambiguous OCs in the question which could not be resolved automatically, or when a POC could not be mapped to an OC automatically. More concretely, a Potential Ontology Concept is mapped to an Ontology Concept in two ways:

1. *Automatically*: if it overlaps with the Ontology Concept in a specific way:
   - Both POC and OC refer to the same text span in the question ($OC == POC$). For example, in *which rivers flow through Texas?*, *rivers* can be identified as an OC, as referring to the class *geo:River*, while it can also be identified as a POC. In this case, the POC is automatically mapped to the OC, as $OC == POC$ (the starting and ending offsets are identical).
   - A POC refers to the text span which is contained within the span to which an OC refers ($POC \subset OC$).
   - The OC is contained within the POC which contains a determiner ($POC = DT + OC$). If we look at the query *Give me all former members of the Berliner Philharmoniker.*, the *POC Identification Algorithm* will find that *the Berliner Philharmoniker* is a POC, while the *Ontology-based Lookup* will find that *Berliner Philharmoniker* is an OC, referring to an instance of *mm:Artist*. As the only difference in the POC and the OC text is a determiner (*the*), the consolidation algorithm will resolve this POC by removing it, and by verifying that this noun phrase refers to the OC (with *dc:title Berliner Philharmoniker*).
2. *By engaging the user*: in cases when the system fails to automatically resolve a POC (or when it is configured to work in the *forceDialog* mode, see Section 9.6) it will generate the dialog.

When the system fails to automatically generate the answer (or when it is configured to work in the *forceDialog* mode, see Section 9.6) it will prompt the user with a dialog. There are two kinds of dialogs in FREyA:

1. *The Disambiguation dialog* involves the user to resolve identified ambiguities in the question.

2. *The Mapping dialog* involves the user to map a POC to the one of the suggested OCs.

While the two types of dialogs look identical from the user's point of view, there are differences which we will highlight here. Firstly, we give a higher priority to the disambiguation dialog in comparison to the mapping dialog. This is because our assumption is that the question terms which exist in the graph (OCs) should be interpreted before those which do not (POCs). Note that FREyA does not attempt to interpret the whole question at once, but it does it for one pair of OCs at the time. In other words, one resolved dialog can be seen as a pair of two OCs: an OC to which a question term is mapped, and the neighbouring OC (context). Secondly, the way the suggestions are generated for the two types of dialogs differ. The disambiguation dialog includes only the suggestions with Ontology Concepts that are the result of the ontology-based lookup (unless it is extended using the *forceDialog* mode, see Section 9.6). The mapping dialog, in contrast, shows the suggestions that are found through the ontology reasoning by looking at the closest Ontology Concepts to the POC (the distance is calculated by walking through the parsed tree). For the closest OC X, we identify its neighbouring concepts which are shown to the user as suggestions. *Neighbouring concepts* include the defined properties for X, and also its neighbouring classes. *Neighbouring classes* of class X are those that are defined to be 1) the domain of the property P where range(P)=X, and 2) the range of the property P where domain(P)=X. Finally, the sequence of disambiguation and mapping dialogs themselves controlled differently for these two kinds of dialogs:

- *The disambiguation dialogs* are driven by the question *focus* or the *answer type*, whichever is available first: the closer the OC to be disambiguated to the question focus/answer type, the higher the chance

that it will be disambiguated before any other. The question focus is the term/phrase which identifies *what the question is about*, while the answer type identifies the type of the question such as *Person* in the query *Who owns the biggest department store in England?*. The focus of this question would be *the biggest department store* (details of the algorithm for identifying the focus and the answer type are described in Damljanovic et al. [2010a]). After all ambiguities are resolved the FREyA workflow continues to resolve all POCs through the mapping dialogs.

- *The mapping dialogs* are driven by the availability of the OCs in the neighbourhood. We calculate the distance between each POC and the nearest OC inside the parsed tree, and the one with the minimum distance is the one to be used for the dialog before any other.

9.1.4 The Disambiguation Dialog

For ambiguous OCs that are identified through the *Ontology-based lookup*, the dialog is modelled and the user needs to disambiguate the specific meaning. This dialog consists of the *ambiguous term* and the list of OCs. The user is then asked:

```
I struggle with [ambiguous term]. Is [ambiguous term] related to:
    OC1
    OC2
    ...
    OCn
```

In QuestIO, we use the approach for automatic disambiguation of question terms: we consider all possible interpretations of the question, and then rank them before we generate the SPARQL query – this way we automatically disambiguate the terms referring to more than one concept by simply excluding those which are not ranked first. This approach proved problematic for ontologies which have hundreds of property definitions. For example, the PROTON ontology which is the core of the Travel Guides ontology (see Section 7.4.2) has more than 150 relations defined between classes *Country*

and *Continent*. Disambiguating relations in this case requires mapping the user's expression to the certain property which is difficult due to expressiveness of Natural Language, but also due to the large number of candidates to which a question term could be mapped.

In FREyA, the automatic disambiguation could be corrected by involving the user into the dialog. For example, if someone is inquiring about *Mississippi*, we might not be able to automatically derive whether the query refers to *geo:River*[5], or *geo:State*, because we do not have enough context for effective disambiguation. However, if the question is *which rivers flow through Mississippi?*, the context can help automatically derive that the question is about *Mississippi state* due to the existing relation in the ontology such as *geo:River – geo:flowsThrough – geo:State*.

9.1.5 The Mapping Dialog

For all POCs that could not be automatically resolved, the dialog is modelled which consists of the *unknown/POC term*, and the list of suggestions. The user is then asked:

```
I struggle with [POC term]. Is [POC term] related to:
    suggestion 1 (OC1)
    suggestion 2 (OC2)
    ...
    suggestion n (OCn)
```

Note that while the OCs in the *Disambiguation dialog* are found through the ontology-based lookup, the OCs (suggestions) in the *Mapping dialog* are found based on the ontology reasoning – they are derived based on the closest OC to the *POC term*[6]. The *closest OC* is found by walking through the syntax tree. Based on the type of the closest OC, rules for generating suggestions vary (see Table 9.1.5). Generating suggestions based on context ensures that any suggestion that is selected by the user can be used to generate the answer.

[5]For clarity of presentation, we use prefix *geo:* instead of *http://www.mooney.net/geo#* in all examples.

[6]In the specific cases the disambiguation dialog can be extended by generating suggestions using the Mapping Dialog rules described in Table 9.1.5, see Section 9.6

Table 9.1: Generating suggestions based on the type of the nearest OC

Type of the closest OC	**Generating suggestions**
class or instance	*get all classes connected to the OC by exactly one property, and all properties defined for this OC*
datatype property of type number	*maximum, minimum and sum function of the OC*
object property	*get all domain and range classes for the OC*
datatype property value	*get suggestions for the instance to which this value belongs*

Option *none* is always added to the list of suggestions (see Table 9.2), unless FREyA is configured differently (see Section 9.6 on different modes). This allows the user to ignore suggestions if they are irrelevant. That is, the system assumes that the POC in the dialog should not be mapped to any suggested OCs, and therefore the system would learn that this POC is either: 1) incorrectly identified, or 2) cannot be mapped to any OC as the ontology does not contain the relevant knowledge. While this option will not be of a huge benefit to end-users, it is intended to identify flaws in the system and encourage improvements.

The task of creating and ranking suggestions before showing them to the user is quite complex, and this complexity arises as the queried knowledge source grows.

Ranking suggestions

The initial ranking in FREyA is based on the string similarity between a POC term and suggestions, and also based on the synonym detection:

String similarity. We combine Monge Elkan[7] metrics with Soundex[8] algorithm. When comparing two strings the former gives a very high

[7] see `http://www.dcs.shef.ac.uk/~sam/stringmetrics.html#monge`
[8] `http://en.wikipedia/wiki/Soundex`

Table 9.2: Sample queries and generated suggestions for the identified POCs

Query	**POC**	**Closest OC**	**Suggestions**
population of cities in California	population	geo:City	1. city population 2. state 3. has city 4. is city of 5. none
population of California	population	geo:california	1. state population 2. state pop density 3. has low point ... n. none
which city has the largest population in California	largest population	geo:City	1. max(city population) 2. min(city population) 3. sum(city population) 4. none

score to those which are exact parts of the other. For example, if we compare *population* with *city population*, the similarity would be maximised as the former is contained in the latter. The intuition behind this is that the ontology concepts are usually named using *camelCased* names, and are more explicit than how they are usually referred to using natural language, e.g., *cityPopulation*, *stateArea*, *projectName*, and the like. Soundex algorithm compensates for any spelling mistakes that the user makes – this algorithm gives a very high similarity to the two words which are spelled differently but pronounced similarly.

Synonym detection. We use WordNet [Fellbaum, 1998] in order to retrieve synonyms of a POC. For example, if a question is *What is the highest peak in the US?*, although there is no mention of *US* in the ontology, WordNet would list *The States* as a synonym for *US*. This would match with the *geo:State* in the ontology and therefore, this option would be ranked very high.

When ambiguous OCs and all POCs are resolved, the query is interpreted as a set of OCs. At this point, there is enough information for identifying the answer type. Before going into details about the Answer type algorithm,

we first explain how generated OCs are combined into triples and used to generate the SPARQL query.

9.1.6 Combining Ontology Concepts into Triples and Generating SPARQL

The list of Ontology Concepts is prepared to conform to the structure that is suitable for generating triples. As the triples are in a form

```
SUBJECT        - PREDICATE - OBJECT
CLASS/INSTANCE - PROPERTY  - CLASS/INSTANCE/LITERAL
```

we first insert any potential *joker* elements in between OCs, if necessary. *Jokers* are wildcards or variables used instead of *classes, instances, literals* or *properties* to generate query interpretations in a triple format. At the time of generating these interpretations it is not known what kind of elements can be expected, and hence the jokers are used. The rules for inserting joker elements are as follows:

- If the first or the last element is a property, then we add a *Joker* element at the beginning or at the end of the list, respectively; a joker here is a variable representing a class, an instance or a datatype property value (literal).
- If any two classes, instances, or datatype property values in the list of OCs are next to each other, we insert the *Joker* element representing a *property* between them.
- If any two properties in the list of OCs are next to each other, insert a *Joker* element representing a *class/datatype property value* between them.

For example, if the first two OCs derived from a question are referring to *a property* and *a class* respectively, one *joker* class would be added before them. For instance, the query *what is the highest point of the state bordering Mississippi?* would be translated into the list of the following OCs:

```
geo:isHighestPointOf  geo:State  geo:border  geo:mississippi
PROPERTY              CLASS      PROPERTY    INSTANCE
```

These elements are transformed into the following:

```
?     geo:isHighestPointOf geo:State geo:border geo:mississippi
JOKER PROPERTY1            CLASS1    PROPERTY2  INSTANCE
```

The next step is *generating a set of triples from OCs*, taking into account the domain and the range of the properties. For example, from the previous list, two triples would be generated[9]:

```
? - geo:isHighestPointOf - geo:State;
geo:State - geo:borders - geo:mississippi (geo:State);
```

The last step is *generating the SPARQL query*. Set of triples are combined and based on the OC type, relevant parts are added to the *SELECT* and *WHERE* clauses. Following the previous example, the SPARQL query would look like the following:

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix geo: <http://www.mooney.net/geo#>
select ?firstJoker  ?p0  ?c1  ?p2  ?i3
where { { ?firstJoker ?p0 ?c1 .
filter (?p0=geo:isHighestPointOf) . }
?c1 rdf:type geo:State .
?c1  ?p2 ?i3 .
filter (?p2=geo:borders) .
?i3 rdf:type geo:State .
filter (?i3=geo:mississippi) . }
```

9.1.7 An Illustrative Example

Figure 9.2 shows the syntax tree for the query *what is the population of new york*. As *new york* is identified as referring to both *geo:State* and *geo:City*

[9]Note that if *geo:isHighestPointOf* had the *geo:State* as a domain, the triple would look like:*geo:State - geo:isHighestPointOf - ?;*.

in the ontology, we first ask the user to disambiguate (see Figure 9.2 a.)). If he selects *city (geo:City)*, we start iterating through the list of POCs. The first POC (*new york* as a city) overlaps with an already identified ontology concept, which causes its immediate verification so we skip it. The next one (*population*) is used to generate suggestions. Among them there will be *city population (geo:cityPopulation)* and after the user select this from the list of available options, we verify that *population* refers to the datatype property *geo:cityPopulation* (see Figure 9.2 b.)).

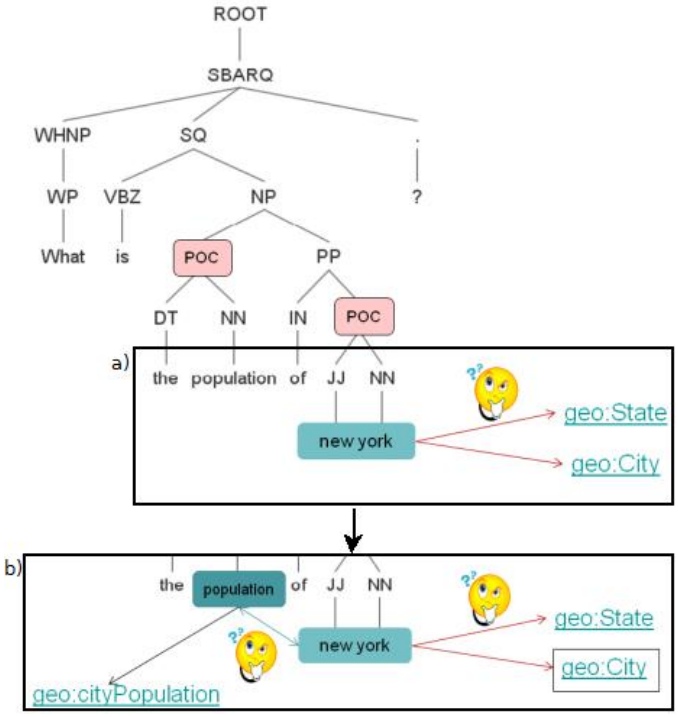

Figure 9.2: Validation of potential ontology concepts through user interaction: an example

An example of the generated suggestions for the same query is shown in Figure 9.3. Suggestions are made based on *city (geo:City)* which is the closest OC. If the user selected *state (geo:State)*, the list of suggestions would contain different options starting with *state population (geo:statePopulation)* (see Figure 9.4). We can see the difference in the generated suggestions in the cases when the user selected that *new york* means the *city*, and the *state*,

respectively. The following answer differs as well.

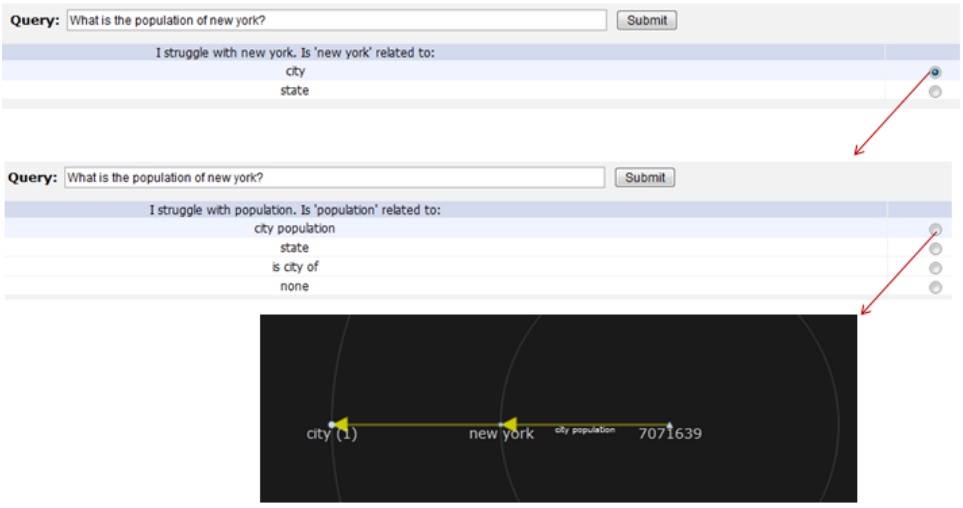

Figure 9.3: Generated suggestions and the result for *city population of the new york city*

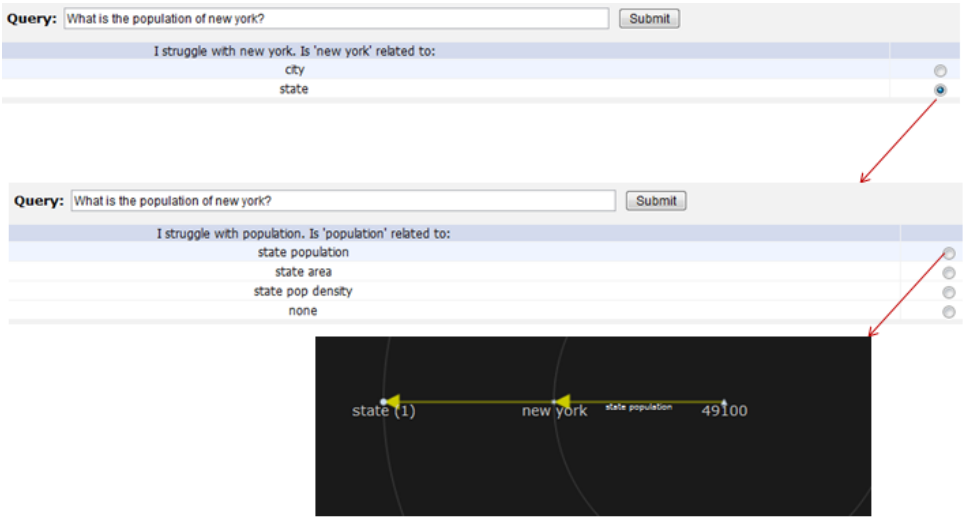

Figure 9.4: Generated suggestions and the result for *state population of new york state*

9.2 Answer Type Identification

Most NLI systems classify questions based on the type such as *What*, *Why*, *Who*, *How*, *Where*, which is followed by the identification of an answer type. The *answer type* refers to the type of the answer, such as *Person* or *Organisation* for questions starting with *Who*.

In NLIs to unstructured data, such as open domain QA systems, the answer type is derived following the question classification. The two most common approaches are [Greenwood, 2006]: 1) manually constructed rules for automatic classification 2) fully automatically constructed classifiers – usually based on Machine Learning algorithms such as Nearest Neighbour (NN), decision trees (DT) and support vector machines (SVM). Both approaches have drawbacks. In the former case, rules are hand-crafted and therefore it takes a considerable amount of time to generate them. In the latter case, automatic classifiers work well only if trained with a large amount of data, but even then the problem is their performance at runtime.

However, identifying the answer type is not always sufficient for finding answers. In Moldovan and Harabagiu [2000] the identification of the *answer type* is followed by the identification of the *focus*. According to Moldovan and Harabagiu [2000], a *focus* is a word or a sequence of words which define the question and disambiguate it by indicating what the question is looking for. For example, in *what is the largest city in Germany?* the *focus* is *largest city*. Figure 9.5 shows a part of the table from Moldovan and Harabagiu [2000] with examples of question categories, subcategories, answer types and focuses of questions. Unlike their approach which is in-line with traditional approaches used in open-domain QA systems, we skip the identification of the question category, and first try to identify the question *focus*, which is used in the subsequent steps to identify the answer type.

In NLIs to unstructured data, the answer type is derived from predefined taxonomies which are more or less fine-grained. In the case of NLIs to structured data, the answer type is usually aligned with the queried knowledge structure, as this could change over time (for example, if the ontology which is being queried changes or if the system is being ported to work with a different domain/ontology). It is not trivial to translate an arbitrary question

Q-class	Q-subclass	Nr.Q	Nr. Q answ.	Answer type	Example of question	Focus
what		64	54			
	basic what	40	34	MONEY/NUMBER/ DEFINITION/TITLE/ NNP/UNDEFINED	*What was the monetary value of the Nobel Peace Prize in 1989?*	monetary value
	what-who	7	7	PERSON/ ORGANIZATION	*What costume designer decided that Michael Jackson should only wear one glove?*	costume designer
	what-when	3	2	DATE	*In what year did Ireland elect its first woman president?*	year
	what-where	14	12	LOCATION	*What is the capital of Uruguay?*	capital
who		47	37	PERSON/ ORGANIZATION	*Who is the author of the book "The Iron Lady: A Biography of Margaret Thatcher"?*	author
how		31	21			
	basic how	1	0	MANNER	*How did Socrates die?*	Socrates
	how-many	18	13	NUMBER	*How many people died when the Estonia sank in 1994?*	people
	how-long	2	2	TIME/DISTANCE	*How long does it take to travel from Tokyo to Niigata?*	–
	how-much	3	2	MONEY/PRICE	*How much did Mercury spend on advertising in 1993?*	Mercury
	how-much-<modifier>	1	0	UNDEFINED	*How much stronger is the new vitreous carbon material invented by the Tokyo Institute of Technology compared with the material made from cellulose?*	new vitreous carbon material
	how-far	1	1	DISTANCE	*How far is Yaroslavl from Moscow?*	Yaroslavl
	how-tall	3	3	NUMBER	*How tall is Mt. Everest?*	Mt. Everest
	how-rich	1	0	UNDEFINED	*How rich is Bill Gates?*	Bill Gates
	how-large	1	0	NUMBER	*How large is the Arctic refuge to preserve unique wildlife and wilderness value on Alaska's north coast?*	Arctic refuge
where		22	16	LOCATION	*Where is Taj Mahal?*	Taj Mahal
when		19	13	DATE	*When did the Jurassic Period end?*	Jurassic Period

Figure 9.5: Sample questions with the identified question type, the answer type and the focus (taken from Moldovan and Harabagiu [2000][p.3])

into a relevant logical representation or a formal query which will lead to the correct answer [Tang and Mooney, 2001]. However, when querying ontologies in order to find the answer to a question, the approaches of question classification can be avoided: unlike documents, which are unstructured and thus have to be processed carefully in order to locate the answer, with ontologies, definitions between concepts already exist, and taking advantage of this enables avoiding strict adherence to syntax. In what follows we describe our approach for deriving the answer type without classifying the question, but rather by combining the syntax tree and the semantics found in the ontology.

Figure 9.6 shows the workflow for the identification of the answer type. *QA Detector* combines syntactic parsing with a set of heuristic rules in order to identify the *focus*. For a specific type of questions, the *focus* is not so important for identification of the *answer type*, and these questions usually have the *Answer Type Identifier (ATI)*. For example, while the *focus* in *How long is Mississippi?* is *Mississippi*, what we need to know in order to find the answer type is what *How long* refers to. Therefore, *QA Detector*

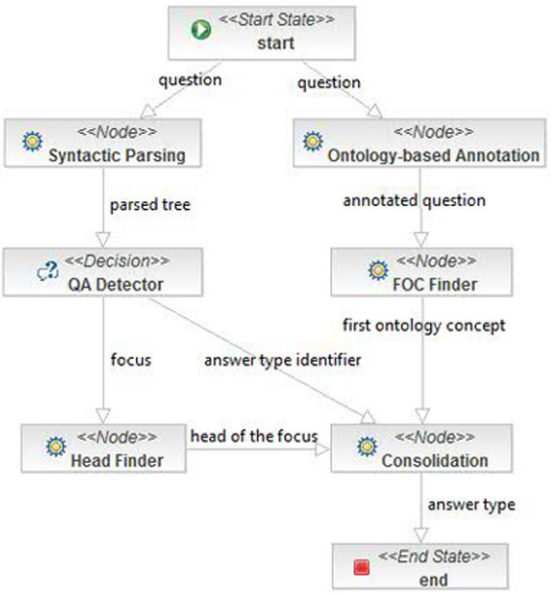

Figure 9.6: The workflow for the identification of the answer type

would identify that *How long* is the ATI. During *consolidation*, *How long* is used together with the first ontology concept (*geo:Mississippi*) to generate suggestions for the user. The user's selection is then saved as the answer type.

9.2.1 QA Detector

Similar to the *POC Identification algorithm* described previously in Section 9.1, *QA Detector* combines the syntax tree with several heuristic rules. The high level pseudo code is as follows:

```
1. find X: the first prepreterminal
2. if X is NP or NN* then focus = X
3. if X is WHADVP || WHNP || WHADJP
   || ADJP
   - if the first child is WRB, and the second is JJ or ADJP,
     then Answer Type Identifier (ATI) = X
   - if there is only WRB then ATI=WRB
```

The output of this algorithm can be:

- The *focus* of the question.
- The *Answer Type Identifier (ATI)*. This indicates that an additional input is required from the user in order to assign the answer type to the question.
- *No match found*: the parser finds neither the *focus* nor the *ATI*.

9.2.2 FOC Finder

The *FOC Finder* identifies the First Ontology Concept (FOC) in a question which is of the `class` or the `datatype property` type. This is because the answer type eventually refers to one of these two types of concepts in the ontology. Therefore, if the FOC refers to other types of ontology concepts, the procedure is as follows:

- If the FOC refers to an `object property`: perform the consolidation with *domain* or *range classes* of this property.
- If the FOC refers to an `instance`: perform the consolidation with a *class* of that *instance*.

9.2.3 Consolidation

For each query the goal is to identify the answer type. Consolidation is an attempt to achieve this by merging the output of the *QA Detector/HeadFinder* with the FOC. While the *focus* itself is important to capture relevant information which helps in finding the correct answer, the *head of the focus* is what we use in order to find the answer type. We identify the *head of the focus* using the *ModCollinsHeadFinder* class of the Stanford Parser package, which is a variant of *HeadFinder* described in Collins [1999].

The consolidation algorithm can be described using the following pseudo code:

```
if ATI!=null then
   1. generate suggestions for the user
   2. the Answer Type = the user's selection
```

```
else
   if the head of the focus != null then
   consolidate it with the FOC and identify the Answer Type
```

Depending on the relation between the *head of the focus* and the *FOC*, we apply different rules in order to identify the *answer type*. Both *head of the focus* and the FOC refer to a word or a set of words in the question. Therefore, they can either overlap, or be placed one before/after another. In the case when these two overlap the consolidation is performed as follows:

- *Exact match*: both the *head of the focus* and the FOC refer to the same word(s) in the question. Therefore, the FOC becomes the *Answer Type* of the question. For example, in *What is the capital of Texas? capital* is the *head of the focus* (as *capital* is the head of *the capital*). The same string (*capital*) is annotated as the FOC referring to *geo:Capital* in the ontology. As these two overlap meaning that their start and end offsets are equal, the answer type of the question is *geo:Capital*.
- *The FOC is contained within the head of the focus, and vice versa*: the user is asked to decide whether the identified *head of the focus* refers to the FOC or not.

When the *head of the focus* and the FOC do not overlap, the consolidation is performed as follows:

- *The head of the focus is before the FOC*: for example, in *what is the area of Idaho?* the *focus* is *the area*, and the *answer type* cannot be resolved without a dialog because the only ontology-based annotation in this question is *Idaho* referring to *geo:Idaho*, a country; the user must choose that *area* refers to one of the suggestions generated based on the neighbouring Ontology Concepts in the question (*geo:Idaho*). This is based on the same principles which are explained earlier in Section 9.1.5. More details are given in Section 9.2.4
- *The system failed to identify the focus or the ATI in the previous step*: in this case the FOC becomes the answer type. For example, in *what is the most populous state?* the *focus* is not identified due to our

algorithm relying on *prepreterminals*; in this case, as the FOC refers to *geo:State*, *geo:State* becomes the *answer type*.

- *The head of the focus is after the FOC*: in this case, the FOC becomes the answer type. For example, in *what state borders Michigan? borders* is incorrectly identified as the *focus* while *state* is annotated as the FOC; therefore, the answer type is consolidated into *geo:State*. This consolidation rule is usually used to correct the mistakes of the parser. Another example is the query *which rivers flow through Nevada?*, where the parser identifies the *focus* to be *Nevada*, which is incorrect. During the consolidation phase, if *rivers* is identified as the FOC which refers to *geo:River*, this will cause ignoring the identified *focus*, and the answer type would be *geo:River*.

9.2.4 Generating Suggestions

A list of suggestions is created based on the ontology reasoning rules, and ranked using combination of synonym detection and string similarity as previously described in Section 9.1.5. For example, in the case of *How big is Alaska?*, where Alaska is recognised as an instance of *geo:Country* in the ontology, the suggestions would include the datatype properties related to *geo:Country* such as: *geo:stateArea* and *geo:statePopulation*. Table 9.5 shows several examples of the identified ATIs, and the suggestions generated based on the FOC in the question – the answer type will be identified after the user makes a selection. Table 9.4 shows the answer type identified for the questions which did not have any ATI.

During the consolidation phase (described in Section 9.2.3) we give priority to ontology concepts, particularly to the First Ontology Concept, when consolidating it with the *focus*. However, when consolidating an ATI with an ontology concept, the ATI is usually prioritised. This is because the ATI usually refers to WH-phrases which occur at the beginning of the question. One example is *How long is Mississippi?*. Our algorithm would identify *How long* to be an ATI. On the other hand, *long* would be annotated as referring to the mountain *Longs* in the ontology. The reason is that the name *longs* is lowercased in the ontology, and therefore, our gazetteer which matches the

Table 9.3: Sample queries with the identified ATI and the generated suggestions: the answer type will be the user's selection

Query	**ATI**	**FOC**	**Suggestions**
How big is Alaska?	How big	geo:Alaska (geo:State)	1.geo:stateArea 2.geo:statePopulation 3.geo:isCityOf ... n. none
How high is the highest point in America?	How high	geo:hiPoint	1.geo:hiElevation 2.geo:isHighest PointOf 3.geo:hasHighPoint 4.none
Where is the highest point in Hawaii?	Where	geo:isHighestPointOf	1.geo:HiPoint 2.geo:State 3.none

Table 9.4: Sample queries with the identified focus and the answer type

Query	**Focus**	**FOC**	**Suggestions**	**Answer Type**
What rivers run through Colorado?	rivers (head: rivers)	geo:River	-	geo:River
What is the smallest city in Alaska?	the smallest city (head: city)	geo:City	-	geo:City
What is the population of Idaho?	population (head: population)	geo:Idaho (geo:State)	1.geo:statePopulation 2.geo:stateArea 3.none	the user's selection

root of the question term with a root from ontology lexicalisations, matches *long* in *How long* with the root of *longs*. As this partially overlaps with the ATI *How long*, we ignore it, and proceed with generating suggestions and asking the user to choose what *How long* refers to. The suggestions are generated using the neighbouring ontology concept *geo:mississippi*, and among them there will be *geo:length* which is the correct one.

One special case is when no ontology concepts are found in the question

($FOC = null$). In this case, we generate suggestions by showing the most generic concepts to the user such as top classes and properties. We then ask the user to relate the identified *focus* to one of the suggested options.

9.2.5 An Illustrative Example

The result of running the algorithm for the identification of the answer type over *what are the highest points of states bordering Mississippi?* is shown in Figure 9.7.

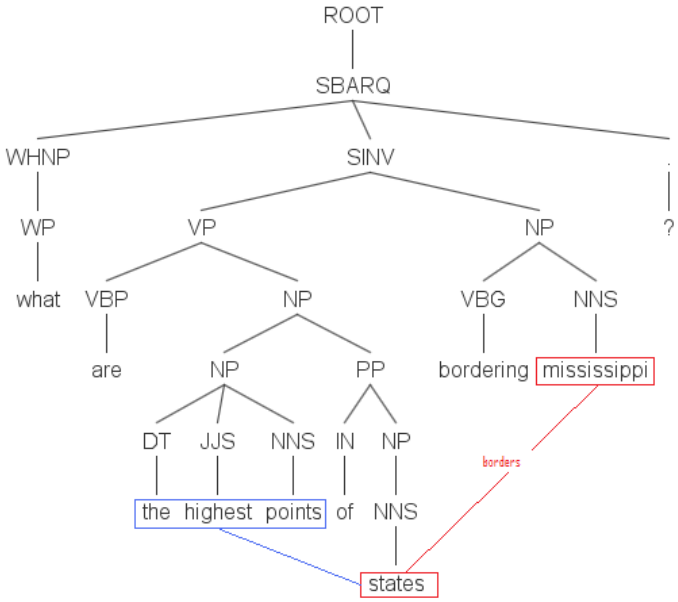

Figure 9.7: Combining the syntactic parse tree with the ontology-based lookup

Words highlighted in red (*states*, *mississippi*) are those that refer to the ontology concepts. Red lines (*borders*) are relations found based on ontology reasoning. The blue highlight (*the highest points*) refers to the identified *focus* following our algorithm. As in this example, the identified *focus* is not related to any ontology concept, it is used to generate suggestions for the user. The user will be prompt with the dialog which looks like in Figure 9.8

– the selected suggestion will be used to infer the answer type; for example, if he selects the first option the answer type will become *geo:hiPoint*.

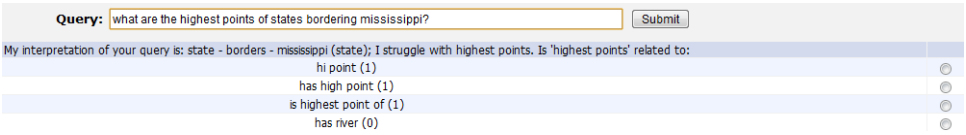

Figure 9.8: The clarification dialog preceding the identification of the answer type

The trade off is that in our initially untrained system, the user will see much more options than the ones he is interested in, but the learning mechanism which works behind the scene would put the correct ones at the top by the time. This is explained in Section 9.5.

9.3 What to Show: Presentation of Results to the User

In this section we describe how we use the *answer type* identified as described previously in Section 9.2, in order to show the concise answer to the user but also in order to show feedback.

Natural Language Interfaces for querying ontologies translate Natural Language into formal languages such as SPARQL. This translation is what most of the existing NLIs focus on, and the problem of showing the results to the user is somewhat de-emphasised.

In Chapters 5 and 6 we discussed the performance of various NLIs to ontologies, and analysed the low performance and the error rate which seems to be often caused by the way the result is shown (or not shown) to the user. Some of the reasons which were elaborated are:

- The knowledge is not in the ontology/knowledge base but the system is not capable of guiding the user to change the topic (as the answer to the initial query can not be found due to the lack of knowledge).
- Feedback messages are not helpful i.e. the user can not figure out how to proceed further.

- Users have assumptions/misconceptions about the system capabilities and the supported language.

We discussed in Chapter 6 the importance of various usability methods which can address some of these problems, however, one of the most important aspects, which needs to be considered is that the user feels confidence and trust, when using the system, and one of the methods which is important in that context is *feedback* – showing the system's interpretation to the user, and communicating the message of what the system understood *clearly*. Based on the results of the user-centric evaluation of feedback in Chapter 8, we modified our initial implementation so that in addition to the question interpretation, we use the identified answer type in FREyA to:

- display the concise answer to the user, and
- show feedback in the graph-based view.

9.3.1 Display the Concise Answer

As previously discussed, the answer type in FREyA is mapped to an ontology concept which could either be: a *class*, or a *datatype property*. Other ontology concepts are resolved to these two during the consolidation phase (see Section 9.2). Based on the type of the ontology concept, we use different albeit similar patterns for displaying the concise answer:

- Answer type is mapped to a *class*: in this case, the answer is usually the list of instances of this class, and the pattern looks like the following:

```
CLASS (number of answers):
instance 1
instance 2
...
instance n
```

- Answer type is mapped to a *datatype property*: the answer is the value of this property and the pattern is as follows:

```
DATATYPE PROPERTY (number of values):
value 1
value 2
...
value n
```

For example, in case of *Show lakes in Minnesota*, *lakes* is identified as referring to the ontology concept *geo:Lake* which is the answer type of the question. As *geo:Lake* is a class we render it as shown in Figure 9.9.

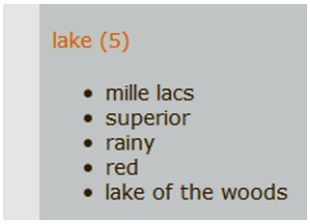

Figure 9.9: The answer to the query *Show lakes in Minnesota*

9.3.2 Feedback: the Graph-based View

According to the user-centric evaluation presented in Chapter 8, the users liked the interactive feature of the tree-based representation where they could click on the node of interest and explore details further. However, the tree-based view had some disadvantages which could not be easily overcome. Namely, attempting to render a *tree* based on a *graph* caused problems in some cases, when attempting to translate the graph returned by SPARQL into the tree like structure required by our interface. Therefore, we adapted more intuitive approach which renders the graph with all nodes, but it can be navigated in the case there are too many.

A sample graph is shown in Figure 9.10, where the answer type is placed in the centre, while the answer is available on the nearest circle. The user can click on any node in order to investigate it further – each click will cause the graph to be re-rendered and the clicked node will be placed in the centre.

We used JIT library[10] for the visualisation of this graph (as well as for the

[10] `www.thejit.org`

visualisation of the tree presented in Chapter 8).

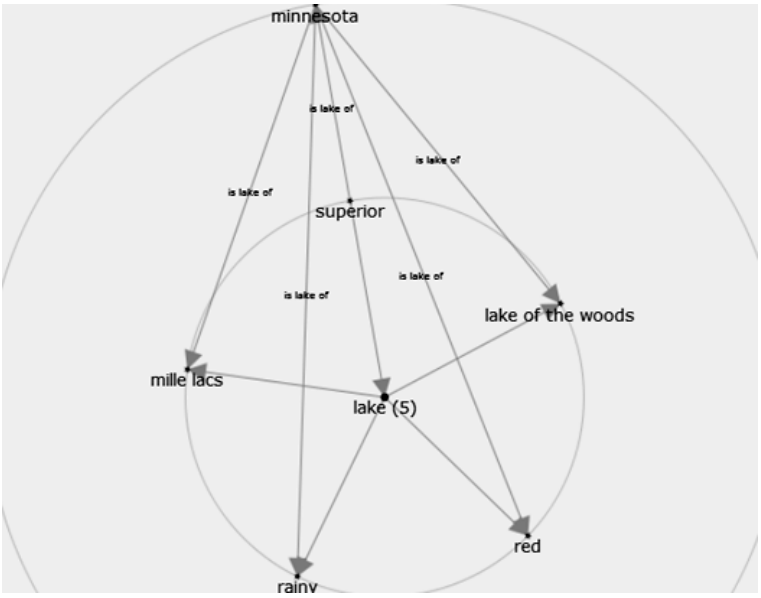

Figure 9.10: The graph showing the system's interpretation of the query *Show lakes in Minnesota*

9.4 Enriching Lexicon through User Interaction

QuestIO's approach, described in Section 7.1, generates the initial lexicon automatically from the ontology lexicalisations. With FREyA, we extend that approach with WordNet [Fellbaum, 1998], and by involving the user into the loop to enable the incremental enrichment of the lexicon over time (see Figure 9.11). When a user starts using the system, if a question term is not found in the lexicon, the *Mapping dialog* is modelled and the user is asked to map the unknown term into the ontology concept[11] and following his selection, the new term is added to the lexicon. In addition, the lexicon carries the semantics related to the context in which a certain word appeared.

The approach of extending the vocabulary is very similar to the one used in AquaLog [Lopez et al., 2007], however, there are a few differences which

[11] If the unknown term cannot be mapped to any of the generated suggestions, the user can choose the option *none* which will cause the *unknown term* in question to be ignored.

we will highlight here. Firstly, our model does not distinguish input from individual users, and building the lexicon is a collaborative effort where the input of one user is used for all the others. Our decision not to personalise this learning feature is influenced by the recent emergence of social networks which have shown the advantages of collaborative intelligence. Secondly, to the best of our knowledge, the learning mechanism in AquaLog is used for learning ontology relations only, when parts of the linguistic triples (verbs, or sometimes nouns) are associated with relations by involving the user into dialog. Our approach is more generic in that it is applied to any ontology element, not only relations. The notion of using *context* is also inherited from AquaLog, however the context is modelled differently, as it will be detailed in Section 9.5.

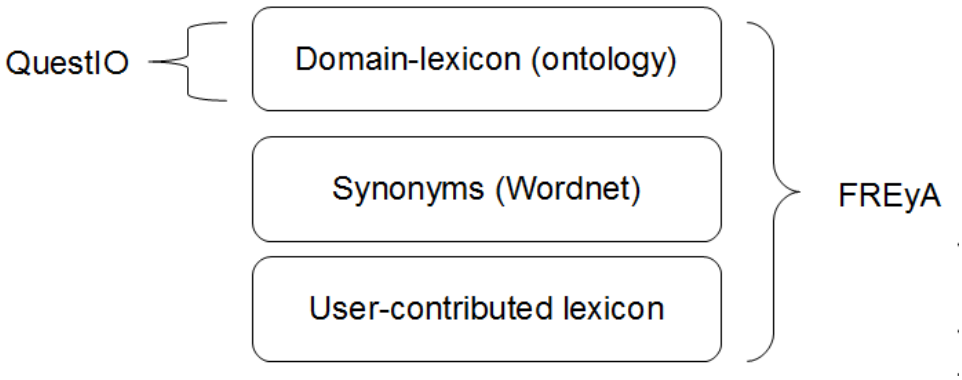

Figure 9.11: Extended Vocabulary in FREyA

Extending the existing lexicon from the user's vocabulary is performed through the following steps:

1. **Perform the ontology-based lookup**. This step is previously described in Section 9.1.1, and its role is to link question terms to logical forms in the ontology. For example, in *What is the population of New York?*, *New York* would be linked to two Ontology Concepts, one referring to *geo:newYork* and the other one referring to *geo:newYorkNY*, because it is matched with the labels of these two URIs which is *New York*. This step is identical to the one used in QuestIO (see Section 7.1).

2. **Analyse grammar** and identify the *candidate words* which could be referring to an Ontology Concept, which are called Potential Ontology Concepts or POCs. This step is previously described in Section 9.1.2. For example, if in the ontology there is no word *population* which can be used to annotate the above question about New York, as *population* is a noun, it would be identified to be a POC. However, we do not know to which concept in the ontology this noun refers, and therefore, we model the dialog.

3. **Dialog modelling**: if a POC cannot be mapped to a logical form automatically ask the user to map the unknown term (POC) into the specific ontology concept using the Mapping Dialog (see Section 9.1.5). In addition, if a question term refers to more than one OC, generate the disambiguation dialog and ask the user to choose (see Section 9.1.4). For example, in *What is the population of New York?* the question is ambiguous as it can be translated to two interpretations, where the first one is the *state population of New York (state)* and the other one is *city population of New York (city)*.

4. **Add the term to the lexicon** as a description of the OC. This description includes the *context* in which the term appears so that it can be reused in the similar context. In the case when the term was already in the lexicon, its ranking in the specific context is updated as will be detailed in Section 9.5. We previously discussed the *What is the population of new york?* example in Section 9.1.7. Figure 9.3 illustrates the example of how *population* is mapped to the *geo:cityPopulation* whenever it appears together with *New York* as a *city*. If the same word (*population*) is used together with *New York state*, then it might need to be mapped to a different form such as *geo:statePopulation* (see previously discussed Figure 9.4)[12].

Table 9.5 shows several questions and the question terms which are initially not found in the lexicon. The query term (POC) and the context (OC)

[12]Note that the system can also work in the automatic mode where it would simulate the user's selection of the best ranked options without the need to engage the user into dialog. This is discussed later in Section 9.6

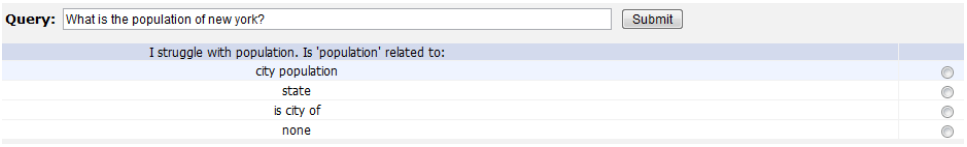

Figure 9.12: A sample dialog in FREyA

are used when generating suggestions as explained in Section 9.1.5. Following the user's selection, the new term will be added to the lexicon. For example, if *smallest* was not in the lexicon initially, and the user selects *min geo:cityPopulation* from the list of suggestions, then this term would be added to the lexicon together with its context which is *geo:City* in this case.

Table 9.5: Sample queries and generated suggestions

Query	**Query term**	**Context (OC)**	**Candidates**
What is the smallest city in Alaska?	smallest	geo:City	1. min(geo:cityPopulation) 2. max(geo:cityPopulation) 3. sum(geo:cityPopulation) 4. none
What is the population of Idaho?	population	geo:Idaho (geo:State)	1. geo:statePopulation 2. geo:stateArea 3. none

Dynamically enriched lexicon from the user-defined vocabulary is used by FREyA[13] however, the lexicon can be easily used by any other NLI system. Currently, its format is in JSON and looks like the following:

```
"Key:
    largest
    http://www.mooney.net/geo#State",
"identifier":
    "http://www.mooney.net/geo#stateArea",
"function":"max"
```

which means that if *largest* occurs followed by a lexicalisation of *geo:State*, then this should be mapped to *geo:stateArea* with the *maximum* function.

[13] `http://gate.ac.uk/freya`

The lexicon is enriched with the term *largest* which did not have any semantics attached before the user selected it through the dialog. Translating this JSON format into the knowledge representation such as OWL in a way which can then be used by any NLI system is straightforward. For example, the format of the OWL file could be such that ACE OWL Verbaliser[14] generates proper ACE sentences so that the lexicon (content words) of ACE can be enriched.

9.5 Learning from the User's Selection

Applying learning to NLIs seems analog to applying it to the Information Retrieval (IR) systems. In IR, input is a query which is usually a set of keywords which provide the most obvious set of features on which classification can be based [Belew, 2000]. This results in very large and sparse learning problems.

Supervised learning requires a set of questions with the right answers in order to give satisfying performance. Unfortunately, as noted by Belew [2000], there are many situations where we do not know the correct answers. In supervised learning every aspect of learner's action can be contrasted with corresponding features of the correct action. On the other hand, semi-supervised approach such as Reinforcement Learning (RL) aggregates all these features into a single measure of performance. Therefore, reinforcement seems to be much better for users as there is less cognitive overhead. In addition, as pointed out by Sutton and Barto [1998, p.32]:

> *"A supervised learning system cannot be said to learn to control its environment because it follows rather than influences, the instructive information it receives. Instead of trying to make its environment behave in a certain way, it tries to make itself behave as instructed by its environment."*

Semi-supervised learning such as RL allows starting with an empty model which can be randomly initialised, and then updated based on the user's input.

[14] `http://attempto.ifi.uzh.ch/site/tools/`

In our case we use a simplified approach inspired by RL to improve the ranking of the suggestions which are shown to the user:

- in the case of ambiguities: if a query concept is mapped to several ontology concepts;
- when a query concept is not automatically mapped to an ontology concept, but our system identifies it as a potential ontology concept.

Our goal is to *learn ranking* of the suggestions shown to the user.

We decide to use semi-supervised approach due to several reasons. Firstly, supervised learning goes in-line with automatic classification of the question, where each question is usually identified as belonging to the one predefined category. Our intention is to avoid this automatic classification and allow users freedom to enter queries of any form. Secondly, we want to minimize the customisation of the NLI system which is required when using supervised learning, in order to map some parts of the query to the underlying structure. For example, we want the system to suggest that *where* should be mapped to the specific part of the ontology concept such as *Location*, rather than the application developer browsing the ontology structure in order to place this mapping.

In RL, an agent learns how to achieve correct rankings by trial-and-error interactions with its environment. Based on the knowledge which is available to the agent, suggestions are ranked and these are shown to the user. In the standard reinforcement learning model an agent interacts with its environment by sensing it, and based on this sensory input chooses an action to perform in the environment. The action changes the environment in some manner and this change is communicated to the agent through a scalar reinforcement signal. There are three fundamental parts of a reinforcement learning problem: the environment, the reinforcement function, and the value function.

9.5.1 Environment

The environment encodes all knowledge which is exposed to the agent, which is in our case a set of three states: the beginning state, the desired state and

the undesired state. The initial state is represented by a list of initially ranked suggestions. A set of actions available at the initial state are the set of suggestions which are generated. Depending on the action which is taken (the suggestion selected from the list), the agent might end up at either desired or undesired state. The desired state is determined by the state which will happen after the user selects a suggestion from the list of those which are available.

9.5.2 Reinforcement Function

Our learning algorithm is inspired by a *pure delayed reward reinforcement function* [Sutton and Barto, 1998], which is defined to be zero after the user selects the clarification option except when an action results in a win (satisfying answer) or a loss (wrong answer or no answer), in which case the agent receives a +1 reinforcement for a win, and a -1 reinforcement for a loss. Because the agent is trying to maximize the reinforcement, it will learn that the states corresponding to a win are goal states and states resulting in a loss are to be avoided.

9.5.3 Value Function

Value function is a mapping from states to state values, and is expressed using Bellman equation (Equation 9.1). In order to decide which action to take, an agent usually follows a *policy* – a mapping from state to actions. The value of state x_t for the optimal policy is the sum of the reinforcements ($r(x_t)$) when starting from state x_t and performing optimal actions until a terminal state is reached.

$$V^*(x_t) = r(x_t) + \gamma V^*(x_{t+1}) \tag{9.1}$$

We initialise the value function based on the string similarity and synonym detection as described in Section 9.1.5. Discounted factor (γ) is 1. When the user changes the selection (selects the option other than the first one

suggested by the agent), the agent will learn that the previous rankings were not correct, and will recalculate its value function.

We assume that the action selected by the user is the one which is desired, and therefore give a reinforcement of +1 to such an action, while we give -1 to all the others. Therefore, if the initial ranking was wrong, there is a good chance that this is corrected by only one user choosing the right option. For example, if the question was *how many people live in florida?* the closest OC to the POC *people* is *geo:florida* which is a state. Our ranking mechanism would place the correct suggestion (*geo:statePopulation*) at the 14th place. This is due to no significant similarity between *people* , and *state population*, at least according to our initial ranking algorithm (Section 9.1.5).

Figure 9.13 shows the values of initial states, reinforcement received after the user selecting *geo:statePopulation*, and finally the rankings after recalculation[15].

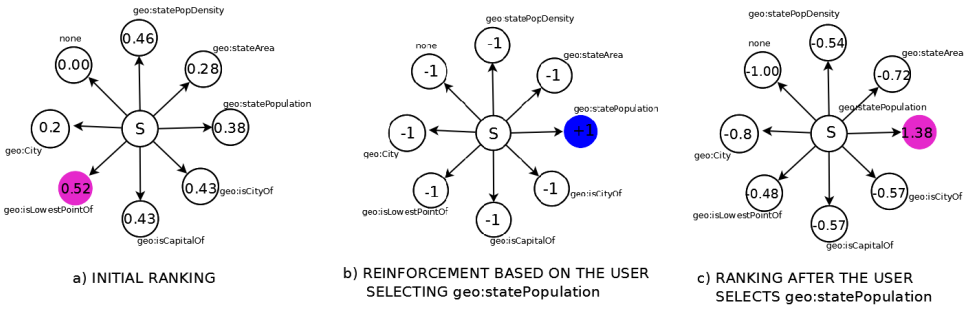

Figure 9.13: Mapping *how many people* to *geo:statePopulation* in the ontology

9.5.4 Generalisation of the Learning Model

We use the ontology as the source for designing the generic learning model. When an OC is related to another concept with a *rdfs:subClassOf* relation, that concept is used to learn the model. For example, if the features are extracted for the OC of type class – *geo:Capital*, the same features would be applicable for the OC *geo:City*, because *geo:Capital rdfs:subClassOf geo:City*.

[15]For the sake of clarity, we only show a subset of generated suggestions in Figure 9.13.

In addition, we do not update our learning model per question, but per combination of a POC and the closest OC. We also preserve a function over the selected suggestion such as minimum, maximum, or sum (applicable to datatype property values). This way we may extract several learning rules from a single question, and if the same combination of a POC and an OC appears in another question, we can reuse it. Table 9.5.4 shows several sample questions and derived features which are used to learn the model.

POC	**context**	**function**	**correct rank**
what is the smallest city in the us?			
smallest	geo:City	min	geo:cityPopulation
What is the population of tempe arizona?			
population	geo:City	–	geo:cityPopulation
what is the population of the capital of the smallest state?			
population	geo:Capital	–	geo:cityPopulation
smallest	geo:State	min	geo:statePopulation

Table 9.6: Features used for learning the model

Figure 9.14 demonstrates how our learning algorithm works for query *What is the highest point of the state with the largest area?*. There is only one token (*state*) annotated as referring to an OC, whereas there are three POCs. We start with the last POC *largest area*. Suggestions are generated based on the closest OC which is *geo:State* in this case (see Figure 9.15). As one of the options will be a datatype property referring to *geo:stateArea*, the user is very likely to select this from the list of available options. We would then resolve that *area* refers to *geo:stateArea*, whereas the *largest* is still a candidate for generating further suggestions and asking the user whether this relates to an operation related to *geo:stateArea* such as finding the minimum or the maximum value of it. With time, the system will learn to associate *largest area* with the maximum function of *geo:stateArea*, even if this combination appears in the context which is not the same but similar.

We then skip the next POC (*state*) as it overlaps with the ontology concept *geo:State*. The last POC *the lowest point* is then used to generate suggestions. In this step we use the closest OC, which is again, *geo:State*. There will be several suggestions and the user is very likely so select a property named *geo:isLowestPointOf*, although this one will be ranked on the third

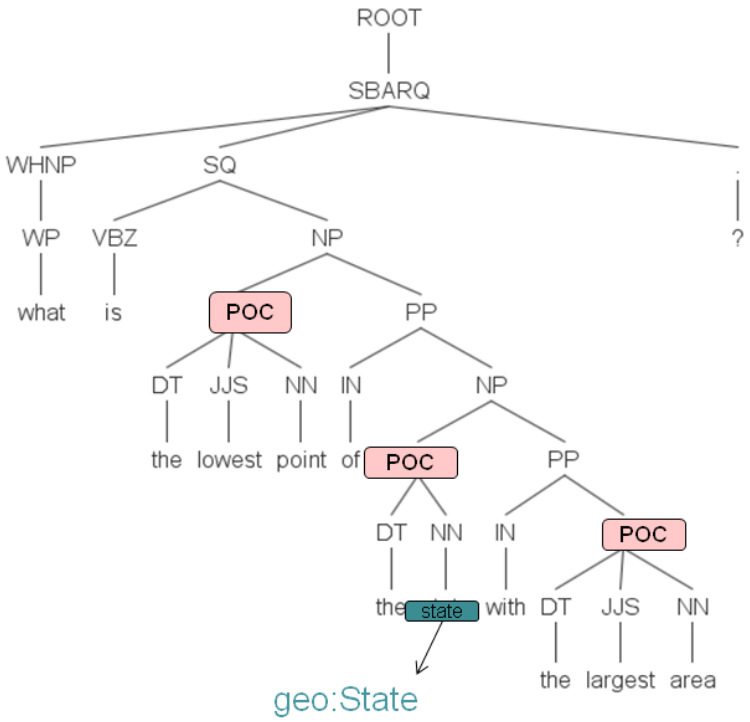

Figure 9.14: Validation of POCs through the user interaction: preparing for the dialog

place. Note that although *lowest* is the superlative it will not be further used to generate suggestions for the user as *geo:isLowestPointOf* is an object property. However, for the next user, the system will learn to rank *isLowestPointOf* first.

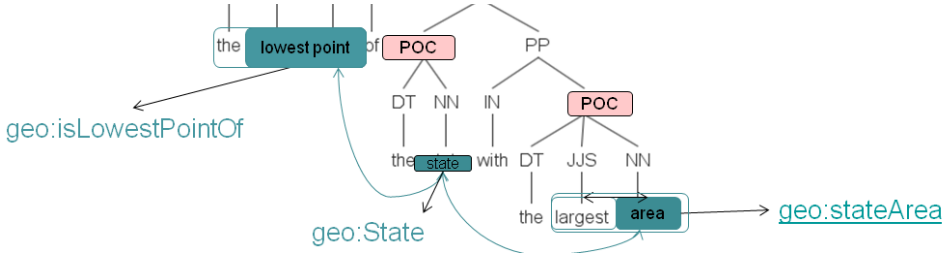

Figure 9.15: Validation of POCs through the user interaction: the user is in control

9.6 Portability

FREyA is a portable NLI in a sense that it can be easily ported to work with a different ontology, or a set of ontologies which are available either from the Web or on the local file system. It can either preload the ontologies into its own repository which is based on OWLIM[16], or connect to the already existing repository, which can be local or remote.

In order to perform the ontology-based lookup at the query processing time, FREyA requires extracting the ontology lexicalisations, processing them, and adding them to an index. The extraction of ontology lexicalisations requires reading the whole repository through the set of SPARQL queries. The number of SPARQL queries depends on the size of the schema which describes the dataset.

FREyA does not require a strict adherence to syntax, however, it relies on the ontology-based lookup. Trying a sample query *What is the capital of France?* with FREyA initialised with a superset of DBpedia (accessed through `http://www.factforge.net/sparql` repository) revealed that according to the extracted lexicon, **each word in the question refers to at least one Ontology Concept**. If there were no automatic disambiguation nor heavy grammar analysis, the system would model the first dialog asking *What is 'what'? Is 'what' related to: LIST OF URIs.* A similar dialog would be modelled for 'is'; the system would ask the user whether *is* is related to: *be*, *was*, or *were*. And so on, for each word in the question.

These situations must be resolved either by performing automatic disambiguation (which might be expensive for datasets with billions of triples) or by constraining the supported language and allowing the user to type in only a limited set of question types. In case of the system failing to automatically interpret the question, it can seek help from the user as is the case with FREyA. The fine balance is in the combination of these approaches: disambiguate as much as possible and use the ranking mechanisms (e.g., those that exist in FREyA, or any other methods for effective ranking such as in Lopez et al. [2009a]), and correct them if necessary using the interactive features of FREyA.

[16] `http://ontotext.com/owlim`

Expressiveness of the language supported by FREyA has a trade-off, which is especially highlighted when coupled with the heterogeneity which comes from the Linked Open Data. Indeed, when trying any dataset with FREyA for the first time, it is advisable to use the dialog as much as possible in order to check the system interpretations and correct them if necessary. In that regard, there are several *modes* that can be used:

- **Automatic mode** The dialog in FREyA is designed in a way that it can be configured based on the level of confidence. The maximum confidence level allows using FREyA in the *automatic mode* meaning that the system will generate the answer by simulating selection of the best ranked options. This mode is used when the confidence is high that the ranking is effective, or the system has been trained enough and can make the decisions on its own.
- **ForceDialog mode** operates on two levels:
    1. *Ignoring the system's attempt to perform the mapping* by adding a 'None element'. This element is used to ignore the system's attempt to map a question term to an OC. That is, the system would assume that the question term in the dialog should not be mapped to any suggested OCs, and therefore the system would learn by the time that this POC/OC is either: 1) incorrectly identified, or 2) cannot be mapped to any OC as the ontology does not contain relevant knowledge. As previously discussed, this option is not likely to provide much benefit to the end-users, but it is intended to identify flaws in the system and encourage improvements.
    2. *Extending the disambiguation dialog* This option extends the disambiguation dialog by adding more suggestions, in addition to the OCs identified through the *Ontology-based Lookup*. This option is important to be used when the knowledge base has a large number of names (e.g., MusicBrainz) so that any question would be a rich set of Ontology Concepts, while the underlying grammar would be somewhat ignored. For example, in question *Which members of the Beatles are dead?* due to a huge number of string literals

dead appearing in the ontology, this element would be annotated to refer to several OCs (such as instances of *rdf:type mm:Album*) while indeed it needs to be mapped to the property *endDate*.

9.7 Evaluation

While QuestIO and the feedback in FREyA are evaluated with users, in order to compare FREyA with the state of the art, we evaluated the system using 250 questions from the Mooney GeoQuery dataset. Although the ontology contains rather small portion of the knowledge about the United States geography, the questions are quite complex and the system must have a good understanding of the semantic meaning in order to correctly answer them. In addition, other NLIs have been evaluated using this dataset, and therefore, by conducting the evaluation with the same ontology and the same set of questions, we can compare our performance with the state of the art. We evaluate *correctness* (Section 9.7.1), *learning* (Section 9.7.2), *ranked suggestions* (Section 9.7.3), and *answer type identification* (Section 9.7.4).

Further on, to demonstrate the *portability* and the *suitability* of FREyA to be used in the real scenario for querying the Linked Data, we present experiments in Section 9.7.5.

9.7.1 Correctness

We report correctness of FREyA in terms of precision and recall (see Section 4.2 for the definition of precision and recall).

Recall and precision values are equal, reaching 94.4%. This is due to FREyA always returning an answer, although partial or incorrect. 34 questions were answered correctly without requiring any dialog with the user, while remaining 202 required at most 4 dialogs in order to correctly return the answer (see Figure 9.16). The system failed to answer 14 questions (5.6%), 5 out of which are not supported by the system, such as negation or comparison e.g. *which states have points higher than the highest point in colorado?*. The remaining 9 were incorrectly interpretted.

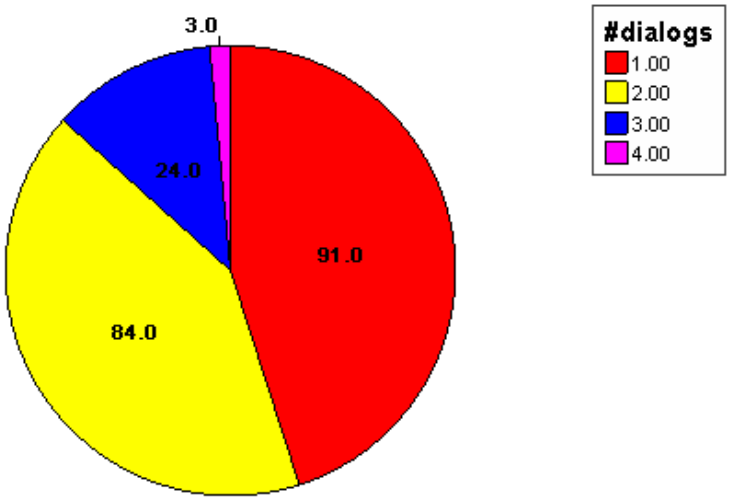

Figure 9.16: The distribution of the number of dialogs for 202 correctly answered questions

Although FREyA required quite a significant user input, its performance compares favourably to other similar systems. PANTO [Wang et al., 2007] is a similar system which was evaluated with the Mooney geography dataset of 877 questions (they removed duplicates from the original set of 879). They reported precision and recall of 88.05% and 85.86% respectively. NLP-Reduce [Kaufmann et al., 2007] was evaluated with the original dataset, reporting 70.7% precision and 76.4% recall. Kaufmann et al. [2006] selected 215 questions which syntactically represent the original set of 879 queries. They reported the evaluation results over this subset for their system Querix with 86.08% precision and 87.11% recall. Our 250 questions are a superset of these 215.

In order to test the statistical significance of our results, we calculated 95% confidence interval for the precision and recall. As we only have one test

set, we used the bootstrapping sampling technique[17]. The method had also been used in the CoNLL-03 competition (see Sang and Meulder [2003] and also Li et al. [2009]).

The 95% confidence interval with 1000 samplings range from 91.6% to 97.2%. As the lower range is still higher than the best previously evaluated system (88.05% for recall of PANTO [Wang et al., 2007], and 87.11% precision of Querix [Kaufmann et al., 2006]), we conclude that precision and recall values obtained with FREyA were significantly better (p=0.05) than the precision and recall of other systems trialed with the same dataset. It should be noted, however, that this high performance of FREyA engaged the user into the dialog. Querix also relies on dialogs, while PANTO answers questions automatically.

What makes FREyA outstanding is the possibility to put the user in control and improve the performance incrementally with each user's new question, by boosting the rankings through learning from the user's clicks. In the next section, we describe the evaluation of our learning mechanism and its effect on performance.

9.7.2 Learning

We evaluate our learning algorithm using *cross-validation* on 202 questions which are a subset of the above 250 – those that can be answered correctly and which required at least one dialog.

Cross-Validation is a statistical method used to evaluate and compare learning algorithms by dividing data into two segments: one used to *train* a model and the other used to *test* it. In typical cross-validation, the training and validation sets must cross-over in successive rounds such that each data point is tested against [Refaeilzadeh et al., 2009]. The basic form of cross-validation is k-fold cross-validation, where the data is first partitioned into k

[17]Given the set of the 250 test samples that we used for computing the precision and recall: T = s1, s2, s3, ..., s250, and obtained P = 94.4, we did one sampling 1000 times: in the step i, get a set Ti by randomly sampling the set T with replacement. The set Ti has 250 samples but some samples may be the same. Then we compute the precision on Ti and get Pi. When we calculated 1000 precisions P1, P2, ..., P1000, we sort them from low to high and get P1', P2', ..., P1000'. Then the 95% confidence interval will be [P25', P975'].

equally (or nearly equally) sized folds. Subsequently k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for testing while the remaining (k-1) folds are used for training.

We performed 10-fold evaluation using the subset of the Mooney GeoQuery questions which could be correctly answered:

- 5 questions were not supported by the system, and they have been removed due to no possibility to map them to the relevant ontology concepts and get the correct answer,
- 9 questions were misinterpreted by the system,
- 34 could be answered automatically so they were removed.

This resulted in 202 questions requiring 343 dialogs in total. In 10 iterations, 181/182 questions were used for training the model, while the remaining 21/20 were used for testing it. Before executing the test, we have generated a *gold standard* in two steps:

- We ran FREyA in the automatic mode where for any required dialog the system would choose the first available option, save the learning items and carry forward to the next question.
- We then manually examined the output and corrected invalid entries. If we had to change the entries we have marked those as incorrect. This enabled us to measure the performance of the baseline system.

The goal of this evaluation was to test whether our learning algorithm can improve the performance of the system. In order to assess this, we compare the precision of the trained system with the performance of the baseline. The results are shown in Table 9.7 and also in Figure 9.17

The average precision for the system trained with 9/10 of questions was 0.48, which is 0.2324 higher than the baseline. While this is a good improvement of the baseline model, the performance is not outstanding. Looking into the questions which could not be answered using our trained system, the reasons are:

Fold	0	1	2	3	4	5	6	7	8	9	Avg.
Baseline	.3	.15	.2	.25	.24	.3	.3	.35	.15	.19	0.2476
Learning	.65	.4	.65	.4	.24	.55	.5	.6	.35	.48	0.48

Table 9.7: Precision for 250 questions evaluated using 10-fold cross-validation

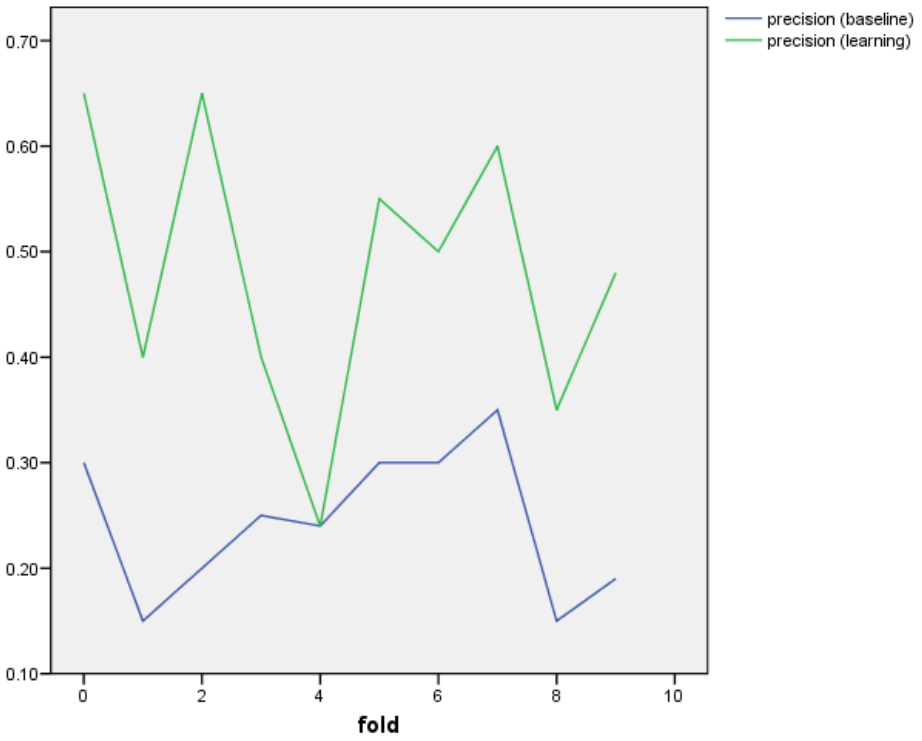

Figure 9.17: Precision for the trained vs. baseline system using 10-fold cross-validation

Ambiguity 30 questions were not correctly answered due to ambiguity. The advantage of our learning model is its simplicity: it is based on a very few features ensuring that questions with similar word pairs would benefit from the training with *similar* and not necessarily *same* questions. However, this is at the same time a drawback as it can introduce ambiguities. For example, if the system learns from *what is the highest point of nebraska?* that *point* refers to *geo:HiPoint*, whenever it appears in the context of *geo:Country*, then, for similar albeit drastically different questions, the system would use the knowledge which might be wrong. Namely, for the question *what point is the lowest in california?* the system would find the previously learned mapping and it will associate *point* with *geo:HiPoint* whereas the correct mapping is the *geo:LoPoint*. This indicates that we should *extend the context* of our learning model to consider the whole phrase in which the 'unknown' term appeared, so that for the mentioned example whenever *point* appears in the context of *geo:Country*

- AND *highest*, map it to *geo:HiPoint*.
- AND *lowest*, map it to *geo:LoPoint*.

Sparsity 65 questions contained a learning item which was seen only once across all questions. For example, the only questions which included *greatest* were: *what state has the greatest population density?* and *what state has the greatest population?*. However, while in the former case the *greatest* is paired with *geo:statePopDensity* in the latter, it is paired with *geo:statePopulation*. Therefore, these two questions cannot benefit from each other. This suggests a possible improvement of our learning model. Namely, instead of using the exact words to match against our learning model we could make it more robust by matching against all the synonyms of *greatest*.

While the performance of the baseline is quite low, we should note here that this figure does not take into consideration the cases when an 'unknown' or 'ambiguous' term can be mapped to more than one ontology concept. In addition, the question is marked as correct if all dialogs have the correct ranking placed first. However, for some cases it is very difficult to judge

automatically which suggestion to place first. It is very likely that different users would select different suggestions for the questions phrased the same way. This emphasises the importance of dialog when modelling NLI systems. To assess this we evaluate the ranking of suggestions in isolation.

9.7.3 Ranked Suggestions

We use *Mean Reciprocal Rank (MRR)* to report the performance of our ranking algorithm. MRR is a statistic for evaluating any process that produces a list of possible responses (suggestions in our case) to a query, ordered by probability of correctness. The *reciprocal rank* of a suggestion is the multiplicative inverse of the correct rank. The *mean reciprocal rank* is the average of the reciprocal ranks of results for a sample of queries (see Equation 9.2).

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{Q}\frac{1}{rank_i} \tag{9.2}$$

We manually labelled the correct ranking for suggestions which are generated when running FREyA with above set of 202 questions. This was the gold standard against which our ranking mechanism achieved MRR of 0.76. However, the median and mode were both 1 indicating that majority of rankings were correct. Indeed, as shown in Figure 9.18, in 69.7% of the cases the correct ranking is placed first, while in 87.5% of the cases the correct ranking is among first five.

From the above set of 343 dialogs, we selected 103 randomly, and then ran our initial ranking algorithm and compared results with manually labelled gold standard. MRR was 0.72. Table 9.8 shows the distribution of the rankings.

We then grouped 103 dialogs by OC, and then randomly chose training and evaluation sets from each group. We repeated this two times. Table 9.9 shows the structure of the dataset grouped by OC for both iterations. Note that these two iterations are independent - both are performed starting with an untrained system.

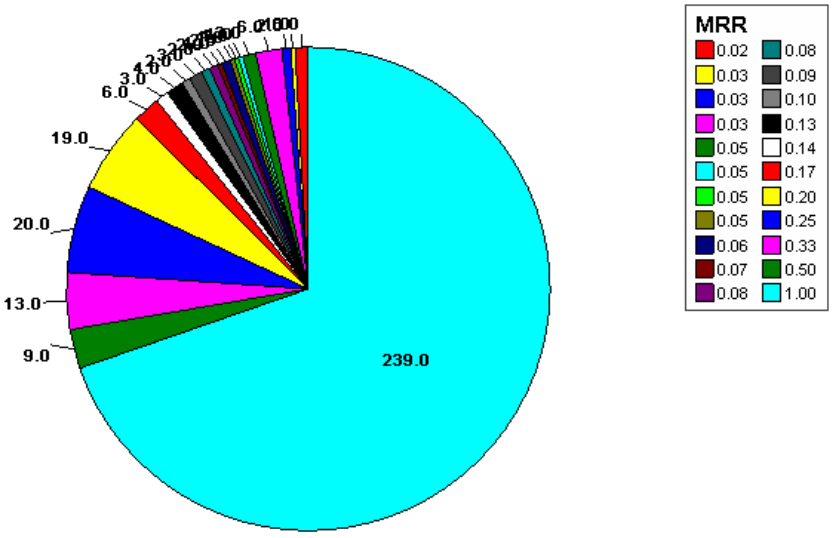

Figure 9.18: The distribution of the MRR for 343 dialogs

Table 9.8: Evaluation with 103 dialogs from the Mooney geography dataset

Correct rank	**Number of questions**
1	64 (62.13%)
2 or 3	22 (21.36%)
4 or more	17 (16.5%)

Table 9.9: The distribution of the training and evaluation sets for 103 dialogs

OC	Iteration 1		Iteration 2	
	Training	**Evaluation**	**Training**	**Evaluation**
geo:State	26	19	19	26
geo:City/Capital	20	19	19	20
geo:River	12	6	9	9
geo:Mountain	1	0	0	1
total	59	44	47	56

After training the model with 59 dialogs from the iteration 1, MRR for the evaluation set (44 of them) reached 0.98. Overall MRR (for all 103 dialogs) increased from 0.72 to 0.77. After training the model with 47 items during the iteration 2, overall MRR increased to 0.79. Average MRR after running these two experiments was 0.78, which shows the increase of 0.06 in comparison to MRR of the initial rankings. Therefore, we conclude that for the selection of 103 dialogs from the Mooney GeoQuery dataset, our learning algorithm improved our initial ranking by 6%.

9.7.4 Answer Type

First we have experimented with *QA Detector* in isolation, and calculated to which extent it was possible to identify the question focus/ATI using the algorithm described in Section 9.2.1. This shows the correctness of *QA Detector* irrespective of whether the answer type was correctly found in the subsequent steps, or not.

The second experiment evaluates the correctness of the consolidation algorithm from Section 9.2.3 used to identify the answer type.

QA Detector algorithm

We first manually labeled the correct focus/ATI for all 250 questions. This was the gold standard for this step.

Out of 250 questions, the ATI was correctly identified for 45 of them (questions starting with *how big*, *how large*, *where* and the like). The results for the remaining 205 were as follows (see Figure 9.19):

- *Correct*: For 174 out of 205 (84.88%), the focus was identified correctly.

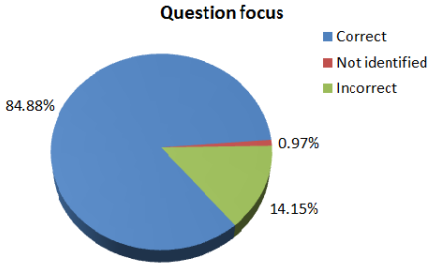

Figure 9.19: Identification of the question focus: results

- *Not found*: For 2 questions (0.97%), our algorithm could not identify neither the focus nor the ATI. This is due to the complex structure in which the prepreterminals were not tagged as noun phrases or nouns. For example, in the questions *what is the most populated state bordering Oklahoma?* or *what is the most populous state?*, the correct focus is *the most populous state* for both questions, however, this noun phrase is not prepreterminal due to *most populous* being tagged as Adjective Phrase – ADJP (see Figure 9.20).

- *Incorrect*: Remaining 29 (14.15%) questions had incorrectly identified focus and errors could be represented through the following patterns:

  - **Negation**: one sentence with negation had been parsed incorrectly: in *What rivers do not run through Tennessee?*, the parser tagged *rivers* as RB (adverb), while it should be Noun. It is interesting that the same sentence with omitted *not*, is parsed correctly (i.e. *rivers* is tagged as noun (NNS)).
  - **What NP**: such as in *What capital is the largest in the US?* and *What city has the most people?*; while the parser correctly identified the span which contains the focus (*What capital* and *What city* respectively), the head finder identified the head of both phrases to be *What*.
  - **Give me NP**: the personal pronoun *me* was tagged as PRP which is correct. However, as it was identified as a part of the prepreterminal noun phrase, our algorithm wrongly identified it as the focus. For example, in *Give me the cities in Virginia?*

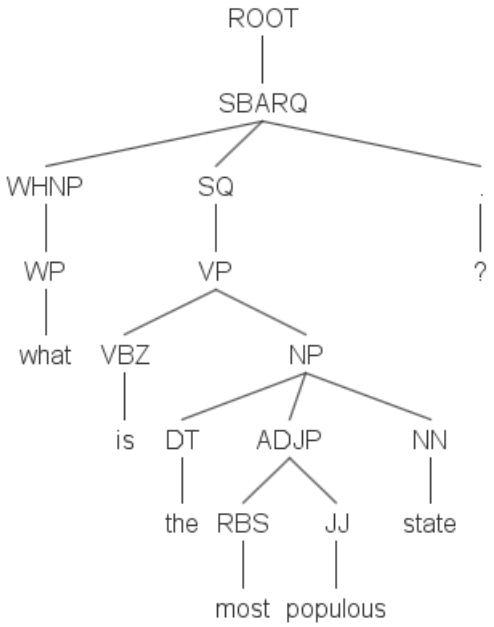

Figure 9.20: Failing to identify the answer type: no identified prepreterminals are nouns/noun phrases

or *Give me the largest state?*, correct focus is *the cities* and *the largest state* respectively, and not the possessive pronoun *me* as identified by our algorithm.

– **State vs. Borders**: When occurring together, these two words have been tagged incorrectly by the parser. For instance, in *Which states borders Arkansas?* state is identified as VBZ (Verb, present tense, 3rd person singular) while *borders Arkansas* is NP consisting of NN (borders) and NNS (Arkansas). Therefore, the focus is identified to be *border Arkansas*, which is incorrect.

Consolidation

Further on, we evaluated the consolidation algorithm in order to identify

- The number of questions for which the focus could be used to successfully identify the answer type with and without engaging the user.
- The number of questions for which the answer type was identified

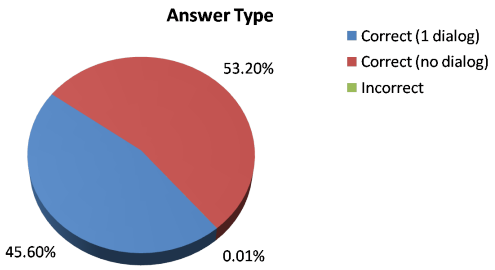

Figure 9.21: Correctness of the identification of the answer type

Focus	**Answer type**		
	Required 1 dialog with the user	**Automatically consolidated**	**Incorrectly consolidated**
correct (174)	65	106	3
not found (2)	0	2	0
incorrect (29)	4	25	0
ATI found (45)	45	0	0
total (250)	114	133	3

Table 9.10: Results of identifying the answer type using the consolidation algorithm

correctly although the focus was incorrectly identified in the previous step.

Figure 9.21 shows the percentage of the correctly identified answer type and also the distribution of the results based on whether this identification required the user to provide input (for 45.6% of the cases) or not (for 53.2% of the cases).

Table 9.10 shows more details. All questions which had the focus identified incorrectly in the previous step, had the answer type identified correctly after the consolidation phase. However, 4 out of 29 questions involved the user into the dialog in order to place this mapping.

With regard to 174 questions for which the correct focus was found in the previous step, 106 (60.92%) could be mapped to an ontology concept automatically. 65 (37.36%) questions required a dialog with the user in order to map the answer type correctly, 6 out of which did not have any FOC

identified, and were answered by modeling suggestions as explained in Section 9.2.4.

3 (1.72%) questions had wrongly identified answer type after the consolidation. This was the case for *compound-nominal expressions* which contain several nouns, each of which being annotated as referring to an ontology concept. For example, the phrase *state capital* refers to *geo:Capital* in *what is the largest state capital in population?*. However, both *state* and *capital* are annotated as refering to different ontology concepts (*geo:State* and *geo:Capital*), and our algorithm would give priority to the *state* as the first ontology concept in the question. In future, we will consider giving priority to the ontology concepts which are the exact matches with the identified head of the focus, such as in this case.

While identification of the answer type through the engagement of the user can be seen as cognitive overhead, our intention is to see whether our learning mechanism can reduce this overhead by the time. In addition, by engaging the user into dialog, he has the full control of the system interpretations and therefore can *train* it towards a very good performance even in cases when the ontology (or a set ontologies which are being queried) does not have human understandable lexicalisations.

9.7.5 Querying Linked Data with FREyA

In this section we report the performance of FREyA using the MusicBrainz and DBpedia datasets provided within the 1st Workshop on Question Answering over Linked Data (QALD-1) challenge[18].

We preloaded the data into our local repository (BigOWLIM 3.4, on the top of Sesame[19]) and then initialised the system using the SPARQL queries. Another option was to connect to the SPARQL endpoint provided by the QALD-1 challenge organisers[20], however, this was a difficult path due to the limited server timeout, which was not sufficient for executing all required queries.

[18] `http://www.sc.cit-ec.uni-bielefeld.de/qald-1`
[19] `http://openrdf.org`
[20] `http://greententacle.techfak.uni-bielefeld.de:5171/sparql`

Generating the index which is required for performing the ontology-based lookup is a mandatory step but is done once per dataset, although it might be time-consuming depending on the size of the data. Table 9.11 shows the statistics of loading the two datasets into the OWLIM repository and generating the index.

	MusicBrainz	**DBpedia**
#explicit statements	14 926 841	328 318 709
#statements	19 202 664	372 110 845
#entities	5 490 237	96 515 478
#SPARQL queries executed	30	361623
initialisation time	1380s (0.38h)	182779s (50.77h)

Table 9.11: Initialisation of the system and the size of datasets

After the index is generated, it is used at the query execution time. We first ran 50 training queries for both datasets and measured the overall precision, recall and f-measure. We then repeat the process with 50 test questions for each dataset. This experiment was conducted with FREyA in the *forceDialog* mode. Results are shown in Table 9.12[21]. MusicBrainz was a challenging dataset due to the existence of properties *beginDate* and *endDate*, which do not have any domain defined, and moreover, which are used extensively throughout the ontology and especially in the combination with the blank nodes. Several failures were due to the malfunction of the *Triple Generator* when these two properties were mapped to the wrong entity. For example, *Since when is Tom Araya a member of Slayer?* resulted in generating the following mappings:

```
Since when >> beginDate
Tom Araya >> Tom Araya (Artist)
a member >> memberOfBand
of >> toArtist
Slayer >> Slayer (Artist)
```

Our Triple Generator then followed by generating:

[21] Demos showing FREyA answering the QALD-1 challenge questions are available from `http://gate.ac.uk/sale/dd/`.

	MB		**DBpedia**	
	Training	*Testing*	*Training*	*Testing*
Precision	0.75/0.77	0.66/0.8	0.74/0.85	0.49/0.63
Recall	0.66/0.68	0.54/0.66	0.58/0.66	0.42/0.54
F-measure	0.70/0.74	0.59/0.71	0.67/0.72	0.45/0.58
# NS questions	6	9	11	7
# RF questions	1	6	4	6
avg.#dialogs per question	3.4	3.65	2.7	2.85
# PC questions	1	1	3	12

Table 9.12: Performance of FREyA using QALD-1 datasets: the left figures exclude while the right figures include the questions correctly answered after reformulation (RF questions). The number of dialogs per question includes only the questions that could be answered *correctly* with or without reformulation. NS (*Not supported*) questions include those that could not be correctly mapped to the correct SPARQL query due to the limited language coverage. For example, questions requiring negation, temporal reasoning such as *Which bands were founded in 2010?* or quantification such as in *Which locations have more than two caves?*. PC (partially correct) questions are those that have returned a portion or a superset of the correct results.

```
?joker1 - beginDate - Tom Araya (Artist)
Tom Araya (Artist) - member of band - ?joker2
?joker2 - toArtist - Slayer (Artist)
```

and the corresponding SPARQL was:

```
prefix mm: <http://musicbrainz.org/mm/mm-2.1#>
prefix mma: <http://musicbrainz.org/mm-2.1/artist/>
SELECT DISTINCT ?firstJoker0 WHERE {
    {{?i1 ?p0 ?firstJoker0} UNION { ?firstJoker0 ?p0 ?i1} .
     FILTER (?p0=mm:beginDate) .
    }
    FILTER (?i1=mma:362105d1-8f4f-4ba1-949f-3e70183880b5) .
    {{?classJoker4 ?p2 ?i1} UNION { ?i1 ?p2 ?classJoker4} .
     FILTER (?p2=<http://musicbrainz.org/ar/ar-1.0#memberOfBand>) . }
    {{?i4 ?p3 ?classJoker4} UNION { ?classJoker4 ?p3 ?i4} .
    FILTER (?p3=<http://musicbrainz.org/ar/ar-1.0#toArtist>) . }
```

```
FILTER (?i4=mma:72de5171-38cf-4734-bc8a-6ac374dea523 ||
        ?i4=mma:bdacc37b-8633-4bf8-9dd5-4662ee651aec) . }
```

which resulted in retrieving the birthday of Tom Araya, and not the date when he joined the group which is the correct answer.

Other challenges related to the ontology design in MusicBrainz include existence of the property *trackList* which has a container of type *rdf:Seq* as range. In addition, the statements with *releaseType* property use subclasses of class *Type* and not instances of that class which caused several failures. For example, the question *Who is the creator of the audiobook the Hobbit?* requires retrieving instances with lexicalisation *the Hobbit*, which are at the same time related to the class *TypeAudiobook* using the *releaseType* property, while FREyA expects that they are related using the *rdf:type* relation.

The main challenge with DBpedia was a selection of the property to use, due to the large number of suggestions that have always been present. For example, *Who created English Wikipedia?* could be mapped to *?joker dbp:created dbpedia:English_Wikipedia* while the correct answer is returned only after using *dbo:author* relation, instead of *dbp:created*[22]. In addition, there are many quality issues such as in the question *Who designed the Brooklyn Bridge?* where *designed* was mapped to *dbp:architect* instead of *dbp:designer* which resulted in retrieving `http://dbpedia.org/resource/John_Augustus_Roebling`, while using *dbp:designer* the result is `http://dbpedia.org/page/John_A._Roebling`. However, as no mapping exist between the two URIs, the former URI is not the same as the latter, and hence this is marked as an incorrect answer. Interestingly, the former URL is redirected to the latter, which indicates that the two URIs should also be connected using the property *sameAs* in the dataset.

Another challenge specific to DBpedia was the lack of the domain and range classes for properties. Therefore, some questions could not be correctly mapped to the underlying Ontology Concepts. In some cases, the reformulation of queries could help (such as using *spouse* instead of *married to*). However, reformulation was not always sufficient. For example, in *Which states border Utah?*, *border* needs to be mapped to the eight properties: *dbp:north*,

[22] We use *dbp for* `http://dbpedia.org/property` *and* dbo *for* `http://dbpedia.org/ontology` *namespaces.*

dbp:south, *dbp:east*, *dbp:west*, *dbp:northwest*, *dbp:northeast*, *dbp:southwest*, and *dbp:southeast*. As none of these have any domain or range, they did not appear in the suggestions and hence the only way to answer the question using FREyA is to ask eight questions such as *Which states are north of Utah?*, *Which states are south of Utah*, and so on for each property. It is interesting to observe that 12 incorrectly answered questions using the DBpedia test questions were indeed partially correct. The correct mappings could only be placed if we were more familiar with the knowledge structure inherent in the dataset. This also explains the difference in the performance of FREyA using the training and the testing set of DBpedia.

Failures that were common for both datasets are related to the equal treatment of the datatype property values. For example, the question *How many jazz compilations are there?* failed to be answered correctly due to FREyA finding all compilations that had the user defined tag 'jazz' which is case insensitive (using FILTER REGEX(str(?var), "`^jazz$`","i"). Therefore, it included also 'Jazz' which lead to the incorrect answer. On the other hand, some entries were missed when the fuzzy matching was necessary such as in *Which companies are in the computer software industry?* that requires finding not only companies with the property *industry* 'computer software' but also 'computer hardware, software', 'computer software and engineering', and the like. At the moment, the datatype property values in FREyA are supported by including the exact match (case insensitive) only. In future, we might extend our approach to support more sophisticated treatment of strings so that the treatment differs depending on the context.

Several reformulations for both datasets resulted in a significant increase of the precision and recall, e.g. adding quotes such as in *Which artists performed the song "Over the Rainbow?"*. Without quotes, *Over* was parsed as a preposition, and the whole question failed to be answered, while with quotes this was a part of the Noun Phrase which lead to the correctly answered question.

Learning To measure the effect of the learning mechanism, we run the experiment in two iterations: we first answered 50 testing questions using an empty learning model and then using the system trained with 50 training questions. Results are shown in Table 9.13. The learning mechanism im-

	MusicBrainz		**DBpedia**	
	untrained	*trained*	*untrained*	*trained*
MRR	0.63	0.68	0.52	0.54

Table 9.13: Mean Reciprocal Rank for the testing set with and without learning

proved the overall ranking of suggestions for 0.05 for MusicBrainz, and only 0.02 for DBpedia. The reason is the size of the datasets and the relatively small number of the training questions. However, improvement of 0.02 is still an achievement considering that DBpedia has almost 100 million entities.

Execution time for queries that could be answered correctly fluctuates based on the complexity of questions (e.g. number of the required dialogs). This is due to our on fly mechanism for finding suggestions which requires executing a large number of SPARQL queries in order to generate a dialog. Long execution is also affected by the complexity of the final generated SPARQL which is used to retrieve the answer. For example, queries which include FILTER statements over literal strings such as FILTER (regex(?var, "`^jazz$`", "i")) currently can take more than ten minutes to be executed[23]. The size of the dataset influences the execution time as well. For MusicBrainz, the average time per dialog was in the range from 0.073 to 11.4 seconds, or 8.5 seconds on average per question. For DBpedia, the execution time was much longer: from 5 to 232 seconds per dialog, and 36 seconds on average per question. This is quite slow, however, it can be optimised (e.g. by using the caching mechanisms for suggestions).

The evaluation using the DBpedia and MusicBrainz testing datasets leads to the f-measure of 0.58 and 0.71 respectively which favourably compares to the other tested systems that participated in the QALD-1 challenge (PowerAqua 0.5 using DBpedia, SWIP 0.66 using MusicBrainz). More importantly, FREyA was the only system that is tested with both MusicBrainz and DBpedia datasets which demonstrates portability. The learning mechanism improved the results for 5% and 2% for the MusicBrainz and DBpedia datasets respectively.

[23]Experiments are run using the CentOS 5.2 Linux virtual machine running on a AMD Opteron 2431 2.40GHz CPU with 2 cores and 20G RAM.

9.8 Summary

In this chapter we discussed the final design, implementation and evaluation of FREyA, an interactive Natural Language Interface that combines the *syntactic parsing* with the *ontology-based lookup* in order to interpret a Natural Language query or its fragment. The query is mapped into the formal query language SPARQL in order to find the correct answer. If it fails to perform the mappings automatically, or in case of any ambiguities, FREyA generates a *dialog* and involves the *user* into loop. The user's selection is saved and used for *training* the system so that it improves its performance over time.

In contrast to QuestIO, which is a *closed-and-single* domain, the scope in FREyA is extended towards *multiple domains*, or rather to any semantic repository that may contain a large number of ontologies and knowledge bases which may originate from different sources. However, FREyA needs to generate the index offline, in order to perform the *ontology-based lookup* at the query analysis time. This process requires executing a number of SPARQL queries and can be time-consuming however, it is performed only once.

We discussed earlier (see the end of Section 7.1) that QuestIO is not suitable to be used even in narrow domains that contain a large amount of identical names, due to the *data ambiguity* problem. In FREyA this problem is reduced by performing a deep grammar analysis of the question. This helps in solving *data ambiguities* caused by either diverse data sources or repositories where the ABox is much larger than the TBox. However, the supported language remains flexible, and both grammatically correct questions, fragments or ill-formed queries are supported. The flexibility has a trade-off (discussed previously in Section 7.6) related to the fact that it is not trivial for the user to translate his information need into the question. Hence, we looked at combining usability enhancement methods *feedback* and *clarification dialogs* in order to improve precision by asking the user to disambiguate, but also in order to extend the system's vocabulary (derived from the semantic resources and enriched by WordNet) from that of the user.

The *vocabulary extension* reduces the **lexical failures** discussed in Section 7.6. Moreover, the lexical failures are avoided due to our *ranking algo-*

rithm which relies on Soundex – the state of the art algorithm that assigns a very high similarity to the two words which are spelled differently but pronounced similarly. Soundex is combined with Monge Elkan string similarity algorithm which assigns a high similarity to the two words, one of which is contained in the other (e.g. a question term *population* is very similar to the ontology lexicalisation *state population* according to Monge Elkan). Combining the two algorithms gives the possibility to go beyond the existing lexicalisations attached to semantic resources, and "understand" words which are either misspelled or expressed differently in comparison to how they are verbalised in the semantic repository.

In order to avoid the **conceptual failures** discussed in Section 7.6, FREyA can be used in the *forceDialog* mode. This mode means that the dialog will be modelled for each attempt to map a question term into an ontology concept. This is a slight modification of the approach described in the first version of FREyA (Chapter 8), where the conceptual failures were handled by showing the user all query interpretations at the time. Moving from a *query interpretation* towards a *concept-based* one is largely influenced by the feedback from users in the user-centric evaluation described in Section 8.2.

The *concept-based interpretation* discussed in this chapter is guided by the *dialog sequence algorithm*. While other existing approaches start by generating linguistic triples from a question (even if in an iterative fashion) and then attempt to generate ontology triples in a form of `Subject-Predicate-Object`, our approach operates on a *pair* of Ontology Concepts at the time, which can be `Subject-predicate` or `predicate-Object` or `Subject-Object`. In that sense our approach is more flexible as it operates on a unit smaller than a triple.

Question interpretation starts by the *syntactic parsing and analysis* to find the *focus* – a question term or phrase that identifies what the question is about, and then consolidating it with the *ontology-based lookup*. The output of the *consolidation algorithm* is the focus which is mapped to one or several Ontology Concepts. Further on, other *Potential Ontology Concepts* are also mapped to Ontology Concepts using the same *consolidation algorithm*. *Potential Ontology Concepts* are candidate question terms or phrases that are extracted by combining a parsed tree with a set of heuristic rules. The

order in which the *Potential Ontology Concepts* are mapped is controlled by the *dialog sequence algorithm*, which is based on favourising the mapping of those concepts that are closer to the *focus* (as per distance calculated by looking into the parsed tree).

For any ambiguity or uncertainty, the system generates a dialog and the user is involved to choose from a set of available options. If the system is run in the *automatic mode* it will return the answer automatically by simulating selection of the best ranked options. To give an example for *which is the largest lake in California?*, the focus *largest city* will be interpreted first. Indeed, first the algorithm attempts to resolve the head of the focus (city), consolidates it with the *ontology-based lookup* (e.g., ontology class *City*), and then it continues to resolve *largest*. Only after these are interpreted the algorithm will follow to resolve *California*. The *consolidation algorithm* may automatically resolve this mapping based on the exact match between *California* with the existing ontology lexicalisation.

Note that for true ambiguities the *automatic mode* might not be the best choice even in the perfectly trained system. For instance, if somebody asks about *How big is New York state?* we might be unable to decide whether *How big* refers to *state area* or *state population* automatically. In this situation, as the system learns from the users' selections, the *automatic mode* would work in favour of majority of the users. However, if the majority of users refer to *state area* when mentioning size, the minority still have a chance to get the correct answer by using FREyA in the *forceDialog* mode and mapping *big* to *state population*.

In addition, in contrast to the *tree-based feedback representation* described in Chapter 8 we decide to use a *graph-based* one. This is because trees were impractical when tested with different ontologies, as the relations between all nodes could not always be represented clearly using the tree. However, the interactive features remain the same as they were widely accepted by users in the evaluation in Section 8.2.

All algorithms as well as the system as a whole are evaluated using the GeoQuery Mooney dataset for the sake of comparison with other similar systems. MRR for the *initial ranking* using 250 questions from the Mooney GeoQuery yielded 0.76. The *answer type identification algorithm*, which is

fundamental in order to precisely answer the question, correctly returned the answer type for 98.18% of questions, although 37.36% required one dialog with the user. The *learning algorithm* showed an improvement of the baseline model for 23.24%. The overall *precision* and *recall* with this dataset reached 94.4% which is significantly better than other similar systems evaluated using the same dataset.

FREyA also participated in the QALD-1 challenge, where the organisers released two datasets, DBPedia 3.6 and an RDF export of MusicBrainz, which differ not only in size but also in the complexity of the ontology structure. Each dataset was initially released with 50 training questions accompanied with the correct SPARQL queries and answers. Further on, the organisers released a testing set of 50 questions per each dataset and all participants had to provide either the correct answers or the SPARQL queries, or both. The evaluation results can be accessed from `http://www.sc.cit-ec.uni-bielefeld.de/sites/www.sc.cit-ec.uni-bielefeld.de/files/overall.pdf`.

FREyA was the only system among the three participating ones, that provided results for both datasets thus demonstrating *portability*. Moreover, FREyA outperformed the other systems although it was used in the *forceDialog* mode and required quite an engagement of the user. Nevertheless, this demonstrates that the implemented methods and algorithms in FREyA can be a good starting point for a more ambitious goal which is Question-Answering on the open Web. We discuss suitability of FREyA for this task in Chapter 11.

At first sight, the two types of modes described above (*forceDialog* and *automatic modes*), look as a perfect match for the two types of users of FREyA: ideally *application developers* can use the system in the *forceDialog* mode until they are satisfied with the system interpretations of the questions. At that point, the *end-users* can take over the system and use it in the *automatic mode* to ask questions. However, the real scenario might be completely different. The system's mode can be changed easily hence if the user uses FREyA in the *automatic mode* and discovers non-satisfying results, he can immediately switch to the *forceDialog mode* in order to investigate the mappings. His input will then improve the system for the next user.

Easy switching between modes makes FREyA a system that can be used by the *end-users* and *application developers* at the same time. In fact, the border between the customisation of the system performed by *application developers*, and the customised version of the system used by the *end-users* is not strict. Hence, the role of the two types of users is to some extent overlapped, which allows the *end-user* to control the answer to the question or to at least understand how the Natural Language query is mapped to the formal query. This leaves us with the same question that we asked in the previous chapter about the end-users. Who are they? For the current state of the methods and algorithms as they are implemented in this thesis, the end-users probably need not to know about semantic technologies if the system works with narrow domains such as the Mooney GeoQuery. As soon as we move towards a large scale data such as DBPedia, and the datasets which are characterised by a large amount of redundant, duplicate, and often false data, FREyA becomes a tool for semantic web experts who can explore the available knowledge by asking questions and being engaged into the dialog. Using FREyA in the *forceDialog mode* and with the low quality data can be used not only to get familiarised with the dataset, but to discover the existing inconsistencies. It is left for the future work to further develop and test mechanisms that will use FREyA in this kind of scenarios, and also for the scenarios where these large knowledge bases are queried by the *end-users* who are not familiar with semantic technologies at all.

Part IV

Conclusion

Chapter 10

Summary of Findings

This dissertation investigates usability of Natural Language Interfaces to ontologies from the point of view of:

- *application developers* who are customising the system (Chapter 5): the less time they spend customising the system, the more usable it becomes;
- *end-users* who are querying the system (Chapter 6): the higher precision and recall, the more usable the system becomes.

The thesis around which our work is centred is stated in the Chapter 1 as:

> a) *combining syntactic parsing with ontology-based lookup in an interactive process of feedback and query refinement can* ***increase the precision and recall*** *of NLIs to ontologies, while*
>
> b) ***reducing porting and customisation time*** *by shifting some tasks from application developers to end-users*

In what follows we reflect on the status of this hypothesis, in the light of the methods and results which have been tested through building two systems QuestIO and FREyA.

QuestIO (Chapter 7) is concerned with *portability* and *relaxation of the user's queries* – similar to others it does not require any customisation in order to be ported from one ontology to another. QuestIO is one of the first NLIs to ontologies which supports relaxed queries (incomplete or ill-formed) as well as grammatically correct ones. The other similar system is NLP-Reduce, which is developed in Zurich at about the same time. QuestIO's query language has been tested using questions from end users, and the test indicated that QuestIO is as good or better than the AquaLog system [Lopez and Motta, 2004, Lopez et al., 2007] which supports grammatically correct questions only (Section 7.4.1). With regard to portability, QuestIO, as other similar systems, is trialled with ontologies which cover different, but narrow domains. Portability is tested by demonstrating that all that is required to port the system is the URI of the ontology – the system automatically generates the domain-lexicon by reading and processing ontology lexicalisations (Section 7.4.2).

QuestIO then uses this domain-lexicon to perform *ontology-based lookup* over a query and produces all possible *query interpretations*. It also ranks them, and returns the answer based on the first interpretation for which the answer is non-empty. On the positive side, returning the best possible answer automatically is very good for users, especially if there were many interpretations of the query – they do not have to see those that would anyway return no answer. On the negative side, *no answer* does not necessarily imply that the interpretation of the query is wrong – it might be that interpretation is correct, but the answer is *missing*, or, it is *negative*. Another issue which is problematic (see Section 7.4.2) is the ranking of the interpretations which, in case of QuestIO, relies on the ontology structure. QuestIO is built with the assumption that *ontologies are perfect*, namely:

- Each concept/relation in the ontology has the human lexicalisation which describes it – not necessarily a definition, but rather a term which a human would use to refer to this concept/relation.

- Each concept/relation is positioned carefully in the taxonomy: all super-concepts/relations are more generic, and all sub-concepts/relations are more specific.

Therefore, this tool, although portable in the sense that it can be easily plugged in with another ontology, is directly dependent on the quality of data available in the ontologies. At about the time of the QuestIO's development, the initiative has started to encourage people to publish their own data (through Linked Open Data Project), generate their own ontologies from databases (such as DBpedia), and soon after, the large amount of ontologies have been made available and interlinked with each other. None of these were perfect – lexicalisations do exist, but not often they reflect "a term which a human would use to refer to this concept" – this again is especially the case for properties. In addition, the flat structure is dominant. One of the reasons for this is scalability: tractable reasoners do not scale well if the structure of the ontology is complex.

Another observation from the evaluation of QuestIO (Section 7.5) is that encouraging users to use keywords-based queries is at times, misleading. There is a difference in keyword-based searches which are used to answer the question and those used to query the search engines. The intention for the former is to find the *answer* to the question which is interpreted through the set of keywords, while for the latter the aim is to find *relevant documents* which would contain given keywords. However, encouraging users to use *keyword-based* queries, makes them expect the results similar to those which would be found and shown by Google (see Section 7.6).

QuestIO was evaluated with users, where they have been given 3 defined and 1 undefined task. Defined tasks were concrete problems such as *find parameters of POS Tagger*, while the undefined task gave a freedom to users to type in whatever they were interested in, such as *think of any task you would like to perform using the system.* The evaluation results emphasised the importance of usability methods (Chapter 6) which can improve the confidence of the user when querying the system. Namely, as all defined tasks have had the answer, the users did not struggle much to finish them. However, for the undefined task, the performance of the system was poor, and the users disliked it: many errors happened due to users not understanding the way the system is interpreting the query. For example, it was not clear whether the system did not parse the question, or whether it did not have any knowledge about the certain query terms.

Therefore, we have used the experience and the evaluation of QuestIO, to explore these interfaces further, and address some of the problems which arose along the way, considering also the changes which have happened on the Web. Our assumption has now changed with the new challenges which have appeared, the main one being that *ontologies are not perfect*, and that tools which work with them must take this into consideration.

Therefore, our exploration resulted in building a fully interactive system FREyA, which makes certain assumptions but none of them are confirmed automatically - the user has to verify each one of them. In contrast to QuestIO which is fully automatic, FREyA involves the user in the loop. The user is put in focus with our new approach, which is based on the assumption that no ranking will be perfect (because ontologies are not perfect and ranking relies on ontology reasoning).

In this respect, we have first explored *feedback*: showing the user all *query interpretations* as a list of linear combinations of the recognized concepts (see Chapter 8). The user then can choose which of the system's interpretations is correct. On the negative side, if there are too many, and especially if the ranking is not effective, it might be tedious and time-consuming for the user to go through the list of the interpretations in order to find the one which correctly interprets his question. On the positive side, the user is aware of the concepts which are known to the system, and if the concepts are recognized, but no relations between them are found, the user could assume that the reason is *no results* (the answer is negative).

In the evaluation with users (Section 8.2), while the system's interpretation was helpful for complex queries, as they could figure out that they need to reformulate the question, for the queries with the negative results, feedback as such was not perceived as useful by a significant percentage of users. Showing the user that the concepts are recognized, but the answer was not found, was often perceived as the system's failure. In addition, this approach, although with a greater potential than the automatic approach taken by QuestIO, has several difficulties:

- Firstly, interpreting the query as a whole might be problematic: even if only one query term is interpreted incorrectly, the whole interpretation would be invalid.

- Secondly, for several ambiguous terms in the question (which is quite realistic given the current state of the Linked Data), the number of interpretations would be large, producing a huge cognitive overhead on the user.
- Thirdly, the question terms are linked to ontology concepts based on the existing lexicalisations for that concept: if the lexicalisation is missing, the query term would not be understood. This is especially problematic for adjectives, for example. To parse the query *What is the biggest city in Europe?* lexicalisations from the ontology are not enough. Understanding *biggest* in this example means looking into the properties of the word which it modifies (*city*) and then finding the biggest value out of them all.
- Finally, recognising WH-phrases can be of a great importance for understanding the meaning of the question, and therefore should be considered more carefully.

Therefore, in order to address all these observations we have moved towards:

- *concept-based interpretation*: instead of interpreting the query as a whole, and showing all query interpretations to the user, each query term is interpreted separately, and the user is engaged into the dialog to resolve ambiguities on the concept level, if necessary;
- *enriching the domain-lexicon* by integrating the vocabulary of the end-users: if an unknown term appears in the question, we model the dialog and ask the user to attach the meaning, by choosing one of the listed ontology concepts. These ontology concepts are found based on the ontology knowledge, and take the *context* in which the term appears into account.

The user's input is saved and used to update the *learning model* which is used to train the system towards better performance. The learning model is *context-based*, and uses ontology reasoning in order to generalise itself whenever feasible.

The aim of this new approach (presented in Chapter 9) is to interpret a user's query in one unambiguous way, by solving the ambiguities through

the dialog, but at the same time, understanding the user's language and trying to map it to the logical form, through considering the context. For example, *How big* can be mapped to different ontology concepts depending on the context, where context is identified by the ontology concepts with which it appeared in the question. If *How big* appears in the context of a *state*, it could be mapped to the *state population*, while when appearing with a *city*, the same phrase could be mapped to the *city population*. However, if we have already *learned* that *How big is a city...* can be mapped to *city population - city*, the same rule would enable us to conclude that *How big is a capital...* could be mapped to *city population - capital*, because of the existing relation in the ontology: *capital - rdfs:subClassOf - city*.

Domain-independent words such as WH-phrases (such as *Where*), and especially those which contain adverbs (such as *How big*) or adjectives (such as *largest city*) can be crucial for understanding the question, and also they can modify the meaning of the question terms. However, in different domains, and in different context, these words have different meanings. *What is the largest city in California*, and *What is the largest lake in California* require mapping *largest* to two different properties, namely *city population* and *lake area*, respectively. Machine Learning approaches suggested by Tang and Mooney [2001] and Wong and Mooney [2006] solve this by labelling the questions and applying an Inductive Learning Programming approach, which, for known sets of questions, and for small ontologies, can work quite well. However, for the larger repositories, and real world applications such as Linked Data for example, we might not know the structure of the ontology in advance. Therefore, it might be very hard if not impossible to label sets of questions and map them to certain ontology concepts in the vast collection, which must be browsed or queried in order to be understood. In addition, for unseen questions, this approach would not work well.

With our approach, this problem is addressed by modelling the dialog, and learning from the user's selections, therefore potentially improving the performance of the system with each user posing a question. The dialog is modelled based on the combination of *syntactic parsing and ontology-based lookup*.

Our proposed methods balance between heavy customisation (which is usually required by application developers to port an NLI system to a different domain), and the end users who need to explore the available knowledge without being constrained by the query language. Our evaluation with the Mooney GeoQuery dataset, shows that FREyA, with precision and recall reaching 94.4%, outperforms other similar systems (see Section 9.7.1). This satisfies our hypothesis. What contributes to its overall performance is:

- *Initial ranking.* Although the user is in focus and has a large influence on the ranking of suggestions which appear in dialogs, initial ranking is very important in order to reduce the cognitive load on users. We have implemented an algorithm which combines *string similarity* with *synonym detection* (see Section 9.1.5), and the evaluation of this algorithm reaches the MRR of 0.76 (see Section 9.7.3).

- *Identification of the answer type dynamically.* This algorithm combines syntactic parsing with several heuristic rules in order to identify the *focus* or the *Answer Type Identifier* of the question. These are further combined with *ontology-based lookup* in order to identify the answer type. If necessary, the user is engaged in the dialog in order to solve ambiguities and precisely identify the answer type (Section 9.2). Our evaluation with 250 questions from the Mooney GeoQuery dataset shows that the answer type is correctly identified for 98.18% of questions, including 37.36% which required one dialog with the user (see Section 9.7.4). Identification of the question category is usually based on static rules which categorise questions based on their syntax. For example, questions starting with *Where* would be in a different category from questions starting with *What*. This approach is used in various guises in many similar NLIs to ontologies such as ORAKEL [Cimiano et al., 2007], PANTO [Wang et al., 2007], Querix [Kaufmann et al., 2006], and AquaLog [Lopez et al., 2007]. Our approach is different in that we try to avoid strict adherence to syntax, while engaging the user in dialog in order to map certain syntactic structures into the ontology concepts.

- *Learning from the users incrementally.* The system is able to *learn*

from the user's selections and train itself to perform better by integrating the knowledge of its users. Our learning model (inspired by Reinforcement Learning, see Section 9.5) evaluated using the Mooney GeoQuery dataset showed an improvement of our initial ranking by 6% (see Section 9.7.3). In addition, as presented in Section 9.7.2, for the Mooney GeoQuery dataset 10-fold cross-validation measurement has shown that the precision of the baseline model has been improved by 23.24%. While learning to map syntax trees to semantics has not been extensively researched in the domain of NLIs to ontologies, several promising approaches have been tried and evaluated in some other domains such as NLIs to databases (e.g., [Ge and Mooney, 2009]). Supervised approaches such as learning the semantic parser based on statistical machine translation [Wong and Mooney, 2007], statistical disambiguation models [Ge and Mooney, 2009], or a hidden-variable approach for learning to interpret sentences in context [Zettlemoyer and Collins, 2009] could all be seen as a complementary to our semi-supervised approach.

Finally, our approach to portability shifts some effort from application developers to end users. The knowledge of the end-users is used to train and improve the system for others. This knowledge is used to update the learning model, which is also preserved for sharing with other similar systems. Therefore FREyA is not only incrementally enriching its own lexicon, but it is also preserving it in a way that other NLI systems can benefit from it. The *portability* and the *suitability* of FREyA to be used in the real scenario for querying the Linked Data is tested through the experiments using the QALD-1 challenge datasets, MusicBrainz and DBpedia (Section 9.7.5).

Chapter 11

Future Challenges

The work described in this thesis can be improved in many aspects. Here we outline some ideas.

11.1 Scalability

The availability of Linked Open Data changed the way we think about ontologies and structured data. In that respect, there are several challenges which remain to be addressed or improved in comparison to how they have been addressed in the course of this thesis. In addition to the previously discussed issue of *ontologies not being perfect*, the scale becomes an issue, and also *incompleteness, heterogeneity, and noise* inherent in these data. A huge number of ontologies interlinked with each other means a high probability that there is the redundant information, which needs to be filtered out by the systems used for querying these data. Moreover, with such enormous knowledge base queries can return thousands or millions of hits, e.g. *show fungi*. The result to this query is more than 2000 instances of different types of fungi. The question is which ones to show first and how to filter out duplicates – this is an important direction towards the increased quality of LOD in the nearest future, which will lead to the better performance of interfaces used to query these data. In addition, the *ranking algorithms* (e.g., [Lopez et al., 2009a]) become increasingly important.

Large datasets introduce other challenges such as *data ambiguity*. Unlike

language ambiguity where one term might have several different meanings, the *data ambiguity* arises when one term refers to several URIs in the semantic repository, while all of them have the identical meaning. This happens when, for example, one term refers to several Ontology Concepts, each belonging to a different ontology namespace. Ideally, all these concepts should be related by *owl:sameAs*, however, based on the current state of the Linked Data, this is often not the case, and the systems that query this data need to handle these situations properly.

11.2 What to Show?

As NLIs for querying ontologies have the goal to find the answer to a question posed in Natural Language, most of the existing approaches focus on the problem of translating NL into the formal languages. However, once the answer is found, it is very important to present it to the user in a user-friendly manner. The way the the answer is shown to the user has a large impact on their confidence when using the system, and there is a room for researching this topic more carefully. One interesting approach would be using Natural Language Generation Tools such as the one described in Davis et al. [2008]. That means that instead of rendering a graph or a list of results as in FREyA, the user would receive the answer in Natural Language. For example:

```
USER: Which countries are located in Europe?
SYSTEM:  There are countries. Countries are
France and Austria. Countries are Belgium and Serbia.
```

11.3 Learning

We discussed previously the disadvantages of our learning model (see Section 9.7.2). Namely, the simplicity which makes the model attractive and re-usable across similar but not identical questions, is at the same time a drawback as it might cause ambiguities. Our current model is in the form of:

```
if
POC or OC appears in the context with the closest OC
then
map it to
candidate, function
```

This model can be relaxed so that instead of a POC, we can preserve the *phrase (e.g. noun phrase)* in which it appears. This would enable more precise mappings and would solve the problem of *point* being mapped to *geo:HiPoint* or *geo:LoPoint* depending on whether it is preceded by *lowest* or *highest* which was discussed in Section 9.7.2.

11.4 Personalised Vocabulary

All discussed methods in this thesis can be employed (and potentially improved) in combination with quality user profiles. However, creating and maintaining quality user profiles requires analysing the domain space (e.g., available domain knowledge) and user space (e.g., user interests and preferences) and making the connection between the two. The nature of ontologies is convenient for designing and intersecting these two spaces and could be accomplished through:

1. *Creating domain space:* creating or locating the domain ontology with defined concepts and relations between them so that they explain the domain precisely. Instantiating the concepts and creating relations between the instances.

2. *Creating user space:* creating or locating the user ontology with defined concepts and relations between them so that they explain user interests, preferences and activities precisely. Instantiating the concepts and creating relations between the instances.

3. *Intersection of two spaces:* connecting the two spaces would result in defining user profiles. In practice, this would mean defining relations between concepts from the domain and user space i.e. domain and user ontologies.

11.5 Using FREyA in the Open-Domain Scenario

While FREyA is tested with large and diverse datasets through the experiments with DBPedia and MusicBrainz, the approach still differs from a truly open-domain scenario, where users ask questions using the system that would crawl the whole Web, or rather, the whole Semantic Web in order to find answers. In this section we outline the obvious next steps that would need to be taken in order to use FREyA in that kind of scenario. We identify two approaches that are feasible.

The *first* approach requires less investigation in FREyA, in a sense that all algorithms and methods can be reused, with a potential requirement for optimisation due to scale. However, the approach requires development of an infrastructure that would:

- Crawl the whole Web – this could be performed using the existing semantic search engines such as Watson [d'Aquin et al., 2007] or Sindice ([Tummarello et al., 2007].
- Load all crawled files into the centralised repository.
- Repeat the previous two steps regularly in order to update the existing data.

However, developing this infrastructure might be extremely expensive and it is even questionable whether it would be possible to update the data regularly to follow updates of the RDF documents available on the Web. However, by locating and transforming the *decentralised data* on the Web into the *centralised repository*, it is possible to fully reuse the existing algorithms and methods in FREyA, as the index necessary for performing the *ontology-based lookup* can be generated, and also the centralised repository can be used to evaluate the final SPARQL query generated by FREyA in order to find the answer. However, due to the large amount of data available on the Semantic Web, some existing algorithms might need to be optimised. The most obvious *optimisations* are as follows:

The learning model. The learning model in FREyA is saved to the file system using the JSON format. The model can easily grow and cause

scalability issues if the system is used with the large-scale data. This can be solved by using a more scalable implementation, for example, the one that is based on Lucene.

Presentation of dialogs and answers. While this is not a mandatory requirement for using FREyA in the open-domain scenario, it would be worth exploring the user-friendly ways of showing URIs to the user. By default, FREyA uses values of *rdf:label* instead of URIs. This can also be customised/configured for specific datasets as there often is the case that some special properties are used for names, instead of labels. It would be worth developing a service that would return the *preferred label* for each URI, which is extracted from all available labels and named properties.

In the *second* approach, while majority of the methods and algorithms available in FREyA can be reused, in addition to the optimisations mentioned above, the following aspects would require further investigation:

Ontology-based Lookup currently requires generating an index offline, which is then used at the query analysis time. This is currently performed using a gazetteer which attaches semantic annotations to the question terms which is characterised by a *URI* for *classes, instances* and *properties*, and an *instance URI* and a *property URI* for *literals*. Using FREyA in the open-domain scenario would require a reliable service for semantic annotation with regard to the semantic resources available on the open Web. This would be a replacement for the currently used gazetteer, and one possibility is to use services such as Watson. This approach has been taken by PowerAqua, however, as pointed out in Lopez et al. [2011], the resources in the open Web that can be accessed through Watson seem to have quality issues: many ontologies are not populated, and there are many redundant, noisy and incomplete data (for example, the schemas could be missing). While this scenario is extremely powerful and can be a great demonstration of what can be done with the large amount of structured knowledge, the current tools and services seem to lack the full potential largely because of the low quality of the available data. Hence, one interesting

direction for future work should go in this direction (e.g., Hartig and Zhao [2009]).

Finding and loading relevant triples on the fly would need to be implemented, as currently, FREyA either generates a new semantic repository and populates it using the data available from the predefined URLs (either from the local system or from the Web), or it can connect to the existing repository which can be local or remote. However, the assumption behind the current implementation is that all data are loaded into the centralised repository – the same one from which the index for the Ontology-based Lookup is generated. FREyA then generates the SPARQL query, which is evaluated against the centralised repository in order to answer the question. As in the truly open-domain scenario triples are distributed on the Web, FREyA would need to implement a mechanism to locate and load the relevant triples into its repository before it generates and executes SPARQL. One way to implement this is using so called *virtual documents* as suggested in Damljanovic et al. [2011]. However, the real question here is how to identify only the relevant triples. The easiest would probably be to query Sindice using query terms as keywords, and then load all RDF documents identified as relevant. However, this approach is problematic as the list of identified documents might be large, and also the content of each RDF document, hence loading them might be time-consuming.

Appendix A

User-centric Evaluation with QuestIO

This section includes questionnaires given to the subjects in the evaluation described in Section 7.5. Figure A.1 presents the pre-task background questionnaire. Figure A.2 shows the questionnaire which is answered by participants after each task, and Figure A.3 shows the Standard Usability Scale (SUS) answered by each participant at the end of the experiment.

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	I understand the term *semantic web* and *ontologies*.	☐	☐	☐	☐	☐
2	I understand the term semantic search.	☐	☐	☐	☐	☐
3	I have used ontology editors to view ontologies.	☐	☐	☐	☐	☐
4	I am familiarised with QuestIO (Question-based Interface to Ontologies).	☐	☐	☐	☐	☐
		<2 years	<5 years	<10 years	other:	
5	How long have you been working with GATE?	☐	☐	☐	☐	
		GATE developer	GATE user	none	other:	
6	Would you consider yourself:(select all that apply)	☐	☐	☐	☐	
		Google	GATE support page	mailing list	source code	other:
7	When you have problems developing/using GATE, what do you do?(select all that apply)	☐	☐	☐	☐	☐
		3-5 days/week	1-2 days/week	1/month	rarely	never
8	How often you perform search about various GATE components (including search for publications on the GATE web site)?	☐	☐	☐	☐	☐
9	How often on average you are asked about the GATE components (including the mailing list, emails, personal contacts)?	☐	☐	☐	☐	☐

Figure A.1: Pre-test questionnaire

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	Was the result found by the prototype relevant? Include any comments you may have in the field below.	☐	☐	☐	☐	☐
2	It was easier to perform the task using prototype in comparison to the standard ways I use to search data about GATE. Include any comments you may have in the field below.	☐	☐	☐	☐	☐
3	Was the possibility to browse the ontology in the prototype confusing or helpful? Include any comments you may have in the field below.	☐	☐	☐	☐	☐
4	How did you find the refinement pane? Helpful or confusing? Report any comments you may have in the field below.	☐	☐	☐	☐	☐
5	Was it easy to formulate a query for prototype? Include any comments you may have in the field below.	☐	☐	☐	☐	☐
6	Do you have any comments on how the prototype could be improved?	☐	☐	☐	☐	☐
7	Do you have any specific problems to report?	☐	☐	☐	☐	☐

Figure A.2: Post-test questionnaire for each task

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
2	I found the system unnecessarily complex	☐	☐	☐	☐	☐
3	I thought the system was easy to use	☐	☐	☐	☐	☐
4	I think that I would need the support of a technical person to be able to use this system	☐	☐	☐	☐	☐
5	I found the various functions in this system were well integrated	☐	☐	☐	☐	☐
6	I thought there was too much inconsistency in this system	☐	☐	☐	☐	☐
7	I would imagine that most people would learn to use this system very quickly	☐	☐	☐	☐	☐
8	I found the system very cumbersome to use	☐	☐	☐	☐	☐
9	I felt very confident using the system	☐	☐	☐	☐	☐
10	I needed to learn a lot of things before I could get going with this system	☐	☐	☐	☐	☐

Figure A.3: Post-test questionnaire (SUS) for the QuestIO prototype

Appendix B

User-centric Evaluation with FREyA

This section includes questionnaires given to the subjects in the evaluation described in Section 8.2, including questionnaires completed after finishing each task, followed by the background questionnaire and the SUS usability survey.

DO NOT DO BOTH TASK 1a and TASK 1b, BUT CHOOSE ONE OF THEM
Task 1a: Find part of speech taggers which exist in GATE. Find out which parameters exist for the POS Tagger of your choice.

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Did you find *Identified context*:
   - Confusing ☐
   - Helpful ☐
   - Other: ☐

Enter any comments/suggestions/problems you may have:

DO NOT DO BOTH TASK 1a and TASK 1b, BUT CHOOSE ONE OF THEM
Task 1b: Find mountains which exist in United States. Find out in which state is the mountain of your choice located.

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Did you find *Identified context*:
   - Confusing ☐
   - Helpful ☐
   - Other: ☐

Enter any comments/suggestions/problems you may have:

DO NOT DO BOTH TASK 2a and TASK 2b, BUT CHOOSE ONE OF THEM

Task 2a: Imagine that you are a GATE developer who needs to extend the RASP Parser. Your task is to find out the names of *init* parameters.

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Was it clear for you that there are no init parameters for RASP Parser?
   - Yes ☐
   - No ☐ If you tick this box indicate why: ☐

Enter any comments/suggestions/problems you may have:

DO NOT DO BOTH TASK 2a and TASK 2b, BUT CHOOSE ONE OF THEM

Task 2b: Find out which states border hawaii.

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Was it clear from the answer that there are no border states?
   - Yes ☐
   - No ☐ If you tick this box indicate why: ☐

Enter any comments/suggestions/problems you may have:

DO NOT DO BOTH TASK 3a and TASK 3b, BUT CHOOSE ONE OF THEM
Task 3a: What are the parameters of the PRs which are included in the same plugin as the Morhper?

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Did you find *Identified context*:
   - Confusing ☐
   - Helpful ☐
   - Other: ☐

Enter any comments/suggestions/problems you may have:

DO NOT DO BOTH TASK 3a and TASK 3b, BUT CHOOSE ONE OF THEM

Task 3b: Which rivers flow through the state in which the mountain harvard is located?

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Did you find *Identified context*:
   - Confusing ☐
   - Helpful ☐
   - Other: ☐

Enter any comments/suggestions/problems you may have:

Task 4: Try exploring the knowledge available in the system. Either search for various components of GATE such as PRs, plugins, LRs, VRs, or explore geography of United States by inquiring about: cities, states, rivers, mountains, highways just to get the idea of what you can search for. Then ask some questions in order to connect these concepts such as 'which states border georgia?' or 'which rivers flow through states which border california'. Input as many queries as you like.

After you finish the task answer the following questions:

1. Which of the following is truth:
   - I completed the task with ease ☐
   - I completed the task with difficulty ☐
   - I failed to complete the task ☐ If you ticked this box please provide the reasons for this:
     - System provided no useful output so I could not figure out what to do ☐
     - System provided confusing output so I could not figure out what to do ☐
     - System provided no output ☐
     - Other reasons: ☐
2. Was it easy to formulate the query for this task:
   - Yes ☐
   - No ☐
3. Did you find *Identified context*:
   - Confusing ☐
   - Helpful ☐
   - Other: ☐

Enter any comments/suggestions/problems you may have:

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
	(tick one box only)					
1	I understand the term ontologies.	☐	☐	☐	☐	☐
2	I have used ontology editors to view ontologies.	☐	☐	☐	☐	☐
3	I have used SPARQL before.	☐	☐	☐	☐	☐

4 What is your age (optional)? ☐

5 Your profession?

☐

6 Which username you want to use? (optional)

☐

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
2	I found the system unnecessarily complex	☐	☐	☐	☐	☐
3	I thought the system was easy to use	☐	☐	☐	☐	☐
4	I think that I would need the support of a technical person to be able to use this system	☐	☐	☐	☐	☐
5	I found the various functions in this system were well integrated	☐	☐	☐	☐	☐
6	I thought there was too much inconsistency in this system	☐	☐	☐	☐	☐
7	I would imagine that most people would learn to use this system very quickly	☐	☐	☐	☐	☐
8	I found the system very cumbersome to use	☐	☐	☐	☐	☐
9	I felt very confident using the system	☐	☐	☐	☐	☐
10	I needed to learn a lot of things before I could get going with this system	☐	☐	☐	☐	☐

Do you have any specific **problems** to report?
Do you have any **suggestions** for improving the system?

Appendix C

Using Large Ontologies

In this section we present the SPARQL queries used to initialise FREyA, and in particular the LKB gazetteer which is used by FREyA, with the DBPedia dataset. Loading the DBPedia lexicon is performed in several phases. The first phase is finding filter classes using the SPARQL query:

```
SELECT ?T ?T1 WHERE {
?T  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.w3.org/2002/07/owl#Class>.
# Selects only the top level classes of DBPedia
OPTIONAL
{ ?T <http://www.w3.org/2000/01/rdf-schema#subClassOf> ?T1.
    FILTER (regex(str(?T1), "^http://dbpedia\\.org/.*$", "i")).
    FILTER (?T1!=?T).}
    FILTER (!bound(?T1)).
# DBPedian namespace filter
    FILTER (regex(str(?T), "^http://dbpedia\\.org/.*$", "i")).
}
```

When this query is executed against FactForge SPARQL endpoint (`http://factforge.net/sparql`) it resulted in 2803 filter classes. For each filter class one SPARQL query is further executed in order to generate the domain lexicon. The SPARQL query looked similar to the following:

```
SELECT DISTINCT ?E ?T ?L ?P
```

```
WHERE {
{ ?E <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?T.
# Label Retrieval sub-component
OPTIONAL {
?E ?P ?L.
FILTER (?P = <http://www.w3.org/2000/01/rdf-schema#label>).
Filter(lang(?L)="en")
}
} UNION

#### Query component extracting the entities knowledge
{ ?E <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?T.
FILTER (regex(str(?T), "^http://dbpedia\\.org/.*$", "i")).
# Direct-type enforcing criterion
OPTIONAL
{ ?E <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?T1.
?T1 <http://www.w3.org/2000/01/rdf-schema#subClassOf> ?T.
FILTER (?T1!=?T).}
FILTER (!bound(?T1)).
# Label Retrieval sub-component
?E ?P ?L.
# Remove sub-properties because they duplicate the base property
OPTIONAL
{ ?P <http://www.w3.org/2000/01/rdf-schema#subPropertyOf> ?P1.
FILTER (regex(str(?P1), "^http://dbpedia\\.org/.*$", "i")). }
FILTER (!bound(?P1)).
# Forces predicates to be from the DBPedia domain OR rdfs:label
FILTER (regex(str(?P), "^http://dbpedia\\.org/.*$", "i") ||
(?P = <http://www.w3.org/2000/01/rdf-schema#label>)).
FILTER isLiteral(?L).
# Forces labels to have at least one latin letter
FILTER (regex(str(?L), "^.*[A-Z].*$", "i")).
# Forces label to have english language tag
FILTER (langMatches(lang(?L), "en")).
Filter(lang(?L)="en")
```

```
}
# General domain name filter for the Entity
FILTER (regex(str(?E), "^http://dbpedia\\.org/.*$", "i")) .
?E <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Class>.}
#LIMIT 10
```

The only difference between the 2803 SPARQL queries was in the last filter statement were FILTER-CLASS is the URI of the class:

```
?E <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> FILTER-CLASS.}
```

Loading DBpedia dictionary took 19 days, and resulted in extracting 2022854 terms into the domain lexicon. The experiment is conducted on a CentOS 5.2 Linux virtual machine running on a AMD Opteron 2431 2.40GHz CPU with 2 cores and 20G RAM allocated to that particular instance.

Bibliography

Bruce W. Ballarda Alan W. Biermanna and Anne H. Sigmon. An Experimental Study of Natural Language Programming. *International Journal of Man-Machine Studies*, 18(1):71–87, 1983.

James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A Robust System for Natural Spoken Dialogue. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 62–70, Morristown, NJ, USA, 1996. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/981863.981872.

Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural Language Interfaces to Databases - An Introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

Grigoris Antoniou and Frank van Hermelen. *A Semantic Web Primer*. MIT Press, 2nd edition, 2008.

Bob Bailey. Getting the Complete Picture with Usability Testing. Usability Updates Newsletter, U.S. Department of Health and Human Services, March 2006. URL `http://www.usability.gov/articles/newsletter/pubs/030106news.html`.

Madeleine Bates. Rapid Porting of the PARLANCE Natural Language Interface. In *HLT '89: Proceedings of the workshop on Speech and Natural Language*, pages 83–88, Morristown, NJ, USA, 1989a. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/100964.100966.

Marcia J. Bates. The Design of Browsing and Berrypicking Techniques for

the Online Search Interface. *Online Review*, 13(5):407–424, 1989b. URL `http://www.gseis.ucla.edu/faculty/bates/berrypicking.html`.

Sean Bechhofer, Robert Stevens, Gary Ng, Alex Jacoby, and Carole Goble. Guiding the User: an Ontology Driven Interface. *User Interfaces to Data Intensive Systems, 1999. Proceedings*, pages 158–161, 1999. doi: 10.1109/UIDIS.1999.791472.

Richard K. Belew. *Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW*. Cambridge University Press, Cambridge, United Kingdom, 2000. ISBN ISBN-13: 9780521630283 — ISBN-10: 0521630282.

Tim Berners-Lee. *Weaving the Web*. Orion Business Books, 1999.

Abraham Bernstein and Esther Kaufmann. GINO – A Guided Input Natural Language Ontology Editor. In *5th International Semantic Web Conference (ISWC2006)*, 2006.

Abraham Bernstein, Esther Kaufmann, and Christian Kaiser. Querying the Semantic Web with Gingseng: A Guided Input Natural Language Search Engine. In *15th Workshop on Information Technologies and Systems*, pages 112—126, Las Vegas, NV, 2005.

Patrick Blackburn and Johan Bos. Working with Discourse Representation Structures. In *Representation and Inference for Natural Language: A First Course in Computational Linguistics*, volume 2, September 1999.

Harold Boley. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. In *Proceedings of the Applications of Prolog 14th international conference on Web knowledge management and decision support*, INAP'01, pages 5–22, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-00680-X. URL `http://portal.acm.org/citation.cfm?id=1767370.1767373`.

Jeen Broekstra and Arjohn Kampman. SeRQL: A Second Generation RDF Query Language. In *In Proceedings of the SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14, 2003.

John Brooke. SUS: a "Quick and Dirty" Usability Scale. In P.W. Jordan, B. Thomas, B.A. Weerdmeester, and A.L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, UK, 1996. URL `http://www.usabilitynet.org/trump/documents/Suschapt.doc`.

Joe Bullock. *Informed Navigation: Description Logic Based Hypermedia Linking*. PhD thesis, University of Manchester, UK, 1999.

Robin D. Burke, Kristian J. Hammond, Vladimir Kulyukin, Steven L. Lytinen, Noriko Tomuro, and Scott Schoenberg. Question Answering from Frequently-Asked Question Files: Experiences with the FAQ Finder System. Technical report, AI Magazine, 1996.

David Chin. An Analysis of Scripts Generated in Writing Between Users and Computer Consultants. *National Computer Conference*, pages 637–642, 1984.

Kenneth Church and Ramesh Patil. Coping with Syntactic Ambiguity or How to Put the Block in the Box. *American Journal of Computational Linguistics*, 8(3-4), 1982.

Philipp Cimiano, Peter Haase, and Jörg Heizmann. 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, 2007. ACM. ISBN 1-59593-481-2. doi: http://doi.acm.org/10.1145/1216295.1216330.

Philipp Cimiano, Peter Haase, Jörg Heizmann, Matthias Mantel, and Rudi Studer. Towards Portable Natural Language Interfaces to Knowledge Bases – the Case of the ORAKEL System. *Data and Knowledge Engineering*, 65(2):325—354, May 2008. doi: http://www.sciencedirect.com/science/article/B6TYX-4R68N7K-2/1/cb51e20a3f7e9877671960f8a1336595.

Peter Clark, Philip Harrison, Thomas Jenkins, John Thompson, and Richard H. Wojcik. Acquiring and Using World Knowledge Using a Restricted Subset of English. In Ingrid Russell and Zdravko Markov, editors,

Proceedings of the 18th International FLAIRS Conference (FLAIRS'05), pages 506–511. AAAI Press, 2005. ISBN 1-57735-234-3. URL `http://www.cs.utexas.edu/users/pclark/papers/flairs.pdf`.

Peter Clark, Shaw-Yi Chaw, Ken Barker, Vinay Chaudhri, Philip Harrison, James Fan, Bonnie John, Bruce Porter, Aaron Spaulding, John Thompson, and Peter Yeh. Capturing and Answering Questions Posed to a Knowledge-Based System. In *Proceedings of the 4th International Conference on Knowledge Capture (K-CAP'07)*, 2007. URL `http://www.cs.utexas.edu/users/pclark/papers/kcap07.pdf`. http://www.cs.utexas.edu/users/pclark/papers/kcap07.ppt.

Edgar F. Codd. Seven Steps to Rendezvous with the Casual User. In *IFIP Working Conference Data Base Management*, pages 179–200, 1974.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

Anne Cregan, Rolf Schwitter, and Thomas Meyer. Sydney OWL Syntax - Towards a Controlled Natural Language Syntax for OWL 1.1. In *OWLED*, 2007.

Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 1 edition, February 2009. ISBN 0136072240. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0136072240`.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

Paulo Cesar G. da Costa, Kathryn B. Laskey, Kenneth J. Laskey, and Michael Pool, editors. *End-User Evaluations of Semantic Web Technologies*, 2005.

Danica Damljanovic. Towards Portable Controlled Natural Languages for Querying Ontologies. In Michael Rosner and Norbert Fuchs, editors, *Pro-*

ceedings of the 2nd Workshop on Controlled Natural Language, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Marettimo Island, Sicily, September 2010.

Danica Damljanovic and Kalina Bontcheva. Enhanced Semantic Access to Software Artefacts. In *Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Karlsruhe, Germany, October 2008.

Danica Damljanovic and Vladan Devedzic. Applying semantic web to e-tourism. In Zongmin Ma and Huaiqing Wang, editors, *The Semantic Web for Knowledge and Data Management: Technologies and Practices*. Information Science Reference (IGI Global), 2008.

Danica Damljanovic and Vladan Devedzic. Semantic Web and E-tourism. In Mehdi Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology, Second edition*. IGI Global, 2009.

Danica Damljanovic, Valentin Tablan, and Kalina Bontcheva. A Text-based Query Interface to OWL Ontologies. In *6th Language Resources and Evaluation Conference (LREC)*, Marrakech, Morocco, May 2008. ELRA.

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Usability of Natural Language Interfaces for Querying Ontologies (poster). In *Workshop on Controlled Natural Language (CNL 2009)*, 2009.

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In *7th Language Resources and Evaluation Conference (LREC)*, La Valletta, Malta, May 2010a. ELRA.

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, Lecture Notes in Computer Science, Heraklion, Greece, June 2010b. Springer-Verlag.

Danica Damljanovic, Johann Petrak, Mihai Lupu, Hamish Cunningham, Mats Carlsson, Gunnar Engstrom, and Bo Andersson. Random Indexing

for Finding Similar Nodes within Large RDF graphs . In *Proceedings of the Fourth International Workshop on Resource Discovery, Collocated with the 8th Extended Semantic Web Conference (ESWC 2011)*, Heraklion, Greece, June 2011.

Hoa Trang Dang. Overview of the TAC 2008 Opinion Question Answering and Summarization Tasks, 2008.

Hoa Trang Dang, Jimmy Lin, and Diane Kelly. Overview of the TREC 2006 Question Answering Track. In *Proceedings of the Fifteenth Text REtrieval Conference*, 2006.

Hoa Trang Dang, Diane Kelly, and Jimmy Lin. Overview of the TREC 2007 Question Answering Track. In *Proceedings of the Sixteenth Text REtrieval Conference*, 2007.

Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Watson: A Gateway for Next Generation Semantic Web Applications. *Poster, ISWC 2007,*, 2007. URL `http://iswc2007.semanticweb.org/papers/Paper366-Watson-poster-ISWC07.pdf`.

John Davies, Dieter Fensel, and Frank van Harmelen, editors. *Towards the Semantic Web: Ontology-driven Knowledge Management*. Wiley, 2002.

Brian Davis, Ahmad Ali Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. RoundTrip Ontology Authoring. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2008. ISBN 978-3-540-88563-4.

Ronald Denaux, Vania Dimitrova, Anthony G. Cohn, Catherine Dolbear, and Glen Hart. Rabbit to OWL: Ontology Authoring with a CNL-Based Tool. In Norbert E. Fuchs, editor, *CNL*, volume 5972 of *Lecture Notes in Computer Science*, pages 246–264. Springer, 2009. ISBN 978-3-642-14417-2.

Yihong Ding, David Embley, and Stephen Liddle. Automatic Creation and Simplified Querying of Semantic Web Content: An Approach Based on Information-Extraction Ontologies. In *Proceedings of the 1st Asian Semantic Web Conference*, pages 400–414. Springer Berlin/Heidelberg, September 2006.

Samuel S. Epstein. Transportable Natural Language Processing through Simplicity—the PRE System. *ACM Trans. Inf. Syst.*, 3(2):107–120, 1985. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/3914.3985.

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

Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn, Gerold Schneider, Loic Royer, and Michael Schröder. Attempto Controlled English and the semantic web. Deliverable I2D7, REWERSE Project, April 2006. URL `http://rewerse.net/deliverables/m24/i2-d7.pdf`.

Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web, Fourth International Summer School 2008*, number 5224 in Lecture Notes in Computer Science, pages 104–124. Springer, 2008.

Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, Brian Davis, and Siegfried Handschuh. CLOnE: Controlled Language for Ontology Editing. In *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, November 2007. URL `http://gate.ac.uk/sale/iswc07/clone/clone.pdf`.

Ruifang Ge and Raymond J. Mooney. Learning a Compositional Semantic Parser Using an Existing Syntactic Parser. In *ACL-IJCNLP '09: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2*, pages 611–619, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-46-6.

Yolanda Gil and Natasha Fridman Noy, editors. *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*, 2009. ACM. ISBN 978-1-60558-658-8.

Mark Greenwood. *Open-Domain Question Answering.* PhD thesis, University of Sheffield, Department of Computer Science, 2006.

Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C. N. Pereira. TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *Artificial Intelligence*, 32(2):173 – 243, 1987.

Thomas R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers. URL `http://www.citeulike.org/group/1808/article/821199`.

Ramanathan Guha, Rob McCool, and Eric Miller. Semantic Search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 700—709, New York, NY, USA, 2003. ACM. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775250.

Catalina Hallett. Generic Querying of Relational Databases Using Natural Language Generation Techniques. In *Proceedings of the Fourth International Natural Language Generation Conference*, pages 95–102. Association for Computational Linguistics, July 2006.

Catalina Hallett, Donia Scott, and Richard Power. Composing Questions through Conceptual Authoring. *Computational Linguistics*, 33(1):105–133, 2007.

Larry Harris. Experience with INTELLECT: Artificial Intelligence Technology Transfer. *AI Magazine*, 5(2), 1984. ISSN 0738-4602. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/437/373`.

Glen Hart, Martina Johnson, and Catherine Dolbear. Rabbit: Developing a Control Natural Language for Authoring Ontologies. In *Proceedings of the European Semantic Web Conference ESWC 2008, Tenerife, Spain*, pages

348—360. Springer, June 1-5 2008. doi: 10.1007/978-3-540-68234-9_27. URL http://www.springerlink.com/content/e6370k7l62gnm514.

Olaf Hartig and Jun Zhao. Using Web Data Provenance for Quality Assessment. In *Proceedings of the First International Workshop on the Role of Semantic Web in Provenance Management (SWPM'09) at the International Semantic Web Conference (ISWC'09)*, Washington D.C., USA, 2009.

Jeff Heflin and James Hendler. Searching the Web with SHOE. In *AAAI Workshop 2000*, pages 35–40, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.8348.

Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a Natural Language to Complex Data. *ACM Transactions on Database Systems*, 3:105–147, 1978.

Lynette Hirschman, Marc Light, Eric Breck, and John D. Burger. Deep Read: A Reading Comprehension System. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 325–332, 1999.

Ian Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4–9, March 2002.

Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosofand, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004. URL http://www.w3.org/Submission/SWRL/.

Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Ranking Approximate Answers to Semantic Web Queries. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bontas Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009. ISBN 978-3-642-02120-6.

Eero Hyvönen and Eetu Mäkelä. Semantic Autocompletion. In *Proceedings of the first Asia Semantic Web Conference (ASWC 2006), Beijing*. Springer-Verlag, New York, August 4-9 2006.

Alex Iskold. Semantic Search: The Myth and Reality. *ReadWriteWeb*, May 2008. URL `http://www.readwriteweb.com/archives/semantic_search_the_myth_and_reality.php`.

Susan Jamieson. Likert Scales: How to (Ab)Use Them. *Medical Education*, 38(12):1217–1218, 2004. URL `http://www.ncbi.nlm.nih.gov/pubmed/15566531`.

Anastasia Karanastasi, Alexandros Zotos, and Stavros Christodoulakis. The OntoNL Framework for Natural Language Interface Generation and a Domain-Specific Application. In *Digital Libraries: Research and Development*, pages 228–237. Springer Berlin / Heidelberg, 2007.

Rohit J. Kate and Raymond J. Mooney. Learning Language Semantics from Ambiguous Supervision. In *AAAI*, pages 895–900. AAAI Press, 2007. ISBN 978-1-57735-323-2.

Esther Kaufmann and Abraham Bernstein. 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.

Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. 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 2006.

Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. NLP-Reduce: A Naive but Domain-independent Natural Language Interface for Querying Ontologies. In *Proceedings of the European Semantic Web Conference ESWC 2007, Innsbruck, Austria*. Springer, June 4-5 2007.

Nancy Davis Kho. Letting End Users Ask Questions, Stat! June 2008.

Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-oriented and Frame-based Languages. *Journal of the ACM*, 42:741–843,

July 1995. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/210332.210335. URL http://doi.acm.org/10.1145/210332.210335.

Dan Klein and Christopher D. Manning. Fast Exact Inference with a Factored Model for Natural Language Parsing. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 - Neural Information Processing Systems, NIPS 2002*, pages 3–10. MIT Press, 2002. ISBN 0-262-02550-7. URL http://books.nips.cc/papers/files/nips15/CS01.pdf.

Jürgen Krause. Natural Language Access to Information Systems. An Evaluation Study of its Acceptance by End Users. *Information Systems*, 5: 297–319, 1980.

Julian Kupiec. MURAX: A Robust Linguistic Approach for Question Answering Using an On-Line Encyclopedia. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett 0002, editors, *SIGIR*, pages 181–190. ACM, 1993. ISBN 0-89791-605-0.

Cody Kwok, Oren Etzioni, and Daniel S. Weld. Scaling Question Answering to the Web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/502115.502117.

Yuangui Lei, Victoria Uren, and Enrico Motta. SemSearch: a Search Engine for the Semantic Web. In *Managing Knowledge in a World of Networks*, pages 238–245. Springer Berlin / Heidelberg, 2006.

Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. Adapting SVM for Data Sparseness and Imbalance: A Case Study on Information Extraction. *Natural Language Engineering*, 15(2):241–271, 2009. URL http://journals.cambridge.org/repo_A45LfkBD.

Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

Serge Linckels and Christoph Meinel. Semantic Interpretation of Natural Language User Input to Improve Search in Multimedia Knowledge Base. *it - Information Technologies*, 49(1):40–48, 2007.

Diane Litman. Spoken Dialogue for Intelligent Tutoring Systems: Opportunities and Challenges . http://www.cs.pitt.edu/~litman/keynote06_noav.ppt, 2006.

Diane Litman and Kate Forbes-Riley. Correlations between Dialogue Acts and Learning in Spoken Tutoring Dialogues. *Natural Language Engineering*, 12(02):161–176, 2006. doi: 10.1017/S1351324906004165. URL http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=439981&fulltextType=RA&fileId=S1351324906004165.

Diane J. Litman and Scott Silliman. ITSPOKE: an Intelligent Tutoring Spoken Dialogue System. In *HLT-NAACL '04: Demonstration Papers at HLT-NAACL 2004*, pages 5–8, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

Vanessa Lopez and Enrico Motta. Ontology-driven Question Answering in AquaLog. In *NLDB 2004 (9th International Conference on Applications of Natural Language to Information Systems)*, Manchester, UK, 2004.

Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. 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 2007.

Vanessa Lopez, Andriy Nikolov, Miriam Fernandez, Marta Sabou, Victoria Uren, and Enrico Motta. Merging and Ranking Answers in the Semantic Web: The Wisdom of Crowds. In Asunción Gómez-Pérez, Yong Yu, and Ying Ding, editors, *The Semantic Web*, volume 5926 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2009a. URL http://dx.doi.org/10.1007/978-3-642-10871-6_10. 10.1007/978-3-642-10871-6_10.

Vanessa Lopez, Victoria S. Uren, Marta Sabou, and Enrico Motta. Cross Ontology Query Answering on the Semantic Web: an Initial Evaluation. In Gil and Noy [2009], pages 17–24. ISBN 978-1-60558-658-8.

Vanessa Lopez, Miriam Fernndez, Enrico Motta, and Nico Stieler. PowerAqua: Supporting Users in Querying and Exploring the Semantic Web Content. *Semantic Web Journal*, 2011.

Eetu Mäkelä. View-Based Search Interfaces for the Semantic Web. Master's thesis, University of Helsinki, June 2006.

Ashok Malhotra. *Design criteria for a Knowledge based English Language System for Management: An Experimental Analysis.* PhD thesis, Massachusetts Institute of Technology, Alfred P. Sloan School of Management, 1975.

Frank Manola and Eric Miller. RDF primer. W3C recommendation, W3C, February 2004. Published online on February 10th, 2004 at `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A Fully Statistical Approach to Natural Language Interfaces. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 55–61, Morristown, NJ, USA, 1996. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/981863.981871.

Dan Moldovan and Sanda M. Harabagiu. The Structure and Performance of an Open-domain Question Answering System. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics (ACL-2000)*, pages 563–570, 2000.

Raymond J. Mooney. Learning for Semantic Interpretation: Scaling Up without Dumbing Down. In James Cussens and Saso Dzeroski, editors, *Learning Language in Logic*, volume 1925 of *Lecture Notes in Computer Science*, pages 57–66. Springer, 1999. ISBN 3-540-41145-3.

Enrico Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving.* IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1999. ISBN 1586030035.

Jacob Neilsen. Heuristic Evaluation. In J. Neilsen and R.L. Mack, editors, *Usability Inspection Methods.* John Wiley and Sons, New York, 1994.

Jakob Nielsen. Enhancing the Explanatory Power of Usability Heuristics. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 152—158, New York, NY, USA, 1994. ACM. ISBN 0-89791-650-6. doi: http://doi.acm.org/10.1145/191666.191729.

Yun Niu, Graeme Hirst, Gregory McArthur, and Patricia Rodriguez-Gianolli. Answering Clinical Questions with Role Identification. In *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, pages 73–80, 2003.

William Ogden and Philip Bernick. Using Natural Language Interfaces. In M. Helander, editor, *Handbook of Human-Computer Interaction*. Elsevier Science Publishers B.V. (North-Holland), 1997.

William Ogden, James Mcdonald, Philip Bernick, and Roger Chadwick. Habitability in Question-Answering Systems, Series: Text, Speech and Language Technology. In T. Strzalkowski and S. Harabagiu, editors, *Advances in Open Domain Question Answering*, volume 32, pages 457–473. Springer, Netherlands, 2006.

Andrew S. Patrick and Thomas E. Whalen. Field Testing a Natural Language Information System: Usage Characteristics and Users' Comments. *Interacting with Computers*, 4(2):218–230, 1992.

John L. Phillips. *How to Think about Statistics*. W. H. Freeman and Company, New York, 1996.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a Theory of Natural Language Interfaces to Databases. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149—157, New York, NY, USA, 2003. ACM. ISBN 1-58113-586-6. doi: http://doi.acm.org/10.1145/604045.604070.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 141, Morristown, NJ, USA, 2004. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220355.1220376.

Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, and Miroslav Goranov. KIM – Semantic Annotation Platform. In *2nd International Semantic Web Conference (ISWC2003)*, pages 484–499, Berlin, 2003. Springer.

Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov. KIM – A Semantic Platform for Information Extraction and Retrieval. *Natural Language Engineering*, 10:375–392, 2004.

Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction*. Addison-Wesley, Wokingham, UK, 1994.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation - 15 january 2008, W3C, 2008. Available at http://www.w3.org/TR/rdf-sparql-query/.

Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-Validation. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009. ISBN 978-0-387-35544-3, 978-0-387-39940-9.

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. FrameNet II: Extended Theory and Practice. Technical report, ICSI, 2005. URL `http://framenet.icsi.berkeley.edu/book/book.pdf`.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada, 2003.

Rolf Schwitter, Kaarel Kaljurand, Anne Cregan, Catherine Dolbear, and Glen Hart. A Comparison of Three Controlled Natural Languages for OWL 1.1. In *In 4th OWL Experiences and Directions Workshop (OWLED 2008 DC*, 2008.

Robert F. Simmons. Answering English Questions by Computer: a Survey. *Commun. ACM*, 8(1):53–70, 1965. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/363707.363732.

Brian M. Slator, Matthew P. Anderson, and Walt Conley. Pygmalion at the Interface. *Communications of the ACM*, 29(7):599–604, 1986.

Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide. World Wide Web Consortium, Recommendation REC-owl-guide-20040210, February 2004.

Nenad Stojanovic. On the Role of a User's Knowledge Gap in an Information Retrieval Process. In *Proceedings of the Third International Conference on Knowledge Capture*, October 2005a.

Nenad Stojanovic. On the Query Refinement in the Ontology-based Searching for Information. *Information Systems*, 30(7):543–563, 2005b.

Tomek Strzalkowski and Sanda M. Harabagiu. *Advances in Open Domain Question Answering*. Springer, Netherlands, 2006.

Heiner Stuckenschmidt, Anita de Waard, Ravinder Bhogal, Christiaan Fluit, Arjohn Kampman, Jan van Buel, Erik M. van Mulligen, Jeen Broekstra, Ian Crowlesmith, Frank van Harmelen, and Tony Scerri. A Topic-Based Browser for Large Online Resources. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *EKAW*, volume 3257 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2004. ISBN 3-540-23340-7.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an Introduction*. MIT Press, Cambridge, Mass., 1998.

Valentin Tablan, Tamara Polajnar, Hamish Cunningham, and Kalina Bontcheva. User-friendly Ontology Authoring Using a Controlled Language. In *5th Language Resources and Evaluation Conference (LREC)*, Genoa, Italy, May 2006. ELRA. URL `http://gate.ac.uk/sale/lrec2006/clie/clie.pdf`.

Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A Natural Language Query Interface to Structured Information. In *Proceedings of the 5h European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 361–375, Tenerife, Spain, June 2008. Springer-Verlag New York Inc.

Lappoon R. Tang and Raymond J. Mooney. Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In *In Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, Freiburg, Germany, 2001.

Craig W. Thompson, Paul Pazandak, and Harry R. Tennant. Talk to Your Semantic Web. *IEEE Internet Computing*, 9(6):75–78, 2005.

Thanh Tran, Tobias Mathäß, and Peter Haase. Usability of Keyword-Driven Schema-Agnostic Search. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2010. ISBN 978-3-642-13488-3.

Thomas S. Tullis and Jacqueline N. Stetson. A Comparison of Questionnaires for Assessing Website Usability. In *Usability Professionals' Association Conference*, Minneapolis, Minnesota, June 2004. URL `http://home.comcast.net/%7Etomtullis/publications/UPA2004TullisStetson.pdf`.

Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the Open Linked Data. In *Proceedings of the 6th International Semantic Web Conference*, Busan, Korea, 2007.

Mike Unwalla. AECMA Simplified English. Communicator, Winter 2004.

Ellen M. Voorhees. The TREC-8 Question Answering Track Report. In *Proceedings of the Eighth Text REtrieval Conference*, pages 77–82, 1999.

Ellen M. Voorhees. Overview of the TREC-9 Question Answering Track. In *Proceedings of the Ninth Text REtrieval Conference*, pages 71–80, 2000.

Ellen M. Voorhees. Overview of the TREC 2001 Question Answering Track. In *Proceedings of the Tenth Text REtrieval Conference*, pages 42–51, 2001.

Ellen M. Voorhees. Overview of the TREC 2002 Question Answering Track. In *Proceedings of the Eleventh Text REtrieval Conference*, pages 115–123, 2002.

Ellen M. Voorhees. Overview of the TREC 2003 Question Answering Track. In *Proceedings of the Twelfth Text REtrieval Conference*, pages 54–68, 2003.

Ellen M. Voorhees. Overview of the TREC 2004 Question Answering Track. In *Proceedings of the Thirteen Text REtrieval Conference*, 2004.

David Waltz. Natural Language Access to a Large Database: an Engineering Approach. In *IJCAI'75: Proceedings of the 4th international joint conference on Artificial intelligence*, pages 868–872, San Francisco, CA, USA, 1975. Morgan Kaufmann Publishers Inc.

David L. Waltz. An English Language Question Answering System for a Large Relational Database. *Commun. ACM*, 21(7):526–539, 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359545.359550.

Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. PANTO: A Portable Natural Language Interface to Ontologies. In *The Semantic Web: Research and Applications*, pages 473—487. Springer, 2007. doi: 10.1007/978-3-540-72667-8_34.

Haofen Wang, Qiaoling Liu, Thomas Penin, Linyun Fu, Lei Zhang, Duc Thanh Tran, Yong Yu, and Yue Pan. Semplore: A Scalable IR Approach to Search the Web of Data. *Journal of Web Semantics*, 7(3), 2009.

David H. D. Warren and Fernando C. N. Pereira. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *Computational Linguistics*, 8:110–122, July 1982. ISSN 0891-2017. URL `http://portal.acm.org/citation.cfm?id=972942.972944`.

William C. Watt. Habitability. *American Documentation*, pages 338–351, July 1968.

Nick Webb and Bonnie L. Webber. Special Issue on Interactive Question Answering: Introduction. *Natural Language Engineering*, 15(1):1–8, 2009. ISSN 1351-3249. doi: http://dx.doi.org/10.1017/S1351324908004877.

Daniel Weld. Invited Talk: "Machine Reading: from Wikipedia to the Web". In Gil and Noy [2009]. ISBN 978-1-60558-658-8.

Terry Winograd. *Understanding Natural Language*. Academic Press, Inc., Orlando, FL, USA, 1972. ISBN 0127597506.

Yuk Wah Wong and Raymond J. Mooney. Learning for Semantic Parsing with Statistical Machine Translation. In Robert C. Moore, Jeff A. Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *HLT-NAACL*. The Association for Computational Linguistics, 2006.

Yuk Wah Wong and Raymond J. Mooney. Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, pages 960–967. Association for Computational Linguistics, 2007.

William A. Woods, Robert M. Kaplan, and Bonnie Lynn Nash-Webber. The Lunar Sciences Natural Language Information System: Final Report. Technical Report BBN Report 2378, Bold Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

Karl L. Wuensch. An Introduction to Within Subjects Analysis of Variance, 2009. URL `http://core.ecu.edu/psyc/wuenschk/MV/RM-ANOVA/ws-anova.doc`.

John M. Zelle and Raymond J. Mooney. Learning Semantic Grammars with Constructive Inductive Logic Programming. In *In Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822. Morgan Kaufmann, 1993.

John M. Zelle and Raymond J. Mooney. Learning to Parse Database Queries Using Inductive Logic Programming. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI'96, pages 1050–1055. AAAI Press, 1996. ISBN 0-262-51091-X. URL `http://portal.acm.org/citation.cfm?id=1864519.1864543`.

Luke S. Zettlemoyer and Michael Collins. Learning Context-dependent Mappings from Sentences to Logical Form. In *ACL-IJCNLP '09: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2*, pages 976–984, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-46-6.

Zhiping Zheng. AnswerBus Question Answering System. In *Proceedings of the second international conference on Human Language Technology Re-*

search, pages 399–404, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

Elizabeth Zolton-Ford. Reducing Variability in Natural-Language Interactions with Computers. In *Proceedings of the Human Factors Society 28th Annual Meeting*, pages 768–772. The Human Factors Society, 1984.

Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. Jupiter: A Telephone-Based Conversational Interface for Weather Information. *IEEE Trans. on Speech and Audio Processing*, 8:85–96, 2000.