# Chapter 5
# Towards Enhanced Usability of Natural Language Interfaces to Knowledge Bases

Danica Damljanović and Kalina Bontcheva

## 5.1 Introduction

One of the most prominent benefits gained from the emergence of Semantic Web technology is the possibility to access data more efficiently, through the use of ontologies [18]. Querying such data requires using formal languages such as SeRQL [7] or SPARQL [39]. However, the syntax of these formal languages tends to be too "artificial" and complex, especially for domain experts who are unfamiliar with such machine-like languages.

To minimise the learning curve mandatory for the access of such data, many user-friendly interfaces have been developed. Some of them provide a graphical interface where users can browse the data (e.g., Protégé [36]), others offer a form-based interface for performing search whilst hiding the complexity of formal languages, e.g., KIM Platform [31]. The most sophisticated ones provide a simple text box for a query, which takes full-blown questions or a set of keywords as an input, and return answers in a user-understandable form.

According to the interface evaluation conducted in [28], systems developed to support Natural Language (NL) 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 [28]. 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 CHESt [40] – a system about computer history that accepts both keywords and NL queries as input – revealed user's preference for keywords unless the full-blown questions yielded better results. Namely, when asked if they would accept typing full blown questions instead of keyword-based

Danica Damljanović and Kalina Bontcheva
University of Sheffield, Department of Computer Science, Regent Court, 211 Portobello Street, S1 4DP, Sheffield, UK, e-mail: {D.Damljanovic,K.Bontcheva}@dcs.shef.ac.uk

V. Devedžić and D. Gašević (eds.), *Web 2.0 & Semantic Web*, Annals of Information Systems 6, DOI 10.1007/978-1-4419-1219-0_5,

© Springer Science+Business Media, LLC 2010

queries, 22% of users answered positive, 69% said they would accept only if this yielded better results, and 8% of users disliked this option.

The development of accurate Natural Language Interface (NLI) systems is "very complex and time-consuming task that requires extraordinary design and implementation efforts" [28, p.281]. According to [22], a major challenge in 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 information about the domain is structured for computer processing. In the case of Natural Language Interfaces to Knowledge Bases (NLIs to KBs), the domain knowledge is in the knowledge base. The knowledge base is typically created by instantiating classes defined in the domain ontology and relating them as per ontology definitions. Therefore, it is very important to consider the ontology structure and content when building NLIs to KBs.

Another big challenge is building a robust NLI due to the very difficult task of automatically interpreting natural language [11]. NLIs are also typically difficult to port to other domains [11]. *Portable* or *transportable* NLIs are those that can be adapted easily to new domains (e.g., from software engineering to cultural heritage). Although they are considered as potentially much more useful than domain-specific systems, constructing transportable systems poses a number of technical and theoretical problems as many of the techniques developed for specialised systems preclude automatic adaptation of the systems to new domains [22]. Moreover, portability affects retrieval performance: "the more a system is tailored to a domain, the better its retrieval performance is" [28, p.281].

This paper explores how these challenges are addressed by different existing NLIs to KBs, with emphasis on their usability and the overall retrieval performance. The usability of NLIs to KBs is observed from the two aspects: that of the developer who is customising the system for a new domain and that of the user who is using it for querying. More specifically, we are presenting the survey of the state of the art, in order to:

- Compare usability of existing customisation methods used to port NLIs to KBs to new domains
- Compare usability of methods for assisting the user in getting the right answers (e.g., assistance while formulating the query)

By conducting this survey we are expecting to answer the question of how existing NLIs to KBs can increase the performance without a significant additional cost for customisation and further, which methods are efficiently used to assist the user, in order to reach better user–system interaction and consequently better performance.

The paper is organised as follows. In Sect. 5.2 challenges for NLI development are discussed, followed by usability measures used for evaluation of such systems, and the aim of the survey is presented in this paper. Section 5.3 discusses the usability of NLIs from the perspective of application developers in charge of system customisation. We review different NLIs to KBs and present their evaluation results, concluding with a discussion on how the performance of the reviewed systems can be improved (see Subsect. 5.3.9). Next, Sect. 5.4 covers usability from an end-user

point of view – specifically methods for assisting end-users when formulating the query and the impact of such methods on performance. Based on this, we draw recommendations for NLI system design in Subsect. 5.4.6. Overall conclusions are discussed in Sect. 5.5.

## 5.2 Natural Language Interfaces to Knowledge Bases

Natural Language Interfaces 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. From end-users' point of view, natural language is easy to use, considering that it is used everyday in human to human communication [37].

Research in the area of NLIs has been around for more than three decades. Most of the developed NLI systems are created to serve as an interface to relational databases (e.g., [23, 38, 47] and many others). Recently, these evolved towards interfaces to semantically richer data in the form of ontologies/knowledge bases. The third popular group of NLI systems are concerned with accessing semi-structured data from documents. NLIs are also used for dialogue and tutoring systems [11], e.g., a chat bot called Asimov, which answers simple questions in English (asimov-software.com). Lastly, a few NLI systems are developed for purposes other than knowledge access, such as a replacement for a programming language, e.g., see the NLC system [6].

In this paper we focus on NLIs to ontologies/knowledge bases and their usability. If an ontology consists of a finite list of terms and the relationships between them (TBox) [1], a knowledge base is a set of interconnected instances, which are created based on defined concepts and relations from the ontology (ABox). As a knowledge base in this case always relies on an ontology (contains references to the ontology), we will use the term knowledge base to refer to the instantiated ontology and ontology at the same time. Therefore, we say that the focus in this paper is on NLIs to KBs. Such NLIs accept natural language queries as input, generate formal queries behind the scene, execute them against an ontology/knowledge base, and present the results to the user.

### *5.2.1 Habitability*

NLIs were invented to assist the communication between users and computers. However, some studies ([10,32]) 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 [37].

One particular approach to the human–computer communication problem is to keep it brief and use restricted natural language syntax [34]. However, a big challenge when restricting the vocabulary of an NLI system is to consider habitability.

*Habitability* indicates how easily, naturally, and effectively users can use 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* [37]. In other words, habitable languages are languages that people can use fluently [19]. According to [19], a language is habitable if (1) users are able to construct expressions of the language that they have not previously encountered, without significant conscious effort; and (2) users are able to avoid easily constructing expressions that fall outside the bounds of the language. Another way of viewing habitability is the mismatch between the users expectations and the capabilities of an NLI system [5].

### *5.2.2 Usability*

The habitability of an NLI system correlates directly to its usability. According to Brooke [8], *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. First, it is important to identify the system's target users, and second – the tasks that these users will have to perform [8].

NLIs to KBs are used by:

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

Therefore, the usability of NLI systems to knowledge bases should be evaluated from two different aspects: (1) that of the user who is customising the system, and (2) that of the user who is querying the system. According to ISO 9241-11, measures of usability should cover [8]:

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
3. *Satisfaction* – the user's subjective reaction to using the system

*Effectiveness: customisation issues.* As discussed in Sect. 5.3 next, the task of the user who is customising the system is usually to create a domain-specific lexicon. The quality of the output of this task can be evaluated by measuring the performance

of the system when it is ported from one domain to another. As this task does not involve actual end-users, performance can be measured in the abstract through the *coverage* of the system. Given a set of questions collected from a real-world application, the percentage of those that are answerable (e.g., covered by the domain lexicon or/and by the knowledge base) can be summarised as coverage. In other words, coverage here refers to the number of questions that would be successfully answered by the system, assuming that the questions are successfully parsed. The richer the lexicon is, the higher value for the coverage. This term should not be mixed with the *language coverage*, which usually refers to the complexity of questions covered by an NLI system.

*Effectiveness: end-user's point of view.* As we are mainly interested in effectiveness in terms of quality of the retrieved answers; typically NLI systems are evaluated in terms of precision and recall, which are measured adapted from information retrieval (see [35,38]). *Recall* is defined as the number of questions correctly answered by an NLI system, divided by the total number of questions.[1] Excluded from these are often questions with errors or which are ungrammatical or clearly out of the scope of the queried knowledge base [11]. *Precision* measures the number of questions correctly answered divided by the number of questions for which the answer is returned at all [11].

*Efficiency.* Efficiency refers to the level of resource consumed in order to perform the specific task. In other words, efficiency reflects how fast a user can accomplish a task. In case of NLI users, this is usually reported 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 the particular information (the end-user's point of view). In the latter case, the efficiency is usually expressed by the execution time for the queries of various complexity.

*User satisfaction.* There is no unique way of measuring user satisfaction. The most common methodology is to engage users into a session with the system and ask them to fill 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 [8].

### *5.2.3 The Aim and the Scope of the Survey*

The goal of the survey presented in this paper is to explore methods for building habitable and usable NLIs to KBs. We review usability, based on evaluation measures discussed above, from the two aspects: that of the users in charge of customising the system and that of the users who are querying the system. These two aspects are independently discussed in Sect. 5.3 and 5.4.

[1] Sometimes, recall is interpreted as the number of questions answered by an NLI system, divided by the total number of questions.

Section 5.3 reviews existing NLIs to KBs, with regards to the performance and customisation issues. We must emphasise that comparing the performance of the different NLIs to KBs is not a trivial task, due to the variation in evaluation conditions (e.g., ontologies used) 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, the benchmark queries are of different complexity [37]. 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, in our view, these results still provide enough evidence of where the major problems lie and where additional improvements can be made, in order to achieve usable and easily portable NLIs to knowledge bases.

Section 5.4 reviews methods for assisting the end-users when formulating the queries and therefore is mainly concerned with the ways to address habitability. We clearly stated methods used for achieving habitable systems from the end-users' point of view, and based on the evaluation results of various systems we have reported how the application of such methods can affect the retrieval performance.

By conducting this survey we expect to answer several questions, such as: which methods can affect the retrieval performance of NLIs to KBs; if existing methods can be combined; and which method is suitable for which situation/domain; which new methods need to be researched and applied.

## 5.3 Customisation and Retrieval Performance

In this section, we review several NLIs to KBs and report on their performance and customisation issues. To give as objective comparison as possible, we show on which dataset was the system evaluated, how the process of customisation is performed, and the recall and precision values. This section only covers a sub-set of NLIs to KBs, i.e., those that reported evaluation results.

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. 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.

### *5.3.1 ORAKEL*

ORAKEL is an NLI to knowledge bases [11], 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, but not complex answers

**Table 5.1** Natural language interfaces to knowledge bases

| Dataset | System | Precision (%) | Recall (%) | Portability |
|---|---|---|---|---|
| Mooney: geography | PANTO | 88.05 | 85.86 | 0 customisation |
| | Querix | 86.08 | 87.11 | 0 customisation |
| | NLP-Reduce | 70.7 | 76.4 | 0 customisation |
| Mooney: restaurants | PANTO | 90.87 | 96.64 | 0 customisation |
| | NLP-Reduce | 67.7 | 69.6 | 0 customisation |
| Mooney: jobs | PANTO | 86.12 | 89.17 | 0 customisation |
| Software engineering ontology | QuestIO | 82.14 | 71.87 | 0 customisation |
| | AquaLog | 86.36 | 59.37 | 0 customisation |
| 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% | – | – |

to *why* or *how* questions that require explanation. The most important advantage of ORAKEL in comparison to other such systems is its support for compositional semantic construction, i.e., the ability to handle questions involving quantification, conjunction and negation.

ORAKEL has a domain-independent component with a shared *general lexicon*, where for example words such as *what*, *which*, etc. are stored. A part of the *domain-specific lexicon* is created automatically from the domain ontology and is called *ontological lexicon*. Another part of the domain-specific lexicon is created manually and contains mappings of subcategorisation frames to relations, as specified in the domain ontology. Subcategorisation frames are essentially linguistic argument structures, e.g., verbs with their arguments, nouns with their arguments, etc. For example, a verb *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 an ontology relation *writes*. Subcategorisation frames are created by the person in charge of customising the system, who is usually the domain expert. He does not have to be familiar with computational linguistics, although he is expected to have a very basic knowledge of subcategorisation frames. The adaptation of the NLI is performed in several iterative cycles in the user interaction sessions, based on the questions that the system fails to answer. In this way, the *coverage* of the lexicon is being increased each time. The evaluation reported in [11] indicates that users preferred creating the lexicon during these interaction sessions, rather than from scratch.

In the user study carried out in [11] the question was if it is feasible for users without expertise in NLP 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 persons had to customise the lexicon, while the remaining 24 users who did not have any background knowledge in computational linguistics received brief instructions for

the experiment: the scope of the KB was explained to them and they were asked to explicitly say if the received answer was correct or not; each user had to ask at least ten questions.

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 min of training on the software (FrameMapper tool [11]) and 10 min of 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 min. In the first round they created the model from scratch, while in the second round they were presented with those questions that the system had failed to answer after the first round of four sessions with different users. Overall, users B and C had 1 h each to construct the lexicon.

The results showed that querying system that used lexicons created by users B and C gives comparable precision and recall to that of using the lexicon created by the 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 user A created the lexicon (53.67%). Precision was in the range from 80.95% (user B) to 84.23% (user A). The customisation system of ORAKEL is designed so that in each iteration, the created lexicon is more accurate and thus gives better performance. Consequently, the more time users spend customising the system, the better the performance of the system is.

### *5.3.2 AquaLog*

AquaLog [33] is a portable question-answering system, which takes queries expressed in natural language and an ontology as input and returns answers drawn from one or more knowledge bases, which instantiate the input ontology with domain-specific information. With a controlled language, such as that used by AquaLog, users can create factual queries beginning with *what*, *which*, *who*, and the like. The types of supported queries are classified into 23 groups. Questions not belonging to one of these 23 types will not be answered as this system heavily relies on its controlled language.

Although the customisation of AquaLog is not mandatory (except providing the URL of the different ontology), it can increase the performance of the system [33]. 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 that represent a location such as *City* and *Country*; similarly, *who* needs to be associated with, e.g., classes *Person* and *Organisation*. Additionally, it is possible to add the so-called *pretty names* to the concepts or relations in case that the term that is used when referring to a concept is not in the knowledge base. 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 [20] for extending the system vocabulary.

During evaluation reported in [33], ten users who are not familiar with the KMi knowledge base[2] or AquaLog generated questions for the system. 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 were handled correctly [33]. However, this includes seven queries with conceptual failures that happen when the ontology does not cover the query (e.g., the ontology is not designed properly, lack of appropriate relation or term to map with, or having instances instead of classes) and 10 questions for which the answer was not in the knowledge base.

To evaluate portability, AquaLog was also trialed 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, and then synonyms for several ontology resources were manually added. As the authors point out in [33], 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] However, the lack of knowledge was not the only cause of low performance, as many problems arose due to the *problematic* ontology structure, which is designed so that it contains a lot of restrictions over properties. To be handled properly by AquaLog, the ontology should have simpler hierarchy structure; also, the terms in a query should be related by no more than two direct relations. For example, if the query would be *which cities are located in Europe*, *cities* might refer to the ontology class *City*, whereas *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 would not be handled by AquaLog. However, if in this chain *County* would not exist, and there would be direct relation between *City* and *Country* (located in), the query would be processed and answered as the number of relations between the terms *City* and *Europe* (as a continent) is 2. In addition, all resources should be accompanied by labels inside the ontology [33].

### *5.3.3 E-Librarian*

E-librarian [40] system accepts a complete question in natural language and returns a set of documents in which a user can find the answer. A dictionary with only domain-specific words is designed and used instead of external sources such as

[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 KMi semantic portal: http://semanticweb.kmi.open.ac.uk

[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 this paper.

WordNet. There is no evaluation on how expensive it is to build this dictionary; however, it needs to be built manually [40].

The E-librarian service was applied in two applications: one is CHESt – about computer history, and the other is about fraction 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 right answer for 97% of the questions; however, the paper does not present sufficient information on the complexity of those questions.

### *5.3.4 CPL*

Computer Processable Language (CPL) [14] 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 h of training individually, followed by 1 week using the question-answering system. Our understanding is that the domain knowledge was created using the CPL language; however, in [13], there is no information about 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.%[5] This low figure is due to the fact that 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 failures revealed that one-third was 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), so the question was unanswerable. Another third was because the knowledge base did not have an answer and the last third was caused by mistakes of the CPL interpreter that misinterpreted CPL English, so the system failed to find the solution.

### *5.3.5 PANTO*

PANTO [49] is a portable NLI to Ontologies. From [49] there is no evidence of what types of questions are supported, but as they claim that the system correctly parsed 170 questions taken from AquaLog's Web site 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

[5] 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 the questions for which the answer was not in the KB before calculating precision.

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 system performance.

PANTO was evaluated with test data provided by Mooney,[6] which have been used widely 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, although it contains relatively simple queries.[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. They also do not comment on customisation issues and if the system was customised prior experimenting with three different domains.

### *5.3.6 Querix*

Querix [30] is another ontology-based question answering system that translates generic natural language queries into SPARQL. However, [30] does not make it clear what types of questions are supported by the system. When Querix was evaluated on 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 majority of unanswered queries required handling of negation, which is not supported by Querix. As the system vocabulary is derived from the ontology vocabulary, there is no need for customisation. The downside of this approach is that the quality of the ontology strongly affects the system's performance.

### *5.3.7 NLP-Reduce*

NLP-Reduce is a naive domain-independent natural language interface for the Semantic Web [29]. It accepts full sentence queries, sentence fragments, or keywords in a text field. However, the performance of the system differs when used with different knowledge bases. This indicates that the quality of data and the complexity of the queries on these knowledge bases is not always the same and that the performance of the system relies on it.

---

[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.

### 5.3.8 QuestIO

Similar to NLP-Reduce, QuestIO (Question-based Interface to Ontologies) [17] is quite flexible in terms of complexity and syntax of the supported queries. Both keyword-based searches and full-blown questions are translated to SeRQL (or SPARQL) queries and executed against the ontology in order to return answers to the user. Customisation of this system is performed automatically from the ontology vocabulary. In evaluation reported in [17], QuestIO and AquaLog systems were trialed with the GATE knowledge base,[8] which contains data about GATE source code, documentation, manuals, and the like. A set of questions was extracted from the GATE mailing list where users are enquiring about different GATE components. None of the two systems were customised for this experiment [17]. Reported results are used to calculate precision and recall shown in Table 5.1.

### 5.3.9 Summary and Discussion

Most of the mentioned systems rely on lexical matching from the ontology. Few of them use external sources to extend the vocabulary such as WordNet. However, the more technical the domain gets, the less is the chance 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 [12]. That is why domain lexicons, which contain only domain-specific vocabulary, tend to be also used by systems such as E-Librarian or ORAKEL (see Figure 5.1). Manually engineering a lexicon as in the ORAKEL system certainly represents an effort, but it allows to control directly the quality and coverage of the lexicon for the specific domain [12]. Moreover, it has been shown that the more time users spend customising the system, the better performance.

If we accept the fact that "there is no free lunch" [12], we then have to accept that, in order to build a NLI to KB with a reasonable performance, while not affecting portability, the system needs to be customisable easily by their users.

However, as we saw that the performance of the systems can be degraded by the problematic ontology structure (see Subsect. 5.3.2), there is potential in avoiding system failures caused by the ontology design. Moreover, scope of the knowledge base (e.g., the number of ontology resources defined) can affect the overall coverage of the system. Experts for customising NLI systems usually have to manually add descriptions or labels to the relevant terms (e.g., ontology resources). If an ontology would be created so that each concept and relation is accompanied by a human understandable label or description, the automation of domain-specific knowledge creation would be feasible, as during the automatic processing of the knowledge base, all human understandable text attached to the ontology resources would be processed and added to the lexicon.

[8] http://gate.ac.uk/ns/gate-kb

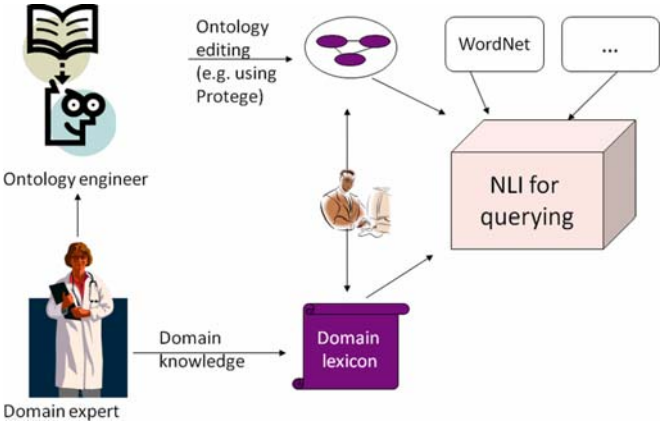


**Fig. 5.1** Process of creating a domain lexicon manually, using an ontology

On the other hand, creating formal data is a high initial barrier for individuals wishing to create ontologies with existing ontology authoring tools such as Protégé as these often require specialist skills in ontology engineering. Therefore, using Natural Language for ontology authoring might be a solution. CLOnE – *Controlled Language for Ontology Editing* ([21, 45]) allows users to design, create, and manage information spaces without knowledge of complicated standards (e.g. OWL) or ontology engineering tools. CLOnE is implemented as a simplified natural language processor that allows the specification of logical data for semantic knowledge technology purposes in normal language, but with high accuracy and reliability. The components are based on GATE's existing tools for Information Extraction and Natural Language Processing [15]. CLOnE is designed either to accept input as valid (in which case accuracy is generally 100%) or to reject it and warn the user of their errors [21]. Many systems similar to CLOnE have been developed with the idea to enable ontology authoring using natural language (e.g., ACE [26], Rabbit [24]).

If the domain expert who is in charge of customising the system uses NLIs for ontology authoring instead of using the tools for customisation (see Figure 5.2), the time for training and learning language-specific terminology can be reduced. For example, if an NLI for ontology authoring allows construction like *Who is usually referring to a person* that will add an additional *label* to the class *Person*, and in this way, the approach used by some of the presented systems (AquaLog, PANTO) for manually customising the lexicalisations of ontology terms would be eliminated. Consequently, the system will "know" that when the user starts a question with *who*, it needs to be associated with a person.

However, in order to create NLIs to KBs with reasonable performance, not only quality customisation is essential, but there is a need to assist end users in the process of query construction. The next section discusses existing methods and how they can affect the overall performance of the system.

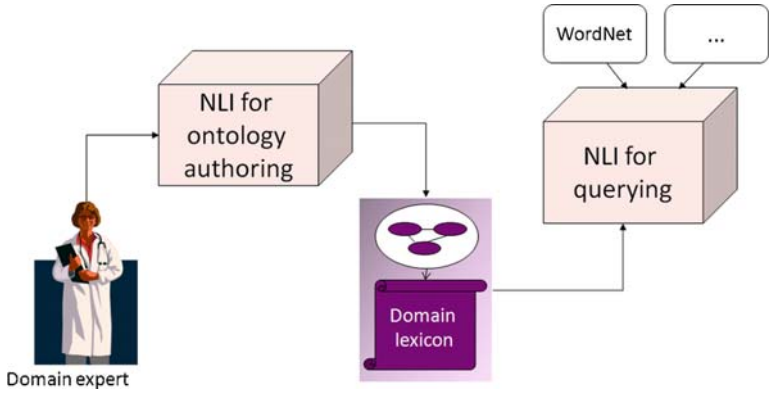


**Fig. 5.2** Automated process of creating a domain lexicon from an ontology

## 5.4 Enhanced usability of Natural Language Interfaces: end-users' point of view

This section reviews methods for assisting the end-users during the search process with NLIs to KBs.

According to the traditional information retrieval, during the search process, the user poses a query based on an existing information need, and after retrieved results are shown, decides to stop or to reformulate the query in a way that promises to improve the result [44]. This is repeated until the "perfect" answer is found. As this traditional model is adequate only for simple cases, a so-called berry-picking model [2] 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 [44]. Although different users behave differently during the search process, it has been shown that majority prefer interactive methods, where the system performs the search, gives the feedback to the user, and lets him decide about the next steps [44].

In the context of NLIs to KBs, from the point of view of end-users, the search process is very similar. The main difference is related to the system design as a big challenge is to address habitability problem. One of the ways to address this problem is to support simple and explicit semantic limitations [19]. One way to achieve this is by restricting the supported vocabulary and grammar.

### *5.4.1 Vocabulary Restriction*

A Controlled Language (CL) is a subset of a natural language that includes certain vocabulary and grammar rules that have to be followed. On one hand, a CL provides

a simple way to retrieve data without extensive training for the end-user, whilst on the other has less expressiveness than the formal languages typically used for accessing structured data [17].

The biggest challenge when designing a controlled language is restricting the natural language so that it still remains intuitive and does not require much training for the end-user. However, applications in industry prove that, actually, CLs can be learnt and used in practise. For example, AECMA Simplified English [48] has been used by the aviation industry since 1986.

Another example is from CPL's evaluation [13], which found that, although users have to be very familiar with CPL in order to use it correctly, they do not have much trouble working with its grammar restrictions, as only a small number of the failures were due to violation of the CPL grammar. Some of the failures were due to the user language: the expressions were not explicit enough for the system (i.e., commonsense facts were not made explicit). The conclusion is that the system would benefit from showing the user the derived query interpretation and any mistakes made. As it is pointed out "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" [14, p.510].

According to [37], 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 and allowing a smooth exchange of information between humans and computers. This exchange can be improved by moving NLI systems towards conversational systems, which means that the system should provide the means of giving feedback to the user, by showing its interpretation of the user's query, so that the user can validate or reject it. Having a limited vocabulary, coupled with a feedback mechanism, means easy training from the end user's point of view [50].

As shown in Figure 5.3, to design habitable NLI system, the system's vocabulary has to be aligned to that of the user. In this paper we will discuss the effect of *feedback* (Sect. 5.4.2) and *guiding the user* (Sect. 5.4.3) through the available questions in order to assist this adaptation (red circle). In addition, as was discussed in Sect. 5.3, 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 (Sect. 5.4.4). Once the user is familiarised 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 (yellow circle). Methods for assisting the user in that adaptation are those that are used to solve *ambiguity* problem and are discussed in Sect. 5.4.5.

An alternative approach to restricting the vocabulary is to support both keyword-based and question-based queries. This allows some flexibility in a way that if the user is not familiarised with the full expressiveness of the controlled language, he can try with keywords, while for more advanced users there is option of using

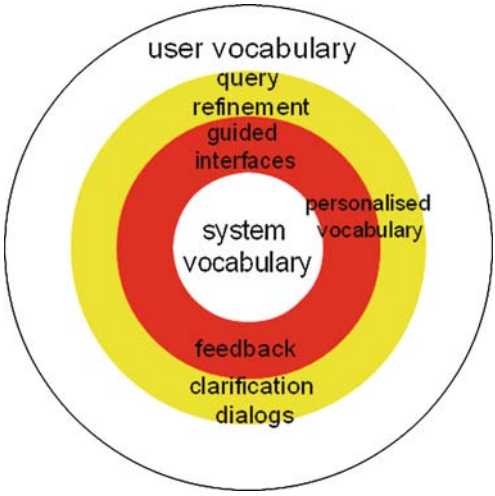


**Fig. 5.3** Synchronizing the vocabulary of the user and the system

full-blown questions. Examples of such systems are QuestIO [17] or NLP-Reduce [29], which would give the same result for both "capital France" and "what is the capital of France?" queries.

### *5.4.2 Feedback*

Showing the user the system's interpretation of the query in a suitably understandable format is called feedback. Several early studies ([41,50]) show that after getting a 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 that are *understandable* to the system.

In the evaluation of Querix and three other interfaces for semantic Web [30], this system was preferred to all 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* [30], 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 [30].

The main drawback of controlled languages is their rather steep learning curve. For example, in order to formulate correctly questions using CPL, users need to know "a bag of tricks" [13]. That is one of the reasons why in CPL an interactive process of question-asking was introduced. After the user poses the question, their

*Advice System* detects CPL errors and returns reformulation advice. There are 106 different advice messages triggered when, for example, 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 the number [13]. The feedback is not using the input text from the user, but rather detecting the error and giving an advice from a static list of feedback sentences. As the authors point out in [13], automatic rewording would be very challenging, especially with longer, complex sentences. In addition to the Advice System, an *Interpretation Display System* is applied, which shows the user how the question is interpreted by the system. 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 interpretation by the system again. In their evaluation with users in [13], this graphical representation was chosen as the most useful.

Although it might be annoying for users, it is not unusual that systems fail to answer the question, due to an unsupported query syntax, even though that same query could be answered if re-formulated. Adding support for extra linguistic coverage is not always easy due to the need to balance expressivity with ambiguity [33]. For instance, the evaluation of AquaLog on the KMI ontology [33] 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 controlled 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 from [13] revealed that users rarely "got it right" the first time. The number of attempts of reformulating the query by the user, before either getting a satisfactory answer from the computer, or giving up, was counted. In physics and chemistry, this number was 6.3 and 6.6, respectively, as the questions were usually very complicated, while for biology the number of reformulated queries was 1.5, as the most common questions where very simple, such as "what is an X?," in contrast to the "story" questions as they call them 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 question 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 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).

### *5.4.3 Guided Interfaces*

According to [3], a major problem with query interfaces is how to guide the user in what queries are supported by the system. Users need knowledge about what it is possible to ask in a particular domain. In [3], relations between concepts are used to assist users by expressing what is possible to ask about the concept which is typed in – this way only meaningful questions can be posted.

According to Bullock [9] 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 [3], Description Logic (DL) is used to help supporting flexible querying and navigation through the information space, by using the tools for manipulation and construction of DL expressions or queries. These tools are driven by constraints known as *sanctions*, which are added to the DL model and which describe the meaningful compositions that 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 topic by a more specific query, and *sibling replacement* – replacing subqueries with sibling concepts. All these manipulations are controlled by sanctioning, ensuring that only reasonable queries are built.

Ginseng [4] is a guided input natural language search engine for the semantic Web. This system allows access to knowledge bases in OWL through NL. The evaluation of Gingseng [4] reported 92.8% precision and 98.4% recall, which indicates that, although the user is limited in the way questions can be asked, this is counterbalanced by high performance–thanks to the offered support. The evaluation of its descendant GINO [5] with six users proves that the use of guided entry overcomes the habitability problem that hampers users' ability to use most full NLIs. The GINO system offers guidance to the user as they formulate a quasi-English query 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 autocompletion. Traditional autocompletion 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 [25]. The proposed *semantic autocompletion* is described in [25] and applied in information retrieval, specifically for multi-faceted search. For example,

the semantic portal MuseumFinland[9] uses semantic autocompletion on request. The search keywords are matched not only against the actual textual item descriptions, but also the labels and descriptions of the ontological categories by which they are annotated and organised into view facets. As a result, a new dynamically created facet is shown on the user request and it contains all categories whose name or other configurable property value, such as alternative labels, matches the keyword. These categories describe the different interpretations of the keywords and the roles with respect to the search items. For example, if the user types in *EU countries*, the system would show list of countries from which the user can choose.

*Summary* To familiarise the user with the system's vocabulary and capabilities, methods for guiding the user through the space of allowed questions could be used. On one hand, the user is limited as the number of questions is limited, but on the other, the performance of such a system is rather high. This means that once the user formulated the query, it is very likely that he will get the answer. A more flexible option is the use of semantic autocompletion, so that users can choose, rather than know the names and type them in. This approach, contrary to fully guided interfaces, leaves the freedom to the user. On the other hand, if the user's input is not fully controlled, the habitability problem could arise nevertheless.

### *5.4.4 Personalised Vocabulary*

As it has been discussed in Sect. 5.3, many NLIs to KBs use external vocabularies such as WordNet in addition to the domain lexicon. However, the vocabulary of the user could be a good source for extending the system vocabulary, as non-known words could be learned by the time, and used to enrich the lexicon and vocabulary used by the system.

The AquaLog [33] 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 Sect. 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 they noticed that when referring to the relation *works-for* users use words like: *is working*, *collaborates*, *is involved in* [33]. When 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

[9] http://www.museeosuomi.fi

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 [33]. First, AquaLog is trialed 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. For example, if the term in the query is a name *Peter*, the user would have to choose in the first iteration from the list of people with names *Peter Scott*, *Peter Whalley*, etc. Finally, the number of queries that require two or three iterations are dramatically reduced with the use of the LM system.

In conclusion, AquaLog LM can improve the performance even for 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 to 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 vocabulary, as well as the domain lexicon that is created individually for each domain, the user-centric vocabulary can play a significant role in increasing the performance of the system over time.

In addition to maintaining the user vocabulary, the AquaLog's approach can be extended to allow users to see and modify the created lexicon at any time. Moreover, in cases when the system cannot offer any options based on the existing user-centric vocabulary, the vocabularies of other users could be used. For example, if the user A asks "Who works for the University of Sheffield?," the system can recognise *The University of Sheffield* as an *Organisation*, and *Who* as 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 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. If now the 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. In an ideal case, the users A and B should be recognised as *similar*. For determining similar users, user profiles need to be modeled. However, modeling user profiles requires a good understanding of the user interests, needs and behaviour as well as understanding the domain knowledge. According to

the number of systems that use ontologies to model user profiles (e.g., [16]), it is likely that their nature has a great potential in creating quality user profiles.

Another direction of improving the personalised feature is related to the presentation of options: 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 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 dependently on the domain.

### *5.4.5 How to Deal with Ambiguities?*

Although controlled languages reduce natural language ambiguities to some extent, some issues, specific to 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 in which one the user is interested in. The way this problem is usually solved is (1) using heuristics and ontology reasoning to implement ranking algorithms to solve ambiguities automatically or/and (2) by asking the user for clarification (clarification dialogues). In cases when the cause of ambiguity is a vague expression of the user information need, query refinement can be used to improve the system performance.

*Automatically solving ambiguities.* E-librarian [40] system uses focus function algorithm in case of ambiguities. A focus is a function that returns the best interpretation for a given word in the context of the complete user question. If there is more than one best interpretation, they are all shown, although the experience with the system revealed that the users generally enter simple questions where the disambiguation is normally successful [40].

OntoNL is an ontology-based natural language interaction generator for multimedia repositories [27]. 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.

In QuestIO [46], relation ambiguities are resolved automatically, based on *string similarity metrics* (comparison of the user input and the name of the ontology relation) in addition to the *position of the relation, its domain and range classes inside ontology hierarchy*. The more specific the relation, its domain and range classes are, the better chance that it will be ranked high.

*Clarification dialogues.* In case of ambiguities Querix [30] send 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 from the two.

Similar to Querix, the AquaLog system [33] relies on clarification dialogues when ambiguity arises. In comparison to Querix, AquaLog is backed by the learning mechanism discussed above (see Sect. 5.4.4), so it saves the results for future sessions.

In general, clarification dialogues can help users resolve ambiguities caused by the content inside the repository. However, if the suggestions provided by the system are not satisfactory, it is likely that the user's need was not expressed precisely, which is the main pre-requisite for retrieving relevant information from the knowledge base (see Figure 5.4).

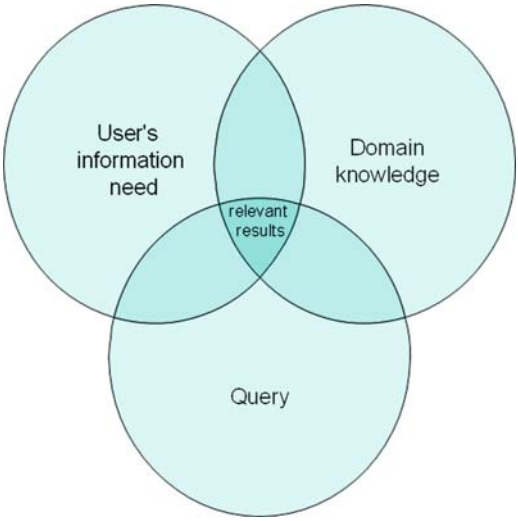


**Fig. 5.4** Retrieving relevant result with NLI systems

According to [42], there is usually a gap between the information need and the query which is 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* [43], can be reduced by the process of query refinement.

*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 [42]. Refining usually means adding more constraints to the query, until the quality of the results corresponds to the user expectation.

Librarian Agent [42] – a system created to replace the human librarian when helping users to find the appropriate books in the library – uses the query refinement technique proposed by Stojanovic [43]. The agent measures query ambiguities regarding 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*, which are implicitly induced by analysing the user's behaviour. Modeling user's need is not trivial especially when users are anonymous as the model of the user behaviour has to be developed implicitly, i.e., by analysing *implicit relevance feedback* whose main idea is to infer the information need by analysing the user interaction with the portal [42].

The agent further defines the neighbourhood of the user's query, which is identified by the query constraints and the ambiguity of 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.* When 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 query refinement process is treated as the process of moving through the query neighbourhood in order to decrease its ambiguity regarding user's need [42].

In Librarian Agent, the ontology lexical layer contains about 1,000 terms, and the information repository (KB) contains about 500 information resources (Web pages about persons, projects, etc.). Each information resource is related to an instance in the ontology. The query refinement system is implemented as an additional support in the searching process so that it can also be switched off. When it is switched on, the user gets the query neighbourhood after posting the query.

For evaluation the authors selected 20 questions. They 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." Six computer science students with little or almost no knowledge about ontologies or the domain, and with no knowledge of the system, were asked to retrieve resources for 10 questions in one session, using the two retrieval methods. Users were asked to explicitly confirm when they get relevant result.

For each search they considered four measures: 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 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 to do 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.

Deriving all possible calculations from available sources could assist in solving the ambiguity problem without user's help. However, it is important to show these decisions to the user in an understandable way as they might not be satisfied with the system's decision and wish to change it. Letting the user choose and express weights of relevant topics/items is a good idea, as this gives users the power of controlling the system's output.

Clarification dialogues are good solutions if they can help system understand what the user is aiming at, whereas in cases of imprecise information needs, query refinement is likely to be a good solution. However, it is important to observe the user, their actions and behaviour during the process of refinement.

### *5.4.6 Summary and Discussion*

Design of habitable NLI systems includes adapting the system vocabulary to that of the user. This adaptation minimises required users' training, which in ideal cases can be eliminated. Methods to achieve this are as follows:

*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, if the user uses words that are "unknown" to the system, feedback can be combined with *clarification dialogues* where the user is prompted by the system provided suggestions. The user's selected option can be further used to build a *personalised vocabulary* of the user. In cases when the system is not able to interpret the type of a question, the user could be prompt by the suggestion, e.g., the list of the supported questions for specific domain. However, the inspection of the queries posted by the user for the specific domain might be useful when defining a way of giving the feedback.

*Guided interfaces.* For small domains, it is possible to provide guided interfaces to the user, which are fully controlled by the system. This means that the user does not have the freedom to enter a query of any length and form. However, as the performance of such systems can be high, and the habitability problem can be eliminated, this might be a preferred solution for domains for which the set of questions that could be asked is limited. A more flexible way of guiding the user is by showing autocomplete options.

*Personalised vocabulary.* NLI systems can benefit from designing a user vocabulary, so that if the term used in a query is unknown to the system, that term can be "learnt" i.e, saved into a user-centric vocabulary that will be used in future. This vocabulary should be *controllable* by the user, so that advanced users can enrich the system vocabulary easily.

After the user is familiar with the system capabilities and the domain, adapting user's vocabulary to that of the system is necessary. However, ambiguity can arise either because of undefined information need, or due to the structure of the knowledge base. Methods for assisting the user include the following:

*Clarification dialogues.* Ontology reasoning as well as heuristics should be used in order to calculate ambiguities. In cases when the calculations are not leading to the one unambiguous solution, the user should be prompt by the *clarification dialogues* to choose from the system provided options.

*Query refinement.* Deriving similar queries, more specific and more generic queries can help users understand the scope of the system and might help in expressing the need more precisely. Ontologies play a significant role in predicting the query refinement process, e.g., defining a set of similar queries, as they contain semantics of the concepts and relations between them.

*Controlling the relevance.* Allowing users to define what does it mean relevant and also allowing them to assign relevance of retrieved results.

*Ranking suggestions.* Ranking popular and relevant queries and suggesting them before any other queries. By relevant queries we assume those that users specified to have satisfying results. Consequently, this means that the users have to be able to define what does it mean to be relevant.

*Defining similarity.* Although NLI systems should have the default measures to express similarity between concepts, it would be great to allow users to define what is similar in cases they want to dig deeper into the power of expressiveness, or in cases they want to have more control over the system output.

All of these five methods 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. Using "semantic web" language for creating user profiles would require the following:

1. *Creating domain space.* creating 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 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 practise, this would mean defining relations between concepts from the domain and user space, i.e., domain and user ontologies.

## 5.5 Conclusion

We have reviewed different NLIs to Knowledge Bases and their usability from the point of view of (1) users (developers) who are customising the system, and (2) end-users who are querying the system. Although systems that require zero customisation have reported high performance, the main concern is the strong dependence of

performance on the quality of the ontology. Additionally, several other systems give better performance when manual customisation is enabled, in order to create associations between the user and the system vocabularies. To address this problem we have proposed using natural language interfaces for ontology authoring, in addition to querying, in order to enhance the quality of the data in the ontology and also to eliminate the process of manual customisation.

Based on the classification of methods used for assisting the users when formulating the queries, we have drawn the recommendations for building habitable and usable NLIs to KBs, from the end user's point of view. These recommendations contain the answer on questions such as when the certain method should be used, which methods could be combined with others and for which domain which method is suitable.

Comparison of NLIs to KBs in this paper reveals that there is the need to standardise the evaluation methods of such systems. Different systems use different datasets (e.g., ontology size), questions of various complexity, and even different evaluation measures. For example, in one case measuring recall might consider only correct queries (e.g., ORAKEL), while in the other case it might be the number of queries for which the system generated an output (e.g., PANTO). The third group often refers to the number of *answered queries* as performance (e.g., AquaLog), although it is not clear if the answers are correct. Moreover, in some cases, queries that return no results due to the knowledge base not containing the answer are counted as correctly handled and as such contribute to the overall performance of the system (e.g., AquaLog). From the evaluation results reported in such way no reliable conclusions can be made with regard to comparison of such systems.

To obtain a progress in the field of NLIs to KBs there is an emergent need to obtain the clear methodology of how these systems need to be evaluated, so that it would be feasible to compare how a method used in one system can affect the performance in another. Only then, when these systems follow the same rules and report on the same measure (evaluated with the same set of queries, the same knowledge base, and under the same conditions) the clear and precise conclusions of how to work on improvement of such systems can be reliable.

**Acknowledgments** This research is partially supported by the EU Sixth Framework Program project TAO (FP6-026460).

## References

1. Grigoris Antoniou and Frank van Hermelen. *A Semantic Web Primer*. MIT Press, 2nd edition, 2008.
2. Marcia J. Bates. The design of browsing and berrypicking techniques for the online search interface. *Online Review*, 13(5):407–424, 1989.
3. S. Bechhofer, R. Stevens, G. Ng, A. Jacoby, and C. Goble. Guiding the user: an ontology driven interface. *User Interfaces to Data Intensive Systems, 1999. Proceedings*, pages 158–161, 1999.

4. A. Bernstein, E. Kaufmann, and E. Kaiser. Querying the semantic web with gingseng: a guided input natural language search engine. In *15th Workshop on Information Technologies and Systems, Las Vegas, NV*, pages 112—126, 2005.
5. Abraham Bernstein and Esther Kaufmann. GINO—A guided input natural language ontology editor. In *5th International Semantic Web Conference (ISWC2006)*, 2006.
6. A.W. Biermann, B.W. Ballard, and A.H. Sigmon. An experimental study of natural language programming. *International Journal of Man-Machine Studies*, 18:71–87, 1983.
7. Jeen Broekstra and Arjohn Kampman. Serql: a second generation rdf query language. In *In Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14, 2003.
8. J. 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.
9. J. Bullock. *Informed Navigation: description Logic Based Hypermedia Linking*. PhD thesis, University of Manchester, UK, 1999.
10. D. Chin. An Analysis of Scripts Generated in Writing Between Users and Computer Consultants. *National Computer Conference*, pages 637–642, 1984.
11. 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.
12. Philipp Cimiano, Peter Haase, Jorg Heizmann, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases âĂŞ the case of the orakel system. *Data and Knowledge Engineering*, 65(2):325–354, May 2008.
13. 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. http://www.cs.utexas.edu/users/pclark/papers/kcap07.ppt.
14. 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.
15. H. Cunningham, D. Maynard, K. Bontcheva, and V. 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.
16. Danica Damljanović 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.
17. Danica Damljanović, 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.
18. J. Davies, D. Fensel, and F. van Harmelen, editors. *Towards the Semantic Web: ontology-driven knowledge management*. Wiley, 2002.
19. Samuel S. Epstein. Transportable natural language processing through simplicity—the PRE system. *ACM Trans. Inf. Syst.*, 3(2):107–120, 1985.
20. Christiane Fellbaum, editor. *WordNet – An electronic lexical database*. MIT Press, 1998.
21. A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Handschuh. Clone: controlled language for ontology editing. In *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, November 2007.
22. 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.
23. Catalina Hallett, Donia Scott, and Richard Power. Composing questions through conceptual authoring. *Computational Linguistics*, 33(1):105–133, 2007.

24. 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.
25. Eero Hyvnen and Eetu Mkel. Semantic autocompletion. In *Proceedings of the first Asia Semantic Web Conference (ASWC 2006), Beijing*. Springer, New York, August 4-9 2006.
26. Kaarel Kaljurand. Writing OWL ontologies in ACE. Technical report, University of Zurich, August 2006.
27. 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 (2007).
28. 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.
29. Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. NLP-Reduce: A âĂIJnaÂÍÄśveâĂİ 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.
30. 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.
31. A. Kiryakov, B. Popov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Semantic annotation, indexing and retrieval. *Journal of Web Semantics, ISWC 2003 Special Issue*, 1(2):671–680, 2004.
32. J. Krause. Natural language access to information systems. An evaluation study of its acceptance by end users. *Information Systems*, 5:297–319, 1980.
33. 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.
34. A. Malhotra. Design criteria for a knowledge based english language system for management: An experimental analysis. Technical report, Massachusetts Institute of Technology, Cambridge, MA (1975).
35. 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, 2001.
36. N.F. Noy, M. Sintek, S. Decker, M. Crubézy, R.W. Fergerson, and M.A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
37. William Ogden and Philip Bernick. Using natural language interfaces. In M. Helander, editor, *Handbook of Human–Computer Interaction*. Elsevier Science, North-Holland, 1996.
38. 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.
39. E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3C recommendation – 15 january 2008, W3C, 2008. URL http://www.w3.org/TR/rdf-sparql-query/.
40. Christoph Meinel Serge Linckels. Semantic interpretation of natural language user input to improve search in multimedia knowledge base. *it – Information Technologies*, 49(1):40–48, 2007.
41. B.M. Slator, M.P. Anderson, and W. Conley. Pygmalion at the interface. *Communications of the ACM*, 29:599–604, 1986.
42. Nenad Stojanovic. On the query refinement in the ontology-based searching for information. *Information Systems*, 30(7):543–563, 2005.
43. Nenad Stojanovic. On the role of a users knowledge gap in an information retrieval process. In *Proceedings of the Third International Conference on Knowledge Capture*, October 2005.
44. 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.

45. V. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva. User-friendly ontology authoring using a controlled language. In *5th Language Resources and Evaluation Conference (LREC)*, Genoa, Italy, May 2006. ELRA.
46. Valentin Tablan, Danica Damljanović, and Kalina Bontcheva. A natural language query interface to structured information. In *Proceedings of the 5h European Semantic Web Conference (ESWC 2008)*, Tenerife, Spain, June 2008.
47. Craig W. Thompson, Paul Pazandak, and Harry R. Tennant. Talk to your semantic web. *IEEE Internet Computing*, 9(6):75–78, 2005.
48. Mike Unwalla. Aecma simplified english. Communicator, Winter 2004.
49. 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.
50. E. 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.