Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 65 (2008) 325–354
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Towards portable natural language interfaces to knowledge bases – The case of the ORAKEL system

Philipp Cimiano [a,*], Peter Haase [a], Jörg Heizmann [b], Matthias Mantel [a], Rudi Studer [a]

[a] *Institute AIFB, Universität Karlsruhe (TH), D-76128 Karlsruhe, Germany*
[b] *ontoprise GmbH, Amalienbadstr. 36, D-76227 Karlsruhe, Germany*

Received 15 July 2007; accepted 25 October 2007
Available online 22 November 2007

Abstract

The customization of a natural language interface to a certain application, domain or knowledge base still represents a major effort for end users given the current state-of-the-art. In this article, we present our natural language interface ORAKEL, describe its architecture, design choices and implementation. In particular, we present ORAKEL's adaptation model which allows users which are not familiar with methods from natural language processing (NLP) or formal linguistics to port a natural language interface to a certain domain and knowledge base. The claim that our model indeed meets our requirement of intuitive adaptation is experimentally corroborated by diverse experiments with end users showing that non-NLP experts can indeed create domain lexica for our natural language interface leading to similar performance compared to lexica engineered by NLP experts.

Keywords: Natural language interfaces; Domain adaptation; Ontologies; Natural language for DKE; Natural language processing

1. Introduction

As the amount of information available globally on the Web and locally in intranets or databases keeps steadily growing, the necessity of mechanisms for effectively querying this information gains importance at the same pace. In fact, it seems crucial to provide end users with intuitive means of querying knowledge as they cannot be expected to learn and use formal query languages such as SQL, which are typically used by programmers. Different paradigms have been proposed in the past for querying information collections, among them *form fillin*, *query-by-example* or *menu-based approaches* (see [50]), as well as natural language interfaces (NLIs), either relying on controlled language [28] or on more or less free language input [44]. While the querying paradigm based on natural language is generally deemed to be the most intuitive from a usage

* Corresponding author. Tel.: +49 721 608 3705; fax: +49 721 608 6580.
E-mail address: cimiano@aifb.uni-karlsruhe.de (P. Cimiano).

0169-023X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2007.10.007

point of view, it has also been shown to be the most difficult to realize effectively. The main reasons for this difficulty are that:

- natural language understanding is indeed a very difficult task due to ambiguities arising at all levels of analysis: morphological, lexical, syntactic, semantic, and pragmatic (compare [1,21]),
- a reasonably large grammar is required for the system to have an acceptable coverage,
- the natural language interface needs to be accurate, and
- the system should be adaptable to various domains without a significant effort.

With the wide availability of cell phones and PDAs, the importance of intuitive ways of interacting with electronic devices has grown even more. Natural language interfaces are an interesting option to interact with mobile devices due to their limited input and output functionality. Clearly, automatic speech recognition is a crucial component towards leveraging the use of natural language interfaces. In this article we are not concerned with speech recognition, but with the process of transforming a user's question into a formal query which can be answered with respect to an underlying knowledge or database. Nevertheless, it is worth emphasizing that speech recognition systems have nowadays reached a degree of maturity which makes it possible to apply them for interacting with phones or other mobile devices (see for example the recent SmartWeb project, which provides natural language access to the Semantic Web [2]).

In the context of this article, we define a *natural language interface* (NLI) as a system accepting as input questions formulated in natural language and returning answers on the basis of a given knowledge base. It is important to emphasize that in our view a natural language interface goes strictly beyond the capabilities of keyword-based retrieval systems known from information retrieval research [3], which are not able to return precise answers to questions but only to return a set of relevant documents given a keyword-based query.

The ORAKEL natural language interface presented in this article addresses all the above challenges, focusing particularly on minimizing the effort of adapting the system to a given domain. ORAKEL is an ontology-based natural language system in two senses. First, the ontology for a certain knowledge base is used to guide the lexicon construction process. On the one hand, parts of the lexicon are automatically generated from the underlying ontology. But most importantly, on the other hand, the ontology is at the core of the whole lexicon acquisition process in ORAKEL, which is performed by the lexicon engineer to adapt the system to some domain and a particular knowledge base. Second, ORAKEL is ontology-based in the sense that it is a natural language interface which relies on deduction to answer a user's query. The ontology as a logical theory together with the facts stored in the knowledge base are thus exploited by the underlying inference engine to provide an answer, even if it is not explicitly contained in the knowledge base but can be inferred from it. As ORAKEL relies on a well-defined deduction process to answer a query, an important requirement is that the user's question is translated into logical form, in particular into a query which can be evaluated by the underlying inference engine.

In general, the ontology model required by the system for the purposes of lexicon acquisition is rather simple, consisting of concepts, ordered hierarchically in terms of subsumption, as well as (binary) relations together with their corresponding restrictions on their domain and range (compare the ontology model described in [25] for a corresponding more formal definition). In practice, we will however rely on standard ontology models such as the ones provided by languages such as OWL [6] or F-Logic [34]. In fact, for the process of query answering, we will rely on the full expressive power of the logical languages used in the background.

The challenge for natural language interfaces is thus the domain-specific interpretation of the user's question in terms of relations and concepts defined in the schema or ontology of the knowledge base. Thus, parsers which create a generic logical form for a given input sentence will clearly not suffice for this purpose. The challenge is to construct a logical query consisting of domain-specific predicates which can be evaluated with respect to the knowledge base, returning the correct answer as a deduction process. Therefore, it is crucial that a natural language interface is adapted to every different knowledge base it is applied to.

In general, the problem of adapting natural language applications to some specific domain still remains largely unsolved. Different models for customization have been proposed in the natural language processing

(NLP) literature. However, the feasibility of different customization approaches from a user point of view has been rarely investigated. While letting users engineer a complete grammar by hand might be a potential solution, it is for sure not feasible as it can neither be expected that general users have grammar engineering experience nor that they would be willing to make such an effort. Some systems support the user in defining linguistic rules, especially in the context of information extraction systems (compare [22]). In contrast, some researchers have examined supervised approaches in which training data is provided and the system learns domain-specific rules using inductive learning techniques [52]. However, it seems still unclear whether providing training data, i.e. questions with their corresponding queries, is a feasible way of customizing a natural language interface to a specific knowledge base from the point of view of an end user. In general, the feasibility of different approaches for customization has been rarely investigated from a user point of view.

Recently, several approaches have been presented which do not rely on any sort of manual adaptation. These approaches exploit external lexical knowledge, for example in the form of lexical databases such as WordNet [26], to account for syntactic variants. This is for example the case of the PRECISE [44] and AquaLog [35] systems, which essentially rely on lexical matches to determine to which entities in the knowledge base the words in the query refer to. At first sight, these approaches seem superior to an approach as presented in this paper in which a lexicon needs to be explicitly created. Nevertheless, such approaches strongly depend on the quality and coverage of the lexical resources used. Recent work by ourselves [19], in which an approach based on lexical matching is explored, has in fact shown that one can rely less on lexical matching the more technical the domains get. In fact, we can not expect to have the complete lexical knowledge necessary for very technical domains in general resources such as WordNet. Manually engineering a lexicon as in the ORAKEL system described in this article certainly represents a considerable effort, but it allows to directly control the quality and coverage of the lexicon for the specific application as the lexicon is represented declaratively and can be directly updated.

Finally, there are systems which support the user in lexicon acquisition by hiding the linguistic details behind some frontend. The well-known natural language interface TEAM [29], for example, achieves the customization by asking domain experts questions and deriving the necessary linguistic knowledge from their answers. Rosé et al. [46] have recently also presented an approach in which a NLP system is created by users as a byproduct of annotating text segments. However, with the only exception of Rosé et al., none of the above work has examined the question whether typical users of the system are indeed able to successfully perform the customization.

In this article, we explore a model of user-centered lexicon customization which merely requires very basic knowledge about subcategorization frames, but no background in computational or formal linguistics. Subcategorization frames are essentially linguistic argument structures, e.g. verbs with their arguments, nouns with their arguments, etc. As in TEAM, we also assume that a user with general expertise about computer systems will perform the customization, i.e. we subscribe to the hypothesis mentioned in [29]:

> "*A major hypothesis underlying TEAM is that, if an NLI is constructed in a sufficiently well-principled manner, the information needed to adapt it to a new database and its corresponding domain can be acquired from users who have general expertise about computer systems and the particular database, but who do not possess any special knowledge about natural-language processing or the particular NLI.*"

In the ORAKEL system, the main task of the person in charge of customizing the system is to create a domain-specific lexicon mapping subcategorization frames to relations specified in the domain ontology. We present experimental evidence in form of a user study as well as in the form of a case study involving a real-world application to corroborate the claim that our model indeed allows non-NLP experts to create an appropriate domain lexicon efficiently and effectively. We show in particular that the results obtained with lexica customized by non-NLP experts do not substantially differ from the ones created by NLP experts. As the coverage of the lexicon has a direct impact on the overall linguistic coverage of the system, we propose a model in which the lexicon engineer can create the lexicon in an iterative process until a reasonable coverage is achieved. We also provide experimental evidence for the fact that such an iterative lexicon construction model is indeed promising. Furthermore, we also assess the coverage of our system, showing that with a few subcategorization frame types we can indeed yield a reasonable linguistic coverage. Before describing the details of ORAKEL, we first present an overview of the system in the next section.

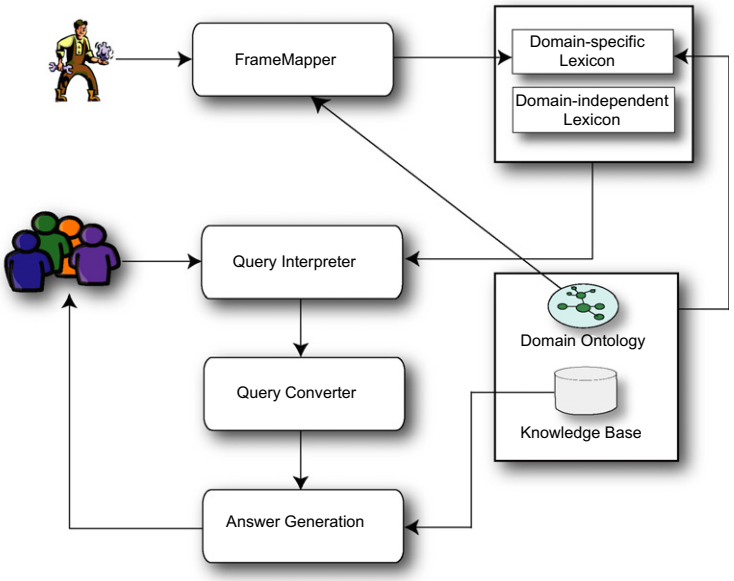


Fig. 1. Overview of the ORAKEL system.

2. Overview of ORAKEL

The input to ORAKEL are factoid questions starting with so-called *wh*-pronouns such as '*who*', '*what*', '*where*', '*which*', etc., but also the expressions '*How many*' for counting and '*How*' followed by an adjective to ask for specific values of an attribute as in "*How long is the Rhein?*". Factoid in this context means that ORAKEL only provides ground facts as typically found in knowledge or data bases as answers, but no answers to *why*- or *how*-questions asking for explanations, the manner in which something happens or causes for some event.

In the ORAKEL system, we assume two underlying roles that users can play. On the one hand, we have *end users* of the system which interact with the system in *query mode*. On the other hand, domain experts or knowledge engineers which are familiar with the underlying knowledge base play the role of *lexicon engineers* which interact with the system in *lexicon acquisition mode*, creating domain-specific lexicons to adapt the system to a specific domain.

The end users ask questions which are semantically interpreted by the *Query Interpreter* (compare Fig. 1). The Query Interpreter takes the question of the user, parses it and constructs a query in logical form (LF), formulated with respect to domain-specific predicates. This logical form is essentially a first-order logic (FOL) representation enriched with query, count and arithmetic operators. The *Query Interpreter* component is discussed in detail in Section 3. The query in logical form is then translated by the *Query Converter* component into the target knowledge representation language of the knowledge base, in particular to its corresponding query language. The overall approach is thus independent from the specific target knowledge language and can accommodate any reasonably expressive knowledge representation language with a corresponding query language. Our system has been so far tested with the knowledge representation languages F-Logic [34] with its query language as implemented by the Ontobroker system [23] and OWL [38] with the query language SPARQL [45] as implemented by the KAON2 inference engine.[1]

The conversion from the logical form to the target knowledge language is described declaratively by a Prolog program. The *Query Converter* component reads in this description and performs the appropriate transformation to yield a query in the target language. So far, we have provided the two implementations for F-Logic as well as OWL/SPARQL. However, our system architecture would indeed allow to port the sys-

[1] http://kaon2.semanticweb.org/.

tem to any query language, in particular the RDF query languages described in [30] or plain SQL to access conventional relational databases. In fact, changing the target language requires a declarative description of the transformation as a Prolog program, but no further change to the underlying system. We describe the process of query translation in more detail in Section 3.3.

The answer generation component then evaluates the query with respect to the knowledge base and presents the answer to the user. Answering the query is thus a deduction process, i.e. the answer to a user's question are the bindings of the variables in the resulting query. Currently, the answer generation component only presents the extension of the query as returned by the inference engine. However, more sophisticated techniques for presenting the answer to the user by describing the answer intensionally or presenting the results graphically are possible. The way of displaying the results in general depends heavily on the application in question and will thus not be discussed further in this article.

As an example, consider the question: "*What is the capital of Baden-Württemberg?*" to a knowledge base containing facts about Germany. This question would be translated into the following internal representation by the *query interpreter*:

$$?x\ capital(baden\_wuerttemberg, x)$$

This internal representation would then be translated into the target query language, e.g. F-Logic by the *query converter*:

$$\forall X \leftarrow baden\_wuerttemberg[capital \rightarrow X]$$

The answer to this query is then generated by evaluating the above query with respect to the inference engine.

We have mentioned already in Section 1 that a crucial question for natural language interfaces is how they can be adapted to a specific domain in order to interpret the user's question with respect to domain-specific predicates. In the model underlying ORAKEL, the lexicon engineer is in charge of creating a domain-specific lexicon thereby adapting ORAKEL to the domain in question. The lexicon engineer is essentially responsible for specifying how certain natural language expressions map to predicates in the knowledge base. For this purpose, we have designed an interface *FrameMapper* (compare Fig. 1) with access to the knowledge base, which supports the lexicon engineer in specifying by graphical means the mapping from language to relational predicates defined in the knowledge base. The result of the interaction of the knowledge engineer is a domain lexicon specific for the application in question. The process of domain adaptation is described in detail in Section 4, while the graphical user interface of FrameMapper is described in Section 5.

Besides the domain-specific lexicon, ORAKEL also relies on a general lexicon which specifies the semantics of closed-class words such as prepositions, determiners, question pronouns, numbers, etc. The semantics of these closed-class words are actually domain independent and specified with respect to elementary or foundational categories as given by *foundational ontologies*. In our ORAKEL system, we rely on the foundational ontology DOLCE [37], which provides fundamental categories such as *physical object*, *agentive physical object*, etc. as well as predicates and relations related to time and space. The latter ones are crucial for representing the semantics of spatial or temporal prepositions.

The general lexicon and the domain-specific lexicon created by the domain expert provide the only sources that ORAKEL needs to answer questions. Both type of lexica are in fact a lexicalized grammar which is used by ORAKEL for parsing but also for constructing the semantics of input questions. Thus, ORAKEL does not need any external grammar or other lexical resources.[2] As the general lexicon is given, the crucial bottleneck is thus the creation of the domain-specific lexicon. An appropriate domain-specific lexicon is crucial for interpreting the user's question with respect to domain-specific predicates. In this article, our focus lies in particular on the adaptation model and adaptation mechanism of ORAKEL. Our aim is to show that, given very rudimentary knowledge about grammar and language, domain experts can indeed successfully adapt ORAKEL to different domains. We also show that an iterative approach in which

[2] The only two exceptions are lists of base forms for nouns and verbs with their corresponding inflected forms which are used by ORAKEL to generate tree families. This is discussed in more detail in Section 4. Further, WordNet is used to provide synonyms for verbs and nouns (compare Section 5). However, this possibility was not exploited in the experiments described in Section 6.

the lexicon engineers modify the lexicon on the basis of failed questions until a reasonable coverage is achieved seems indeed reasonable.

We have carried out experiments on two different domains to corroborate our claim. On the one hand, we have carried out a user study with a small knowledge base containing facts about Germany. On the other hand, we have used a database containing metadata about research publications from British Telecom's – henceforth BT – digital library, which is orders of magnitude larger than the geography knowledge base. Our studies show that ORAKEL can indeed be successfully adapted to different domains in a reasonable amount of time, typically a few hours. The British Telecom case study was especially challenging as ORAKEL had to be modified to scale up to tens of thousands of facts contained in the BT database.

ORAKEL has also a few limitations which are however not of a principled nature. Currently, ORAKEL can neither handle ungrammatical input nor deal with unknown words. On the one hand, as we will see below, ORAKEL assumes a full parse of the input sentence and thus expects the sentence to be grammatical. In case the question is not grammatical it will simply fail and tell the user that it did not understand the question, without giving any further feedback. In case a word is unknown, the system will at least inform the user about which word is currently unknown. While handling ungrammatical input and unknown words is a must for any commercial natural language interface, we have decided to abstract from these issues at this stage as our research is mainly concerned with aspects related to the easy customization and portability of NLIs. Nevertheless, we are confident that ORAKEL can be extended to show a more robust behaviour with respect to ungrammatical input and unknown words and to generate appropriate feedback to the user.

3. Query construction

In this section, we describe how the logical query to the knowledge base is constructed on the basis of a user's question formulated in natural language. In order to make this article self-contained, we describe all the components necessary to understand the ORAKEL system. However, we only describe these components rather briefly and omit most of the technical details of the system. The interested reader is referred to our technical report for details (see [18]).

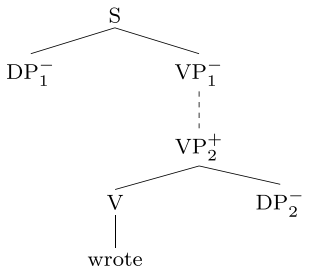
In the next Section 3.1, we first describe the syntactic formalism underlying our system as well as the corresponding parser. Then, in Section 3.2 we describe how a query in our enriched first-order logic (FOL) language is constructed. Section 3.3 discusses how the FOL query can be translated into an appropriate target query language, e.g. into a F-Logic or SPARQL query in our case.

3.1. Syntactic theory and parsing

The underlying syntactic theory of our system is a formalism called Logical Description Grammars (LDG) (compare [40]). LDG is inspired by Lexicalized Tree Adjoining Grammars (LTAGs) [31], which essentially are tree rewriting systems consisting of a finite set of trees associated with lexical items, so-called elementary trees (etrees). The two main operations in LTAG are substitution and adjoining. Substitution can be regarded as a local operation for the insertion of arguments. Adjoining typically folds one tree into another, thereby introducing modifiers or recursively embedding structures, such as clausal arguments.

The structures used in LDG are essentially (descriptions of) trees consisting of nodes labeled with syntactic information as depicted below. An important characteristic of these trees is that they encapsulate all syntactic/semantic arguments of a word. The following tree for *wrote* for example explicitly indicates that it requires a subject (the author) at the DP_1 position as well as a direct object (the written document) at the DP_2 position. The fact that the line between VP_1 and VP_2 is dashed denotes that this dominance relation is not immediate, i.e. some other tree could slip in.[3] Typical trees which could slip in into this position are adverbs, e.g. *often*, or negation particles, e.g. *not*.

[3] Here, DP stands for a *determiner phrase*, VP for a *verb phrase*, V for a *verb* and S for a *sentence*.



In essence, negatively marked nodes correspond to arguments which need to be inserted, while positively marked nodes denote variables to be inserted as an argument.

In the LDG formalism used in ORAKEL, there is only one operation, which consists in identifying positively with negatively marked nodes with each other within one or across trees. Hereby, two nodes can only be identified with each other if (i) they have complementary marks (negative/positive), (ii) they have the same syntactic category, (iii) their feature structures are compatible as well as (iv) syntactic dominance and surface order of words is respected. Feature structures in ORAKEL are in essence flat lists of attribute-value pairs. Two nodes can then only be identified with each other if they have the same value for a common attribute (see below the discussion of the features used in ORAKEL).

As noted above, the verb '*write*' requires a subject and an object. We say that '*write*' *subcategorizes* a subject and an object. It is therefore a *transitive* verb. However, there are not only transitive verbs, but also other types such as *intransitive verbs*, which subcategorize only a subject, *intransitive verbs with a prepositional complement*, *transitive verbs with a prepositional complement* as well as *ditransitive verbs* subcategorizing two objects. We call a verb together with a specification of which arguments it subcategorizes a *subcategorization frame*. Subcategorization frames are central in ORAKEL as they provide the basic structures which a lexicon engineer is supposed to map to domain-specific relations. Subcategorization frames give raise to another central notion: the one of *tree families*. Tree families encapsulate all the different ways in which a subcategorization frame can be expressed and thus capture generalizations of a given subcategorization frame type across words. For example, the tree family of a transitive verb such as '*write*' consists (at least) of elementary trees corresponding to the standard active form, a passive form, a realization as relative clause complementing a noun phrase as well as a form in which the object is extracted and moved to the front of the sentence or question. The different forms allow for example to ask the following questions:

- Who writes/wrote a book? (active)
- Which authors write/wrote a book? (active plural)
- Who did not write a book? (active, negation, auxiliary construct)
- Which book did Tolkien write? (extracted object)
- Which book was written by Tolkien? (passive)
- Who is the author who writes/wrote "The Lord of the Rings"? (relative clause)
- Which is the book which Tolkien writes/wrote? (relative clause with extracted object)
- Which is the book which was written by Tolkien? (passive relative clause)

The above are merely syntactic variants of one and the same subcategorization frame, i.e. the one corresponding to the transitive usage of the verb '*write*'. For a natural language interface to be successful, it does not only have to account for syntactic, but most importantly for lexical variations. Given a relation *inhabitants*(*location*,*integer*) which models the inhabitants of a location (city, state or country) as a datatype property with an integer as range, a user might ask the following questions:

- How many people live in Karlsruhe? (1a)
- In which city do the most people live? (1b)
- Which is the biggest city? (1c)
- Which city has more inhabitants than Karlsruhe? (1d)

On the one hand, the above example questions show that the way people ask for information rarely corresponds straightforwardly to the way information is modeled in an ontology. In particular, very different lexical and syntactic variants can be used to ask for the very same information. For example, to ask for the city with the most inhabitants we can either ask "*Which is the biggest city?*" – thus using a superlative, or "*Which city do the most people live in?*" – using the intransitive verb '*live*' with a prepositional complement introduced by the preposition '*in*', or "*Which is the city with the most inhabitants?*" – using the preposition '*with*' followed by a noun phrase with head '*inhabitants*', or "*Which city has the most inhabitants?*" using a similar construction involving the verb '*have*'.

On the other hand, similar constructions can be used to ask for information which is modeled differently in the ontology. For example, to ask for the number of inhabitants of a city, which is modeled as a *datatype property*, we can ask "*How many people live in Karlsruhe?*", while when asking for the number of rivers which flow through a city, which is modeled through an *object property*, we can ask in the same way – modulo lexical differences due to the different relations involved – "*How many rivers flow through Karlsruhe?*".

This exemplifies in fact that the correspondences between the way we talk about things and the way they are modeled in an ontology are far from straightforward. Thus, language is transparent to the way information is modeled in a given ontology. This shows why the problem of adapting a natural language interface is indeed a non-trivial task. As already mentioned, we support the customization of ORAKEL through a graphical user interface by which users can graphically specify how certain subcategorization frames map to relations (or joins of these) in the ontology. In the background, the system generates all the different syntactic variants as specified in the tree family of the corresponding subcategorization frame. The advantage of such an approach is that the semantics of each word needs to be specified exactly once by associating it with the corresponding subcategorization frames. Thus, all the generated trees from the corresponding tree family feature already the appropriate semantic representation. The generation of the trees, however, remains totally transparent to the end user and lexicon engineer. The lexicon engineer is not even aware of the fact that she/he is associating semantic representations to the subcategorization frames specified.

We have briefly sketched above how the tree family for transitive verbs looks like. It is easy to imagine how the tree families for intransitive verbs with a prepositional complement, transitive verbs with a prepositional complement, etc. look like. In ORAKEL, we also have tree families for adjectives as well as relational nouns. Relational nouns are those which subcategorize a prepositional complement, such as *mother (of)*, *brother (of)*, *capital (of)*. Typically, relational nouns can be used in a form in which the prepositional complement is existentially quantified over, as in "*Which rivers flow through a capital?*". Thus, for relational nouns, ORAKEL also generates variants in which the prepositional complement is not realized syntactically but existentially quantified over (compare [24] for a more deep and formal discussion of this issue).[4]

In the LDG formalism used in ORAKEL, parsing boils down to identifying positively and negatively marked nodes with each other, respecting category information, feature values and surface order of words. The ORAKEL system implements a procedural version of LDG in which parsing proceeds as in typical LTAG parsers in two stages. In fact, we implemented an Early-type bottom-up parser as described in [47]. First, appropriate elementary trees for each word in the input are selected from the lexicon, and, second, these elementary trees are combined to yield a parse of the sentence (compare [47]). In particular, ORAKEL relies on full parsing and does not make any use of partial parsing techniques. Partial parsing techniques would be necessary to process input sentences which are ungrammatical or can not be processed by the parser and would give the system a more robust behaviour. However, robustness with respect to ungrammatical input is not in the current focus of the system as described here. This is certainly an important issue for future work.

In ORAKEL, we have implemented a procedural version of the parsing mechanism inherent in the LDG approach. The parser basically identifies positively and negatively marked nodes respecting:

- the syntactic category of nodes,
- feature values,
- ontological constraints,

[4] A description of the tree family of relational nouns featuring one prepositional complement is given in our technical report [18].

- surface word order, and
- syntactic dominance relations.

The parser is an Early-type bottom-up parser using top-down information as described in [47,48]. It scans and reduces the input string from left to right, traversing the corresponding elementary trees in a top-down fashion. However, the parser can be called a bottom-up parser as it uses the words in the input string to guide the whole process. The interested reader can find the details about the parser in our technical report [18].

3.2. Semantics construction

ORAKEL implements a compositional semantics approach to construct the logical formula corresponding to the input question. Compositional means here that the query to the database or knowledge base – i.e. the semantic representation of the input sentence – is recursively computed on the basis of the meaning of every single word in the input sentence as well as the way the words are connected. Thus, the logical query representing a question is constructed en par with the syntactic analysis of the question. Such an approach requires some sort of syntactic processing grouping words to larger syntactic units and ordering them as trees to guide the recursive computation. This is accomplished by the parser described in the previous section.

The semantics of a sentence is then the semantics of the top node of the elementary tree marked as root and is specified by a FOL-like formula which is translated in a subsequent step to a formula in the target query language via a Prolog conversion program.

The semantic construction proceeds en par with the syntactic construction in a traditional compositional manner (compare [39]). Thereby, each node specifies how its meaning is constructed on the basis of the meaning of its children using the lambda calculus. In ORAKEL we use an extended version of the lambda calculus implemented in Prolog by Blackburn and Bos [8].

A compositional semantics construction approach as implemented by ORAKEL requires relatively rich lexical resources specifying the logical meaning of each word. This is exactly where our user-centered model for lexicon customization fills a gap as the rich semantic lexicon is generated in the background as a byproduct of the interaction of the lexicon engineer with the system's lexicon acquisition frontend, called *FrameMapper* (see Section 5). Details about the semantics of each word remain completely transparent to the user. Indirectly, the lexicon engineer is thus generating a grammar as well as associating logical meanings to words without even being aware of it. We will discuss this process in detail in Sections 4 and 5.

As a short illustrating example, imagine a user asking the question: "*Which river passes through Berlin?*" to a knowledge base containing facts about German geography. The meaning of the diverse lexico-syntactic units in the input can be expressed in functional lambda notation roughly[5] as follows:

Which river	$\lambda P?x(river(x) \wedge P(x))$
passes through	$\lambda x \lambda y flow\_through(x, y)$
Berlin	$\lambda Q Q(Berlin)$

So the semantic representation of '*passes through*' expects two individuals as arguments to be inserted into the appropriate relation *flow_through*. The expression '*which river*' expects some property P which x, a river, needs to fulfill. '*Berlin*' requires some predicate Q into which it can be inserted as an argument.

Given the simplified syntactic structure together with instructions how the semantic expressions are applied to each other in Fig. 2, and evaluating the tree in a standard bottom-up fashion, we would first carry out the functional application

$$\lambda u\ (\lambda Q\ Q(Berlin))((\lambda x\ \lambda y\ flow\_through(x, y))(u)),$$

[5] Roughly as in principle each word should be associated with a semantic representation. We abstract from this for the sake of clarity of presentation.

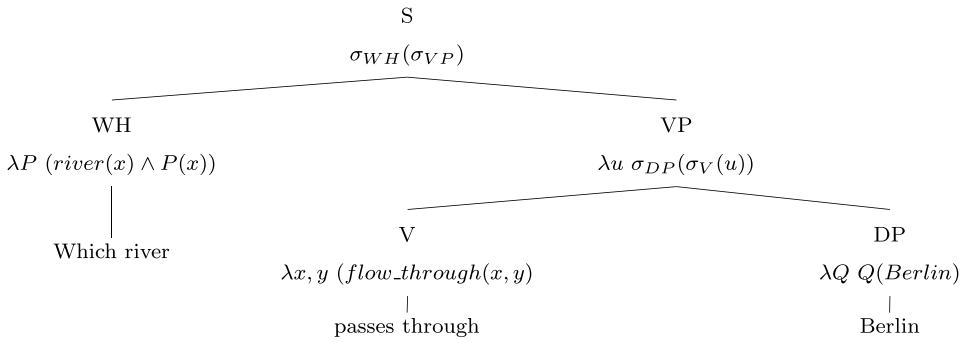


Fig. 2. Syntactic analysis with semantic representations for each word specified according to the λ-calculus and instructions how to combine the different representations with each other.

yielding as semantic representation of the *VP* node

$$\lambda u\ flow\_through(u, Berlin)$$

in which the argument *Berlin* has been correctly inserted. To yield the final semantic representation of the top sentence node *S*, we would carry out the functional application

$$(\lambda P\ ?x(river(x) \wedge P(x)))(\lambda u\ flow\_through(u, Berlin))$$

resulting in the final logical query:

$$?x\ (river(x) \wedge flow\_through(x, Berlin))$$

3.3. Query conversion

In order to increase its flexibility, ORAKEL has been designed to be, on the one hand, domain independent and, on the other hand, independent of the specific knowledge representation and query language used in the background. Domain independence is achieved by separating the general and domain lexica as is typically done for transportable NLIs (compare [29]). The latter one needs to be handcrafted by a domain expert. The independence of the target logical language is achieved by introducing a First-Order-Logic (FOL) language enriched with additional predicates for quantifiers as well as query and numerical operators, which is produced by our semantic analysis component. The question "*Which city do the most rivers flow through?*" is for example represented as follows in our FOL-like query language:

$$?c\exists r, n_1\ city(c) \wedge flow\_through(r, c) \wedge count(c, r, n_1) \wedge \forall c', r', n_2(flow\_through(r', c') \wedge count(c', r', n_2) \rightarrow n_1 \geqslant n_2)$$

In the above formula, the $count(a, b, c)$-predicate is evaluated in such a way that $c(a)$ is the number of elements b standing in the relation expressed in the query grouped by the a's. So in the above query $n_1(c)$ is bound to the number of rivers r flowing through each city c. Queries in this FOL-like language can then be translated to any logical language by a translation component. Hereby, the translation is specified declaratively in Prolog and is thus exchangeable.[6] The Prolog conversion programs essentially specify recursively how the operators of the query language (?, $\exists$, $\wedge$, $\rightarrow$, $count(\ldots)$) are translated into the target query language. The above query is for example translated into F-Logic as follows:

$$\forall C \leftarrow \exists R, N_1 C : city \wedge R[flow\_through \rightarrow C] \wedge count(C, N, N_1) \wedge \forall C', R', N_2(R'[flow\_through \rightarrow C'] \wedge count(C', R', N_2) \rightarrow geq(N_1, N_2))$$

While all the queries specified in our FOL-like query language can be translated into F-Logic, this is not the case for the SPARQL language as implemented in the KAON2 system. Currently, the SPARQL implemen-

[6] The Prolog code for the conversion into F-Logic and SPARQL can be found at http://www.cimiano.de/orakel.

tation behind the KAON2 system supports only conjunctive queries such that the above query would not be translatable to SPARQL in our system.

A direct translation to some target formalism as performed in [12] is also possible, but clearly such an approach is not as flexible as the one pursued within ORAKEL. Currently, our system supports two formalisms used in the Semantic Web, the Web Ontology Language (OWL)[7] with the query language SPARQL[8] as well as F-Logic as ontology language together with its corresponding query language [34]. The ontologies essentially provide the schema for the knowledge base and thus the concepts and relations relevant for the domain in question. This system design allows to port our system to any domain and any (reasonably expressive) logical formalism with a query language. The only requirement on the language is that it provides extralogical predicates for counting and for numerical comparisons.[9]

4. Domain adaption

In our system, we pursue an approach in which the domain lexicon is constructed in interaction with the lexicon engineer, whose task is to map relations in the knowledge base to appropriate verb and noun subcategorization frames, adjectives, etc. Before explaining in detail the underlying model which allows a lexicon engineer to create a domain-specific lexicon and thus customize the system to a certain knowledge base, it is important to mention that the overall lexicon of the system has a bipartite structure consisting of:

- a *domain-independent lexicon*, containing the semantic representations for determiners (*a*, *the*, *every*, *most*, ...), wh-pronouns (*who*, *what*, *which*, *where*) as well as certain spatio-temporal prepositions (*on*, *in*, *at*, *before*, ...),
- a *domain-specific lexicon*, defining the meaning of verbs, (relational) nouns and adjectives occurring in the domain, and containing lexical entries and the semantics of instances and concepts, which are typically represented linguistically as proper nouns and nouns, respectively.

The *domain-independent lexicon* is, as the name suggests, independent of any domain as it specifies the meaning of words occurring in several domains and with a constant meaning across these. This is the case for determiners, wh-pronouns and prepositions. The semantic representations of the words in this domain-independent lexicon thus make reference to domain-independent categories as given for example by a foundational ontology such as DOLCE [37]. This assumes obviously that the domain ontology is somehow aligned to the foundational categories provided by the foundational ontology. The obvious benefit of such a modular design of the lexicon is that the meaning of closed-class words such as prepositions, wh-pronouns or determiners are available independently of any domain ontology and need not to be specified for every different domain the system is applied to. A more detailed description of the general benefits and rationale of such a modularized approach can be found in [16].

The *domain-specific lexicon* is partially derived in an automatic fashion from the domain ontology loaded into the system without any manual intervention. In fact, the system reads in all the concepts and instances of the ontology and relies on their labels to generate appropriate grammar trees representing these. Obviously this assumes the availability of labels for each concept and instance in the ontology. However, in general it is regarded as good practice to include such labels into the ontology to enable human inspection. For the generation of nominal trees on the basis of concepts, we use a lexicon with morphological information to generate the appropriate plural form. This lexicon was generated on the basis of Tree Tagger's tagger lexicon [49]. For illustration, Fig. 3 shows the elementary trees which are automatically generated from the instance *Saarbrücken* as well as the concept *country*.

The other part of the domain-specific lexicon component is generated by the lexicon engineer by mapping verbs, adjectives and relational nouns to corresponding relations specified in the domain ontology. The domain-specific lexicon is actually the most important one as it is the one specifying the mapping of linguistic

[7] http://www.w3.org/TR/owl-ref/.
[8] http://www.w3.org/TR/rdf-sparql-query/.
[9] This is currently not met by SPARQL, thus leading to a reduced expressivity in the target language.

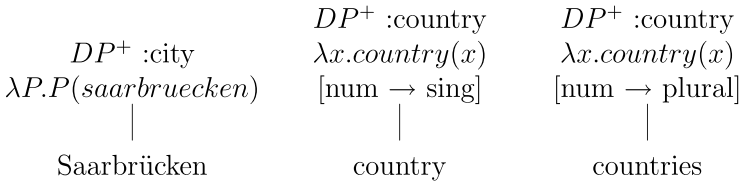


Fig. 3. Elementary trees automatically generated from the KB.

expressions to domain-specific predicates. It is important to emphasize that our natural language interface does not require any sort of pre-encoded grammar as input of the system. The grammar underlying the ORAKEL system consists exactly of the union of the trees in the domain-independent and the domain-specific lexica. Thus, the task of the lexicon engineer is to actually provide a domain-specific grammar to the system. As this is a difficult task – compare the discussion of syntactic variants in Section 3.1 – in our natural language interface we implement an approach in which the user simply instantiates subcategorization frames and maps these to domain-specific relations in the ontology. Actually, the linguistic subcategorization frames as well as the relation types are organized in a type hierarchy, such that only structures of compatible arity are mapped onto each other. As shown in Fig. 4, in our type system we distinguish between binary, ternary and quaternary subcategorization frames which can be mapped to binary, ternary and quaternary relations, respectively.[10]

Examples for binary subcategorization frames are transitive verbs, intransitive verbs with a prepositional complement, relational nouns with one prepositional complement as well as participles with prepositional complements:

- transitive: verb(subject,object), e.g. *border*
- intransitive + prepositional complement: verb(subject, prep:pobject), e.g. *flow through*
- noun + pp: noun(prep: pcomp), e.g. *capital of*
- participle+pp: participle(prep: pcomp), e.g. *located in*

For example, the lexicon engineer could create the mappings shown in Fig. 5 for a geography knowledge base. While some of these mappings may seem straightforward, they are indeed crucial for ORAKEL to generate a full domain-specific grammar mapping linguistic expressions to appropriate semantic representations. How should ORAKEL in fact know that the relation *border* is best expressed with a transitive verb with the same name? How should it know that the *capital* relation should best be expressed by the noun '*capital (of)?*' Though simple heuristics based on matches between relation names and verbs or nouns might be applied, they will in general not suffice to cover all the possible lexical variations one can use to ask for a specific relation. Actually, language is too variable to be captured by such straightforward heuristics. Further, it is crucial to determine the order in which the arguments of the relation map to arguments of the linguistic predicate, e.g. the verb or noun in question. Instead of building on an automatic, heuristic, and therefore error-prone process, in ORAKEL we build on a more controlled approach in which users can specify lexical variants (with some support though) as well as the correct order in which the arguments map onto each other. Examples of mappings which are not as straightforward are (3) and (7). The third mapping is interesting in that it provides a non-straightforward lexical variant for asking for the inhabitants of a city. The seventh mapping introduces a further non-obvious lexical variant to ask for the *flow_through* relation between rivers and cities. By this, we introduce a lexical ambiguity into the lexicon, as '*pass through*' can denote either the *flow_through* relation between rivers and cities as well as the *located_at_highway* relation between highways and cities.[11] Moreover, it is not always the case that the domain of a relation is mapped to the subject and the range to the object in the corresponding verb subcategorization frame. Such an example is provided by mapping (8) where the sub-

[10] Note that there is no principled limit to the arity of relation. However, according to our experience considering relations of up to four suffices to cover most examples in practice.

[11] Though we do not discuss this further in this article, it is important to emphasize that ORAKEL can recognize and handle such lexical ambiguities. The details are given in our technical report.

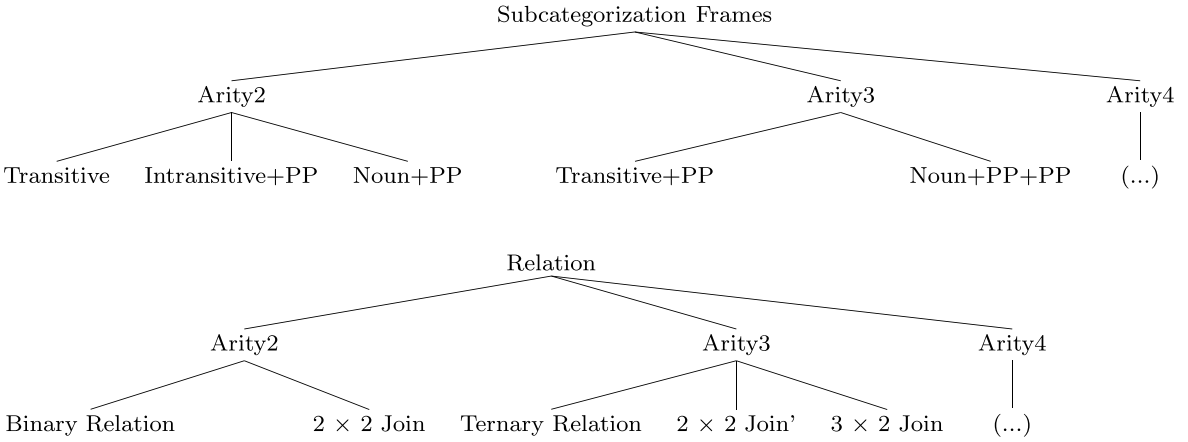


Fig. 4. Type hierarchies of linguistic templates and relations.

location(pcomp(of): x)	→	locatedIn(x: location, y: location) (1)
inhabitants(pcomp(of): x)	→	inhabitants(x: location, y: integer) (2)
live(subj: x,pobj(in): y)	→	inhabitants(y: state/city, x: integer) (3)
capital(pcomp(of): x)	→	capital(x: state, y: city) (4)
length(pcomp(of): x)	→	length(x: river, y: integer) (5)
flow(subj: x, pobj(through): y)	→	flow_through(x: river, y: city) (6)
pass(subj: x, pobj(through): y)	→	flow_through(x: river, y: city) (7)
pass(subj: x, pobj(through): y)	→	located_at_highway(y: city, x: highway) (8)
height(pcomp(of): x)	→	height(x: mountain, y: integer) (9)
border(subj: x, obj: y)	→	borders(x: location, y: location) (10)
located(pcomp(in): x)	→	locatedIn(y: location, x: location) (11)

Fig. 5. Some example mappings.

ject and object of '*pass through*' are mapped to the range and domain of *located_at_highway*, respectively. It is therefore necessary that the lexicon engineer also specifies the order in which the relation's arguments map to the ones of the subcategorization frame. For the noun subcategorization frames, the argument of the relation which has not been mapped to the *pcomp* position – the *y*-argument in the above examples – is stored externally to the actual frame as it will be normally expressed in form of a copula[12] construct such as "*What is the length of the Rhein?*". Note that this holds also for participles which are also typically used in copula constructs, e.g. "*Where is Karlsruhe located in?*".

Further, for nouns complemented by the preposition '*of*', the system also generates trees allowing to ask for the corresponding relation using the verb '*have*' (see the examples below). For methods such as *capital*, which do not have a datatype such as a string or an integer as range, and which have been mapped to a noun+pp, ORAKEL's grammar generation mechanism does not only generate relational noun phrases such that one can ask: "*What is the capital of Baden Württemberg?*" using a copula construct, but also a form in which

[12] A copula is an expression involving the verb '*be*' and linking the subject to some property or object.

the argument mapped to the *pcomp* position is existentially quantified over. This allows to ask a question like "*Which rivers flow through a capital?*" For verbs, it generates the active, passive and verb-last forms, but also relative clauses complementing a noun phrase. On the basis of the above example mappings, the system then generates elementary trees, such that it is able to interpret the following questions where the relevant mappings are indicated in brackets:

- What is the location of Stuttgart? (1)
- How many inhabitants does Baden Württemberg have? (2)
- How many people live in Karlsruhe? (3)
- What is the length of the Rhein? (5)
- What is the capital of Baden Württemberg? (4)
- Which river flows through the capital of Baden Württemberg? (4,6)
- Which rivers flow through a capital? (4,6)
- What is the length of the Rhein? (5)
- Which river flows through the most cities? (6)
- Which river flows through a state which borders Baden Württemberg? (6,10)
- Which river passes through München? (7)
- Which highways pass through Berlin? (8)
- What is the height of the Zugspitze? (9)
- Which countries does Baden Württemberg border? (10)
- Which countries are bordered by Baden Württemberg? (10)
- Which countries border Baden Württemberg? (10)
- Which state borders the most countries? (10)
- Where is Karlsruhe located in? (11)

Binary relations with an integer as range are special types of relations which can also be mapped to adjectives by specifying (i) the base, (ii) the comparative, and (iii) the superlative form of the adjective, additionally indicating whether it denotes a positive or negative scale (this is similar to the approach in TEAM [29]). For example, the adjectives '*big*', '*long*' and '*high*' are mapped to the relations *inhabitants*, *length* and *height*, respectively:

- adj(big,bigger,biggest,positive) → inhabitants(city,integer) (Adj1)
- adj(long, longer, longest,positive) → length(river,integer) (Adj2)
- adj(high,higher,highest,positive) → height(mountain,integer) (Adj3)

This then allows to ask the following questions:

- How long is the Rhein? (Adj2)
- How high is the Zugsitze? (Adj3)
- How big is Karlsruhe? (Adj1)
- Which is the longest river? (Adj2)
- Which river is longer than the Rhein? (Adj2)
- Which is the highest mountain? (Adj3)
- Which cities are bigger than Karlsruhe? (Adj1)

The positive/negative distinction is necessary to generate the correct semantics for comparative and superlative adjectives. In fact, '*big*', '*long*' and '*high*' are positive adjectives in our sense, while '*small*' is an example of a negative adjective. In general, specifying the semantics of adjectives in base form is a quite delicate issue as an adjective such as '*big*' actually denotes a fuzzy set in the sense of Zadeh [54]. However, we need to specify the semantics of adjectives in order to answer queries such as "*Which rivers flow through big cities?*". The solution adopted in ORAKEL is to expect a definition of the semantics of an adjective in terms of a rule, e.g.

$$\forall x\ big(x) \leftarrow city(x) \wedge inhabitants(x, y) \wedge y > 500.000$$

It is important to emphasize that currently ORAKEL can only handle scalar adjectives such as '*big*', '*high*', '*long*', etc. In particular, it can not deal with non-scalar adjectives such as '*German*', which would need to be translated into a corresponding relation in which a specific value is inserted. The adjective '*German*', for example, could be translated into the expression $\lambda x locatedIn(x, Germany)$.

In order to allow a lexicon engineer for specifying the above described mappings, we have created a tool called FrameMapper which supports the lexicon engineer via a graphical user interface in performing the mappings. Besides allowing the lexicon engineer to create verb and noun subcategorization frames and graphically map their arguments to the domain and range of a certain relation, FrameMapper also features an adjective view which supports the specification of the semantics of scalar adjectives. In addition to entering the base, comparative and superlative forms of the adjective, the lexicon engineer is also expected to specify the relation in the knowledge base which the adjective refers to, whether the adjective denotes a positive or negative scale as well as to indicate a threshold value. Thus, indirectly, the lexicon engineer is specifying the semantics of adjectives in a way similar to the above stated rule.

As shown in the type hierarchy depicted in Fig. 4, the mapping model is not restricted only to binary relations. Subcategorization frames can also be mapped to joins of several relations, e.g. a subcategorization frame of arity 2 can also be mapped to two binary relations joined at a given position (2×2-Join in Fig. 4), a subcategorization frame of arity 3 can be mapped either to a simple ternary relation, a join of two binary relations in which the joined position is also mapped to an argument in the frame (2×2-Join' in the figure) or to a join of three binary methods (3×2-Join in the figure), etc. Hereby *Join*' denotes a join in which the joined position has also been mapped to an argument in the subcategorization frame while for *Join* this is not the case. This explains why $n \times 2$ *Join*' joins have an arity of $n + 1$ while $n \times 2$ *Join* joins have an arity of n.

The reason for introducing such an elaborated type system is the fact that linguistic expressions in many cases do not correspond directly to one relation in the knowledge base, but express a combination of different relations in the knowledge base which can be expressed through joins.

As a more complex example, assume the following relations are given in the knowledge base: *author*(*article*,*author*), *title*(*article*,*title*), *year*(*article*,*string*). If we create a 3×2 Join by joining the article position of the three relations, we can map this ternary relation to a transitive verb '*publish*' with a prepositional complement introduced by the preposition '*in*' such that we can ask a question like "*Who published which article in 2002?*" (see also the discussion of this join in Section 5).

Summarizing, the crucial aspect here is that the domain-specific grammar necessary for understanding domain-specific expressions is generated in the background as a byproduct of a lexicon engineer interacting with the system and mapping subcategorization frames onto appropriate relations in the knowledge base. Thus, no pre-encoded grammar is actually needed in the system. In order to map relations defined in the ontology to appropriate subcategorization frames, lexicon engineers are supposed to use the *FrameMapper* lexicon creation frontend, which allows to select a relation and to create corresponding subcategorization frames. The ontological restrictions on the concepts which can be used at the different argument positions of the relation will then be used as selectional restrictions in the subcategorization frames and exploited for disambiguation. After the lexicon engineer has assigned all the relations to corresponding subcategorization frames or adjectives, she/he can export the lexicon, which can then be used by the natural language interface to answer users' questions with respect to the knowledge base. In our model, we do not expect a lexicon engineer to model the lexicon in one turn from scratch, but assume that the lexicon is created in several iterations. After the lexicon engineer has created a first version of the lexicon, the system is deployed. The lexicon engineer gets presented the questions which the system failed to answer and the process is iterated. Our hypothesis is in fact that with such an iterative method, the quality of the lexicon can be constantly improved. We will present experimental evidence for this hypothesis in Section 6. Before presenting the results of our experiments in Section 6, in the following section we describe FrameMapper's graphical user interface.

5. Graphical user interface

Fig. 6 shows a screenshot of FrameMapper's graphical user interface. It shows how a lexicon engineer is mapping the *flow_through* relation to the intransitive verb '*flow*' featuring a prepositional complement

Fig. 6. GUI of FrameMapper showing a simple mapping for the geographical domain.

introduced by the preposition '*through*'. The figure shows the three main panes of *FrameMapper*. In the top pane, the lexicon engineer sees the relations specified in the ontology. In the second pane, the lexicon engineer can see the different subcategorization frames assigned to the active relation. In the third pane, she/he sees a graph visualization of the current subcategorization frame and of the selected relations. She/he can then graphically map the arguments of the frame to the ones of the selected relation(s). In the GUI screenshot in Fig. 6, the lexicon engineer has already mapped the intransitive verb '*pass*' with a prepositional complement introduced by '*through*' to the *flow_through* relation (this can be seen in the middle pane). Currently, the lexicon engineer is also mapping the intransitive verb '*flow*' with a prepositional complement introduced by the preposition '*through*' to the same relation. In particular, the lexicon engineer has already mapped the subject position of the verb '*flow*' to the domain of the *flow_through* relation and the prepositional complement to the range position of the same relation. Further, in the screenshot she/he has already entered the appropriate preposition '*through*' in the graph representing the subcategorization frame and is currently editing the verb, specifying that its base form is actually '*flow*'. With this information, the system can in the background generate all the grammatical variations of the intransitive verb '*flow*', thus allowing to ask for the *flow_though* relation in a variety of ways. In order to add a further verb, the user simply has to instantiate a new subcat-

egorization frame and perform the mapping again. The newly created subcategorization frame would then be added to the list of those subcategorization frames already created for the active relation(s) in the middle pane. In order to ease the process of adding lexical variants, we have also integrated the WordNet lexical database [26] with the purpose of automatically suggesting synonyms for the verb or noun currently edited. For this purpose, we only consider the first sense of each word, suggesting each of the words contained in the corresponding synset to the user as lexical variants on demand in the form of a check-box. Each selected synonym is then used to generate subcategorization frames only differing in the lexical element. However, this functionality was added recently and not used in the experiments described in Section 6.

It is important to mention that the type hierarchy described in the previous section is used to constrain the subcategorization frames offered to the user. For example, if the lexicon engineer selects a binary relation, she/he will only be able to instantiate a transitive, intransitive+PP or noun+PP subcategorization frames. In the adjective view, only relations with an integer as range are visualized such that (scalar) adjectives can me mapped to them.

Note that the user can also select various relations and carry out joins between them to specify more complex mappings involving more than one relation. Fig. 7 shows a screenshot of the GUI in which the user has chosen the three relations *author*(*publication*,*person*), *title*(*publication*,*string*) and *year*(*publication*,*string*), all joined through their domains, i.e. through the publication. The user has further instantiated a subcategorization frame for the transitive verb '*publish*' featuring a direct object as well as a prepositional complement introduced by the preposition '*in*'. Further, she/he has mapped the range of the author (*publication*,*person*) relation to the subject position, the range of the *title*(*publication*,*string*) relation to the object position as well as the

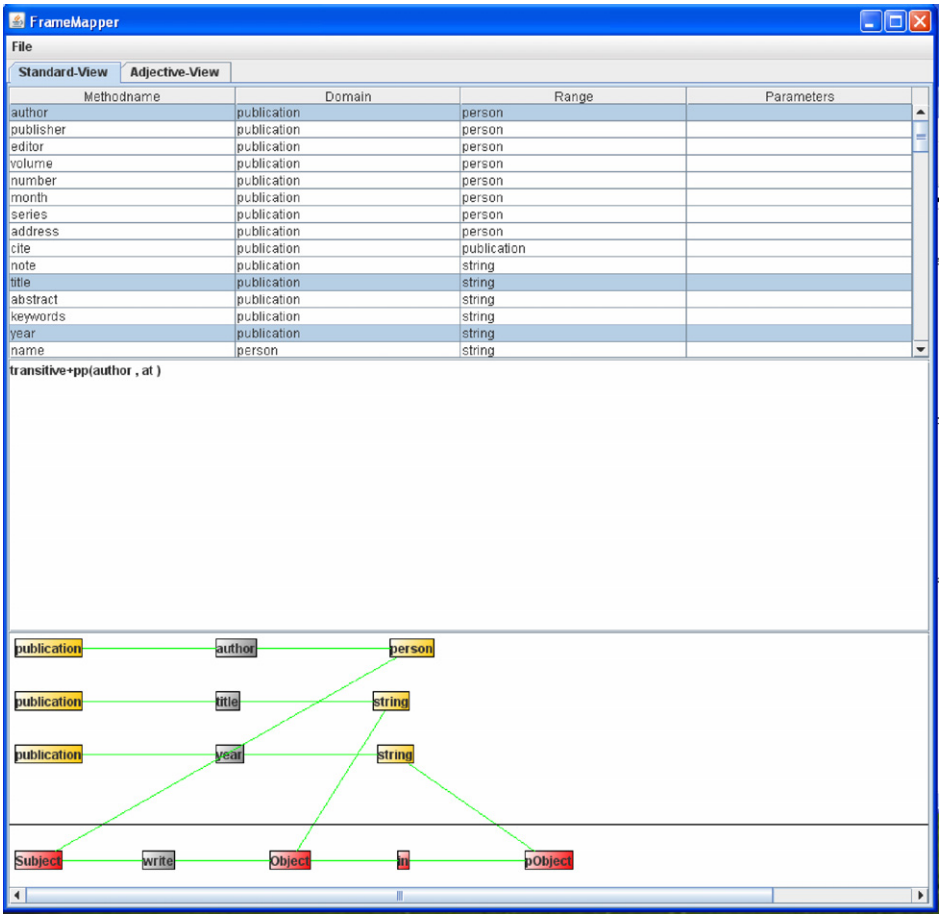


Fig. 7. GUI of FrameMapper showing a more complex mapping involving joins for the academic domain.

range of the *year*(*publication*,*string*) to the prepositional complement. This mapping would then allow to ask a question like "*Who published which article in 2002?*".

Finally, the lexicon engineer can export the lexicon, which can then be loaded into the ORAKEL natural language interface, but she/he can also import an already created mapping lexicon to add more subcategorization frames, thus supporting our iterative lexicon generation model.

6. Experiments

In this section, we first present the settings and results of our experiments, which have been carried out on two different domains showing that the ORAKEL system can be adapted to different domains without major efforts. First, we present a user study carried out with a knowledge base and corresponding ontology containing facts about German geography. The aim of this study was to demonstrate that computer scientists without any NLP expertise can indeed generate domain-specific lexica for the ORAKEL system without major difficulties. Second, we provide some statistics demonstrating that the system has potentially a reasonable linguistic coverage. The results of the first study have been partially presented also in [17] but are described here in more detail. In this article, we additionally discuss a case study carried out at British Telecom in which the ORAKEL natural language interface was successfully applied to offer enhanced search over a digital library. The application of ORAKEL as well as other tools to the BT use case has been previously described in [9].

6.1. User study

The aim of the user study was to show that computer scientists without any NLP expertise can indeed generate reasonable domain-specific lexicons for the ORAKEL natural language interface. The study also provides first evidence that our iterative approach is indeed feasible.

The knowledge base used for the experiments contains geographical facts about Germany. In particular, it contains states, cities, rivers and highways in Germany, as well as the name of the neighboring countries. It is a small knowledge base handcrafted by students at our department independently of the experiments described here. The knowledge base contains the number of inhabitants of each state and city as well as the capital of each state. For rivers and highways, it contains information about the cities they pass. For rivers, it additionally contains their origin as well as length. It also contains mountains and their heights. Overall, the knowledge base comprises 260 entities: 108 highways, 106 cities, 18 rivers, 16 states, 9 (bordering) countries and 2 (bordering) seas as well as one mountain peak, i.e. the *Zugspitze*. The relations defined in the ontology are the following ones (given in F-Logic style notation):

- city[locatedIn => location].
- city[inhabitants => integer].
- state[inhabitants => integer].
- state[borders =>> location].
- city[located_at_highway =>> highway].
- river[length => integer].
- river[origin => location].
- river[flows_through =>> city].
- mountain[height => integer].
- city[capital_of => state].

Here, => denotes that the relation is functional, i.e. it can have at most one instance as range, and =>> denotes that there can be more than one instance as range of the relation.

The user study involved one of the authors of this article, as well as 26 additional test persons from four different institutions, both academic and industrial. Of these 26 test persons, 25 were computer scientists and 1 a graphic designer, most of them without any background in computational linguistics. The role of the author as well as two of the other participants was to construct a lexicon each (thus playing the role of lexicon engineers), while the rest played the role of end users of the system. We will refer to the author as *A* and the other

Table 1
Results for the different lexica

Lexicon	Users	Rec. (avg.) (%)	Prec. (avg.) (%)
A	8	53.67	84.23
B (1st lexicon)	4	44.39	74.53
B (2nd lexicon)	4	45.15	80.95
C (1st lexicon)	4	35.41	82.25
C (2nd lexicon)	4	47.66	80.60

two participants constructing a lexicon as *B* and *C*. While *A* was very familiar with the lexicon acquisition tool, *B* and *C* were not and received 10 minutes of training on the tool as well as 10 min explanation about the different subcategorization types, illustrated with general examples. Whereas *A* constructed a lexicon in one turn, *B* and *C* constructed their lexicon in two rounds of each 30 min. In the first round, they were asked to model their lexicon from scratch, while in the second round they were presented those questions which the system had failed to answer after the first round consisting of four sessions with different users. They were asked to complete the lexicon on the basis of the failed questions. Overall, they thus had 1 hour to construct the lexica. The 24 persons playing the role of the end users also received instructions for the experiment. They received a document describing the experiment, requiring them to ask at least 10 questions to the system. Further, the scope of the knowledge base was explained to them. They were explicitly told that they could ask any question, also involving negation and quantification, with the only restriction that it should begin with a *wh*-pronoun such as *which*, *what*, *who*, *where* as well as *how many* or *how* + *adjective*. For each answer of the system, they were asked to specify if the answer was correct or not. The results are thus reported in the following as *recall*, i.e. the number of questions answered correctly by the system divided by the total number of questions asked to the system. Excluded from this were only questions with spelling errors or which were obviously ungrammatical, as well as questions which were clearly out of the scope of the knowledge base.[13] We also give the *precision* of our system as the number of questions for which the system returned a correct answer divided by the number of questions for which it returned an answer at all. Note that precision and recall are defined here in line with [44] and not in the standard information retrieval sense (cf. [3]). Table 1 shows these results for each of the lexicon constructors and the two iterations. The first interesting conclusion is that, for both *B* and *C*, there is an increase in recall after the first round. Thus, the results show that our iterative methodology to lexicon customization is indeed promising. The involved users also confirmed that it was easier to extend the lexicon given the failed questions than creating it from scratch. The second interesting result is that the lexicons created by *B* and *C* show a comparable recall to the lexicon developed by *A*. In fact, we found no significant difference (according to a Student's *t*-test at an α-level of 0.05) between the results of *B*'s lexicon ($p = 0.32$) and *C*'s lexicon ($p = 0.15$) compared to *A*'s lexicon. This shows that our lexicon acquisition model is in fact successful. In general, the results have increased after the second iteration, with the exception of a slight drop in precision for user *C* at the second round. We expect that further iterations will continuously improve the lexica. This is, however, subject to further analysis in future work.

6.2. Question analysis

Having shown that domain experts are able to map relations in a knowledge base to subcategorization frames used to express them, an important question is to determine how big the coverage of the different subcategorization frames is with respect to the questions asked by the end users. Overall, the end users asked 454 questions in our experiments (actually much more than the number of questions requested). Table 2 summarizes the constructions used together with illustrating examples, giving their percentage with respect to the

[13] As already mentioned before, our aim here was not to develop a system robust enough to cope with misspellings, ungrammatical input or sentences for which no parse can be found. In order to verify the appropriateness of our adaptation model we have thus assumed ideal settings, i.e. that the sentence is grammatical, has no misspellings and is in the conceptual scope of the knowledge base. A real system which is deployed should certainly address these issues in order to achieve a robust behavior. We think that robustness can be achieved by implementing certain heuristics on top of our current system. However, we leave these issues for future work.

Table 2
Usage of constructions in main clause (in percent with respect to the total number of questions)

Construction	#	%	Example
Intransitive+pp (∗)	169	37.22	How many cities *does* the A1 *pass through* ?
Transitive (∗)	56	12.33	How many states *does* Baden Württemberg *border* ?
be+np (∗)	102	22.47	What *is the capital of* Bayern ?
be+adj (∗)	22	4.85	How *long is* the Donau ?
be+pp (+)	22	4.65	Which cities *are in* Bayern ?
be+dp (∗)	18	3.96	Where *is* Düsseldorf ?
be+np (poss) (−)	1	0.22	What *is* Bayern*'s capital* ?
be+participle (−)	12	2.64	Which cities *are located in* Bayern ?
be+np (inv) (−)	4	0.88	Which state *is* München *capital of* ?
be+np (superlative) (∗)	2	0.44	What *is the capital of* Bayern ?
be+comp (∗)	7	1.54	Which cities *are bigger than* Frankfurt ?
Passive (∗)	1	0.22	Which states *are bordered by* at least 2 countries ?
have (∗)	32	7.04	Which states *have* more inhabitants than Hessen ?
Number of queries:	454	100	

total number of questions. The results show that in principle, assuming that the lexicon is complete, with our basic subcategorization frames *transitive*, *intransitive+pp*, *np* and *adj* as well the constructions automatically generated from these (marked with a '∗' in the table), we get a linguistic coverage of over 93%.[14] This 93% of linguistic coverage essentially measures the ideal system coverage in terms of syntactic constructions given that the lexicon is complete. Of course, due to the fact that the lexica considered in our experiments are from complete, the system's actual coverage is far from 93%. Nevertheless, it is important to know that our system has in principle a reasonable linguistic coverage given that the lexicon captures a sufficient number of lexical and syntactic variants for expressing a certain relation. In general, the ideal linguistic coverage shows that it is indeed feasible to focus on a few subcategorization types. The intelligence lies anyway in the generation of the corresponding elementary trees from the subcategorization frames. The generation, however, remains totally transparent to the user. For the constructs marked with '+', we additionally need to specify the semantics of prepositions with respect to domain-independent relations. For example, '*in*' maps to the relation *locatedIn* in our system. This is assumed to be a relation which is available in any domain. For other domains however, '*in*' might also have a temporal interpretation which can be formalized with respect to the DOLCE foundational ontology [16]. Constructions which were added after the experimental evaluation are the ones marked with a '−', i.e. the *be+np (possessive)*, *be+np (inverse)* and *be+participle* constructions, which account for around 3.75% of the cases. Considering the first two is merely a question of generation of the appropriate elementary trees and was accomplished in a straightforward way. For the third construct, the subcategorization types had to be extended to participles subcategorizing a prepositional phrase, e.g. such as *located in*.

6.3. Real-world application

As a further experiment, our approach has been applied within the British Telecom (BT) case study in the context of the SEKT project.[15] In this case study the aim was to enhance the access to BT's digital library by a natural language interface. BT's digital library contains metadata and fulltext documents of scientific publications. The knowledge base considered within this case study is several orders of magnitude larger than the one considered in the context of the experiments carried out on the geographical domain. The ontology used to describe the metadata is the PROTON ontology,[16] which consists of 252 classes and 54 relations. While the PROTON ontology (the schema of the data) is stored in an OWL ontology in this scenario, all the publication metadata are stored in a database. The schema of the database, however, has been mapped to the PROTON ontology, such that queries to the ontology are evaluated by taking into account the meta-

[14] The constructions marked with a '−' were added after the user study described before.
[15] http://www.sekt-project.com/.
[16] http://proton.semanticweb.org/.

data in the database. The knowledge base contains metadata about 67.015 authors, 17.174 topics and 33.501 documents (journal articles, conference articles, conference proceedings, periodicals and books). Further, there are 66.870 instances of the *AuthorOf* relation and 165.089 instances of the *isAboutTopic* relation. As the data size is indeed several orders of magnitude larger compared to our geography domain, in the context of this use case it was not feasible to generate a grammar entry for each value in the database. Therefore, we performed a slight change to the ORAKEL system to allow for the dynamic creation of elementary trees at query time for names of instances. This was achieved by considering every sequence of upper-case words as a potential candidate for an instance, generating appropriate elementary trees at runtime. The domain-specific lexicon is thus generated only for concepts in the ontology in this scenario, while the part of the lexicon containing the instances is generated dynamically. This move was important to ensure efficiency in this setting.

A graduate student and a Ph.D. student spent overall approximately six hours creating a lexicon for a subset of PROTON relevant for the digital library using FrameMapper with the result that queries about authors, topics etc. about documents could be answered successfully against the BT ontology and database. Examples of such questions are:

- What conference articles did John Davies write?
- Which conference articles do you know?
- Who wrote which document?
- Who wrote "The future of web services"?
- Who wrote about "Knowledge Management"?
- What article deals with Photography?
- Which journal articles were written by whom?
- What are the topics of "The future of web services"?
- Which conference articles were classified as religion?
- Which articles are about "Intellectual Capital"?
- What articles were written by Lent and Swami?
- Who is the author of a document that talks about which concept?
- Who wrote which articles about what?
- Which documents are about F-Logic and Insurance?

As in our case study with the small geographical knowledge base described above, we also carried out an evaluation of the ORAKEL system on the BT digital library. As in the experiments described above, the end users received written instructions describing the conceptual range of the knowledge base, requiring them to ask at least 10 questions to the system. In each of three iterations, four end users asked questions to the ORAKEL system and the graduate student updated the lexicon on the basis of the failed questions after the first and second round for about 30 min., respectively. The end users were also asked to indicate whether the answer was correct or not, which allows for the evaluation of the system's performance in terms of precision and recall. The results of the evaluation are presented in Table 3, which clearly shows that the second iteration performed much better than the first one, both in terms of precision and recall. In the third iteration, there was a further gain in recall with respect to the second iteration. Overall, this indeed shows that our iterative lexicon model is reasonable and in fact leads to incremental improvement of the system's lexicon. Fig. 8 shows the user web browser interface deployed at BT. In essence, it consists of a plain text field into which the user enters his/her query and a result view showing the results of the evaluation of the query.

Overall, the application of ORAKEL to enhance the access to BT's digital library showed on the one hand that, given certain straightforward modifications, our approach can actually scale to much larger knowledge

Table 3
Results for the different iterations

Iteration	Rec. (avg.) (%)	Prec. (avg.) (%)
1	42	52
2	49	71
3	61	73

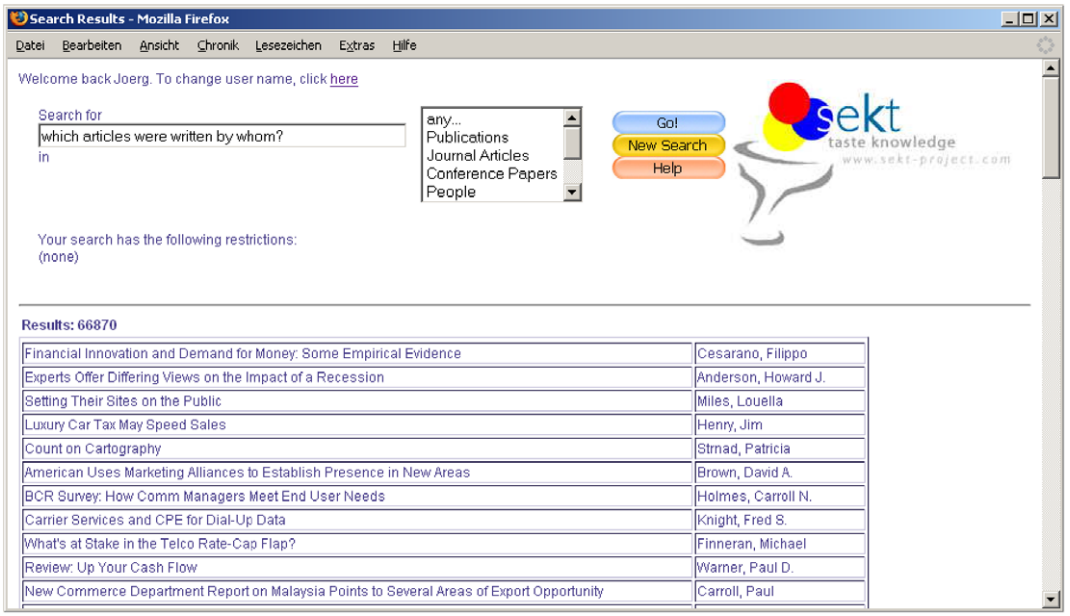


Fig. 8. User interface.

and data bases. Further, the additional use case confirms that the system can indeed be ported between domains in a more or less straightforward way. We refer the interested reader to [15,9,53] for further details about the case study at British Telecom.

7. Related work

Discussing all the different NLI approaches presented in the literature is certainly out of the scope of this article. For a detailed review of natural language interfaces, the interested reader is referred to the overviews in [21,1], as well as to the evaluation survey in [42]. The latter shows that there have been already many laboratory and field studies of natural language interfaces. However, most of the results reported are neither conclusive nor address the issue of how easy it is for non-NLP experts to adapt the systems to a given domain.

In this discussion we focus on the different methods which have been proposed in the literature to adapt a natural language interface to a certain domain. On the one hand, we briefly discuss *data-driven* approaches, which learn the syntax-semantics interface in a supervised fashion on the basis of training examples provided to the system. In the case of a natural language interface, training examples consist of pairs of natural language questions and corresponding knowledge or database queries. Second, we discuss approaches based on lexical matching. Approaches which fall under this category are PRECISE [44] and AquaLog [35]. These approaches rely on heuristics for matching the words in the input to elements in the ontology (in the case of AquaLog) or to the database schema (in the case of PRECISE). For this purpose, these systems rely on lexical knowledge as contained in lexical resources such as WordNet. Finally, we also discuss systems which rely on deep linguistic processing and require a manually engineered explicit mapping between more complex linguistic and ontological structures beyond mere lexical correspondences. This is for example the case of the TEAM system [29], the Quetal [27] system, the system by Bernstein et al. based on ACE [7] as well as ORAKEL.

7.1. Data-driven approaches

Customization to a domain in the system of Thompson et al. [52] is achieved by training a parser using Inductive Logic Programming (ILP) techniques on the domain in question. In particular, a standard

shift-reduce parser is assumed and ILP is used to learn parsing control strategies. Such an approach obviously needs training data and Thompson et al. do not discuss if such an approach relying on training data is actually feasible from a user's point of view. The system has been evaluated on two domains (jobs and geography), and achieves very decent accuracy levels between 25% and 70% for the geography domain and between 80% and 90% accuracy for the job domain, depending on the amount of training data used. In fact, while such an approach in which the syntax-semantics interface can be learned is certainly appealing, it is unclear in how far it is feasible to require users to provide training examples for the system. In particular, it would be interesting to compare the efforts involved in providing a sufficient number of training examples and engineering a lexicon for the ORAKEL system. This is certainly an interesting issue for future work.

7.2. Approaches based on lexical matches

In this section we discuss two systems, PRECISE and AquaLog, which essentially rely on simple lexical matches between words in the input question and elements in the database or ontology schema. We compare these systems quantitatively and qualitatively to our own system.

The PRECISE system [44] focuses on the reliability of NLIs and is formally proved to be 100% precise, given an appropriate domain-specific lexicon. PRECISE implements an approach based on graph matching in which, essentially, the words in the input question are mapped to database relations, columns and values. For this purpose, so-called syntactic markers are removed from the input sentence with the result that only content words remain, which are mapped to tokens representing database relations, columns or values. Hereby a token is a set of words matching a certain database element. PRECISE introduces the notion of *semantically tractable questions*, i.e. questions which have a complete tokenization such that each word is mapped to a distinct token corresponding exactly to one database element and such that every value has been mapped to one attribute and at least one of the value tokens matches a "wh-value". The problem is thus reduced to finding a mapping between words and tokens such that attributes are connected to their values. Popescu et al. formulate this task in terms of determining the maximal flow through a graph and apply the max-flow algorithm to compute a maximal flow which satisfies the above constraints. It is interesting to emphasize that PRECISE also allows ellipsis, that means, attributes to be left out, as in "*Which Chinese restaurants are downtown?*", where only the relation *restaurants* and two values – '*Chinese*' and '*downtown*' – are specified. The precision of PRECISE of almost 100% on real data is certainly impressive, but in contrast to our approach it only focuses on conjunctive SQL queries. In fact, the questions that it can handle are only a subset of the questions that ORAKEL can handle, which supports arbitrary quantification, Further, our system has also been demonstrated to be very reliable as the precision ranges between 74% and 85%. Strictly speaking, PRECISE does not need any customization, but on the contrary it is not able to handle all the range of questions that ORAKEL can handle. Though PRECISE is claimed to handle also quantification and counting, it is not clear in how far questions like "*Which river flows through every country?*", "*What is the largest city in the state with the smallest population?*" and "*What river does not traverse the state with the smallest population?*" can be handled. In fact, such questions are strictly speaking not semantically tractable as '*every*' and '*not*' would either be treated as syntactic markers and therefore removed or would not map to any values, columns or relations. For the superlatives '*smallest*' and '*largest*', similar remarks apply. The aims of PRECISE and our system are thus complementary, as we have focused on developing an approach by which domain experts can quickly create an appropriate domain-specific lexicon such as needed by any NLI.

The recently presented AquaLog system [35] essentially transforms the natural language question into a triple-representation and then relies on similarity measures to map the triples to appropriate relations defined in the ontology. AquaLog relies on GATE [22] to transform natural language questions into linguistic triples which need then to be mapped to the ontology. For this purpose, AquaLog relies on a so-called Relation Similarity Service (RSS) to calculate similarities between the word in the question and relations and concepts in the ontology. Some basic support for lexical and structural disambiguation is provided. Further, AquaLog also features a learning mechanism which stores unknown words together with a representation of the context and the decision taken by the user in order to apply this knowledge for future disambiguation of similar examples (see [36]).

PRECISE and AquaLog are examples of systems relying on lexical matches to map a natural language expression to appropriate knowledge base or database relations. However, such approaches face principled limits. For example, given that the relation between an author and his publications is termed *authorOf*, a question like "*Who wrote which publication?*" could not be analyzed correctly unless we have other knowledge available linking *writing* to an *author*. While AquaLog is for example able to account for such examples by exploiting lexical knowledge from WordNet, it is clear that such approaches face principled limits in the sense that they strongly depend on the availability of the appropriate lexical relations in the resource used. An approach similar to ours, where a lexicon engineer can directly provide mappings between linguistic and ontological structures, does not face these principled limits and is directly controllable. The drawback is certainly that a lexicon engineer needs to make the effort of creating a lexicon. However, our experiments have shown that the effort can be significantly reduced by an intuitive user interface and an appropriate adaptation model. A further difference between ORAKEL on the one hand as well as PRECISE and AquaLog on the other is that our system produces full fledged logical queries, while PRECISE only generates SQL-like queries and AquaLog produces triples. The expressivity of such approaches is thus restricted in comparison to ORAKEL. In particular, ORAKEL can in principle deal with arbitrary quantification by translating questions into appropriate queries. It is not clear in how far PRECISE or AquaLog could be extended to handle quantification in questions without the need for adhoc procedural extensions. Further, ORAKEL supports joins between different relations as a result of the process of interpreting the question compositionally, while AquaLog and PRECISE perform joins as a post-processing step after interpretation when a SQL query is constructed (in the case of PRECISE) or certain elements in the triples are unified (in the case of AquaLog).

7.3. Approaches relying on engineered lexica

TEAM's [29] approach to customization consists in asking questions to a user to acquire linguistic knowledge about certain words, i.e. verbs, nouns, adjectives, etc. as well as their relation to database fields. For a verb, TEAM would for example ask a user for its arguments and whether they are optional or mandatory, for prepositional phrases, for particles and whether they are separable from the verb as well as for different possible types of realization, e.g. passive, unaccusative or dative constructions, etc. TEAM also acquires information about adjectives in a similar way as ORAKEL by asking for the predicate in the knowledge base expressing the corresponding attribute as well as for the direction of the scale measured (e.g. positive for '*big*', negative for '*small*'). Further, TEAM also handles closed-class words by assigning them a domain-independent meaning. In general, TEAM and ORAKEL share three very important aspects:

- the assumption that the knowledge base is independent of the system and should not be changed for the purposes of the natural language interface,
- the requirement that users with knowledge about the domain or underlying database but without any knowledge about formal linguistics or natural language processing should be able to adapt of the system, as well as
- a system design which cleanly separates domain-independent from domain-specific components thus supporting the adaptation in a principled way.

Concerning the last point, in TEAM questions are parsed and translated into a general logical form including quantifiers (but also intensional operators and high-order operators, query operators, etc.), which is then translated in a second step to the structure of the underlying database. Further commonalities are the fact that both systems support joins of relations in the database or knowledge base and distinguish between *object* and *datatype* properties, which are called *symbolic* and *arithmetic* fields in TEAM, respectively. Further, TEAM offers some support for type coercion and disambiguating quantifier scope. Concerning the interpretation of light verbs such as '*have*' or nominal compounds, TEAM applies a different strategy than in our approach. While we rely on a generation approach in which '*have*' has as many elementary trees as possible domain-specific interpretations, TEAM tries to map '*have*' to an appropriate database relation while analyzing the sentence. In our approach, the possible meanings of '*have*' are actually determined by the noun+of structures

instantiated by the lexicon engineer, for example *capital (of)*, *inhabitants (of)*, etc. This allows then to correctly interpret questions such as "*Which capital does Baden Württemberg have?*" or "*How many inhabitants does Karlsruhe have?*". Analogous is the treatment of the vague preposition '*with*' in our approach, allowing to ask questions like "*Which is the city with the most inhabitants?*". Overall, it has never been shown that a user-centered customization approach such as used by TEAM is indeed feasible. Our work has aimed at addressing exactly this gap.

The approach in the Quetal system [27] implements the mapping from a question to a query via three intermediary stages: (i) construction of Robust Minimal Recursion Semantics (RMRS) representation, (ii) mapping to domain-independent frame-like structures relying on SUMO [43], FrameNet [4] and WordNet [26], as well as (iii) construction of so-called *proto-queries*. The use of RMRS[17] (Robust Minimal Recursion Semantics) supports underspecification of scope ambiguities. Further, the approach implements a hybrid technique to interleave shallow and deep processing for the purpose of robustness and to yield a rich semantic analysis of the questions to which several components can contribute, e.g. a HPSG parser, a shallow finite-state parser, a named entity recognizer etc. The proto-queries are abstract database-like queries comparable to our logical forms, but lacking any sort of quantification, negation or counting operators. Counting is for example performed after the answers have been returned by the inference engine. As in our approach, these proto-queries are translated to different query languages, i.e. SQL or SeRQL, the query language implemented in Sesame [11]. Domain independence is achieved by mapping RMRS structures to domain-independent resources. While in our approach certain closed-class words are mapped to DOLCE, in the Quetal system, RMRSs are mapped to conceptual frames from FrameNet and domain-independent concepts from SUMO. The mapping to the specific knowledge base is achieved by special rules which need to be written for every different knowledge base. These transformation rules thus constitute the adaptation mechanism behind the Quetal system. These rules are defined on SUMO and FrameNet structures and thus map domain-independent structures to domain-specific ones. Thus, the domain of these rules is defined in a principled way and remains constant across domains. However, also in the case of the Quetal system, the question needs to be raised if such transformation rules can be created by an average user. At least, the specification of these rules requires familiarity with SUMO, FrameNet and WordNet. Such a requirement might impose too large of a burden on a naive user and/or domain expert. The system has been successfully implemented on the one hand for a database containing information about Nobel Prize winners as well as an RDF knowledge base containing information about language technologies, patents, researchers, etc. The system has been evaluated on 100 questions from the Nobel Prize domain. Given an appropriate filtering and voting mechanism of the three best parse trees, a correct proto query is generated in 58% of the cases. Of the cases for which a correct proto query is generated, 74.1% yield a correct answer. The performance of the Quetal system is thus comparable to the one of our ORAKEL system.

The system presented by Bernstein et al. [7] builds on a controlled language approach as implemented by the ACE (Attempto Controlled English) framework (see [28]). It requires a transformation from the parser output structures – DRSs[18] (Discourse Representation Structures) – as produced by the Attempto Parsing Engine (APE) to PQL queries formulated with respect to the relations and concepts of an underlying ontology. This transformation needs to be specified by hand by a system engineer. It is thus not clear if the system can indeed be adapted by end users.

In a strict sense, every natural language interface only supports a restricted language which is determined by the underlying grammar as well as the other capabilities of the system. Approaches based on controlled language, however, besides supporting only a restricted language, also give guidelines about how to (syntactically) express meaning in order to avoid ambiguities. This is the approach followed by ACE (Attempto Controlled English). The English sentence "*Every airline owns an aircraft.*" is for example semantically ambiguous between two readings in which either the universal quantifier outscopes the existential quantifier or the other way round. In ACE, a user would be requested to write "*Every airline owns an aircraft.*" in the first case, while writing "*There is an aircraft that every airline owns.*" in the second.

[17] See [20].
[18] See [32].

7.4. Summary and discussion

We have seen that there are at least three different main paradigms how customization of NLIs can be approached. Certainly, as for many other challenging problems, there is no free lunch here! While systems exploiting lexical resources in the background and relying on lexical matches seem not to require customization at first sight, they are inherently dependent on the coverage of a certain lexical resource. The more technical the target domain gets, the less we can expect general lexical resources to provide the necessary lexical variations. Thus, customization will never come for free in principle and extending and refining lexica for the application in question will ultimately be necessary. In this line, intuitive models for lexical acquisition as presented in this paper will be definitely necessary under the assumption that systems will not be completely used off-the-shelf. The three paradigms: data-driven learning, lexical matching and manually engineered lexica should thus complement rather than compete with each other. Thus, the strengths, limits, effort required as well as assumptions of different approaches need to be investigated more closely in the future. For sure, we should give up the dream of a free lunch and not expect systems which can simply be taken off-the-shelf and applied without any sort of customization and tuning.

8. Conclusion and future work

We have presented the design choices, architecture, implementation and evaluation of our adaptable natural language interface ORAKEL. ORAKEL is based on the explicit representation of a lexicon which maps natural language constructions to ontological structures. The fact that the lexicon is explicitly represented allows to directly control the lexicon, a clear advantage with respect to approaches which essentially exploit lexical knowledge to establish matches between words in the query and ontological structures and thus fully rely on the availability of the necessary lexical knowledge in the resource used. This assumption is especially critical in the light of highly technical domains.

On the downside, for approaches similar to ORAKEL, which rely on an explicitly represented mapping between language and ontology, the lexicon needs to be created and constantly updated. However, we have shown in our experiments that the time needed to engineer a lexicon for a medium size ontology is in the range of hours, not of days or months as for older NLIs. According to Bates [5], porting PARLANCE takes between 6 and 8 person weeks for databases with between 32 and 75 fields. Such an effort is enormous compared to the one presented in this article. Our model for acquiring lexical knowledge is not inherently novel. In fact, a similar model was explored already in the TEAM interface, which has clearly inspired the design, architecture and implementation of ORAKEL. Nevertheless, we are not aware of any other work in the literature which has proven that a lexical acquisition model as integrated in TEAM is actually feasible. We have closed this gap and provided evidence that such a lexicon acquisition model is indeed feasible from the perspective of a naive user. In particular, our experiments have shown that subcategorization frames, which are central in our system design, represent a level of linguistic representation which can be understood by people without any background in computational or formal linguistics. In particular, our experiments have shown that the very rudimentary knowledge about subcategorization frames needed can be quickly acquired by domain experts. We have conducted extensive experiments with 26 users, mostly computer-scientists with no background in computational linguistics. Our extensive experiments with the ORAKEL system clearly corroborate the hypothesis that our model represents a feasible approach for domain adaptation. On the one hand, our results have shown that the domain lexica created by the two users in charge of the lexicon creation do not substantially differ from the one created by one of the authors, a computational linguist, with respect to the coverage of the system. In a further real-world case study at British Telecom, we have further shown that ORAKEL can indeed scale to knowledge bases with thousands of instances. The additional use case has also provided additional proof that our iterative lexicon development methodology is indeed successful. On the other hand, we have demonstrated that, given a suitable mechanism to generate grammatical structures out of the subcategorization frames, by sticking to the few types defined in our system we achieve more than 90% of linguistic coverage with respect to the questions asked, assuming that the lexicon is complete.

Further, it is also important to mention that ORAKEL builds on a clear separation between the two roles of end user and lexicon engineer. The lexicon engineer is assumed to be a person with computer sci-

ence background which can be trained to understand the adaptation model underlying ORAKEL. Such a lexicon engineer could for example work for a company which distributes natural language interfaces. The lexicon engineer's role would be one of constantly monitoring the suitability of the lexicon in the different applications and updating it appropriately. The adaptation model will of course remain completely transparent to the end user, whose role is only to ask questions and receive answers. Most of the recent approaches discussed in the related work section have focused on end users. However, assuming that NLIs will work completely off-the-shelf seems quite unrealistic given the current state-of-the-art. In fact, it seems reasonable to assume that some basic tuning will be necessary. A developer will thus be needed in any case. In our model, this developer will then be in charge of creating and maintaining a suitable lexicon for the application in question.

On a final note, it is important to emphasize that given the current state-of-the-art concerning natural language interfaces, a systematic analysis of the strengths and limits of approaches based on lexical matching in comparison to approaches based on an engineered lexicon has not yet been provided. At first sight, approaches based on lexical matching are certainly appealing as they require no manual adaptation. However, this is only true at the surface as there will always be cases which can not be covered with the current state of the lexical resources used in the background. On the other side, the standard drawback of approaches such as implemented in ORAKEL is the fact that the lexicon has to be partially engineered by hand. For sure, we will need to find a compromise between using existing resources, handcrafting new ones from scratch and techniques which can adapt the lexicon on the fly. With the work described in this article, we have contributed a model and a clear analysis thereof in terms of the efforts and skills needed by a lexicon engineer to handcraft a lexicon for a certain domain. Thus, our research contributes towards understanding and clarifying the principles underlying the challenging issue of adapting natural language interfaces to a certain domain.

Future work will also aim at decoupling our approach from our own parser, thus allowing to use any parser and syntactic theory. This will also allow for the usage of languages other than English. A major bottleneck of ORAKEL is that it is not robust in the sense that it fails when the input is not grammatical or when the input contains unknown words. Currently, it simply reports the unknown word to the user. Concerning ungrammatical input, our system can be in principle extended to use the subcategorization frames to generate a feedback question to the user asking for an alternative and grammatical paraphrase. For example, if the user asks *Who published?*, the system could then use the subcategorization information to return the feedback question: "*How published what?*" A further issue is the problem of what to do in the case there are no answers which satisfy the query (or one part of it) in the knowledge base (compare [33]). In this case an explanation why there is no answer should be given to the user. This issue is currently not addressed by the ORAKEL system and thus remains for future work.

On a more general note, given that the application barrier for natural language technologies is still very high for end users, future work should indeed take up the challenge of developing models by which this barrier can be effectively lowered, thus enabling a wider application and possibly commercialization of natural language processing technologies. The customization model underlying ORAKEL can be certainly regarded as a first step in this direction. Finally, an interesting question for future research is whether the mappings between linguistic expressions and relations defined in the knowledge base can be learned automatically from a corpus. This idea was already preliminary explored in [14,19], but further research is needed to clarify its full potential as well as weaknesses.

Acknowledgements

This research has been supported by the following projects: the BMBF project SmartWeb,[19] financed by the German Ministry of Education and Research as well as the EU projects Dot.Kom,[20] SEKT[21] and X-Media.[22]

[19] http://www.smartweb-project.de/.
[20] http://nlp.shef.ac.uk/dot.kom/.
[21] http://www.sekt-project.com/.
[22] http://www.x-media-project.org/.

Thanks to our students Johanna Wenderoth and Laura Goebes for creating the German geography knowledge base. Thanks to all our colleagues from the AIFB, the FZI and ontoprise as well as to Ursula Cimiano, Sofia Pinto and the people from British Telecom for taking part in our experiments.

References

[1] I. Androutsopoulos, G. Ritchie, P. Thanisch, Natural language interfaces to databases – an introduction, Journal of Language Engineering 1 (1) (1995) 29–81.
[2] A. Ankolekar, P. Buitelaar, P. Cimiano, P. Hitzler, M. Kiesel, M. Krötzsch, H. Lewen, G. Neumann, M. Sintek, T. Tserendorj, R. Studer, Smartweb: mobile access to the semantic web, in: Proceedings of the ISWC 2006 Poster and Demo Session, 2006.
[3] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, 1999.
[4] C. Baker, C. Fillmore, J. Lowe, The Berkeley FrameNet project, in: Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics (COLING-ACL), 1998.
[5] M. Bates, Rapid porting of the parlance natural language interface, in: Proceedings of the Workshop on Speech and Natural Language, 1989, pp. 83–88.
[6] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinees, P. Patel-Schneider, L. Stein, OWL Web Ontology Language Reference. <http://www.w3.org/TR/owl-ref>, 2004.
[7] A. Bernstein, E. Kaufmann, A. Göhring, C. Kiefer, Querying ontologies: a controlled english interface for end-users, in: Proceedings of the 4th International Semantic Web Conference (ISWC), 2005, pp. 112–126.
[8] P. Blackburn, J. Bos, Representation and Inference for Natural Language – A First Course in Computational Semantics, CSLI Publications, 2005.
[9] S. Bloehdorn, P. Cimiano, A. Duke, P. Haase, J. Heizmann, I. Thurlow, I. Völker, Ontology-based question answering for digital libraries, in: Proceedings of the 11th European Conference on Research and Advaced Technologies for Digital Libraries (ECDL), 2007, pp. 14–25.
[11] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: a generic architecture for storing and querying rdf and rdf schema, in: Proceedings of the International Semantic Web Conference (IWSC), 2002, pp. 54–68.
[12] P. Cimiano, Translating wh-questions into F-Logic queries, in:R. Bernardi, M. Moortgat (Eds.), Proceedings of the CoLogNET-ElsNET Workshop on Questions and Answers: Theoretical and Applied Perspectives, 2003, pp. 130–137.
[14] P. Cimiano, ORAKEL: a natural language interface to an F-Logic knowledge base, in: Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB), 2004, pp. 401–406.
[15] P. Cimiano, P. Haase, Y. Sure, J. Völker, Y. Wang, Question answering on top of the BT digital library, in: Proceedings of the World Wide Web conference (WWW), 2006, pp. 861–862.
[16] P. Cimiano, U. Reyle, Towards foundational semantics – ontological semantics revisited, in: Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS), vol. 150, IOS Press, 2006, pp. 51–62.
[17] P. Cimiano, P. Haase, J. Heizmann, Porting natural language interfaces between domains – a case study with the ORAKEL system, in: Proceedings of the International Conference on Intelligent User Interfaces (IUI), 2007, pp. 180–189.
[18] P. Cimiano, P. Haase, J. Heizmann, M. Mantel, Orakel: a portable natural language interface to knowledge bases, Technical report, Institut AIFB, Universität Karlsruhe. <http://www.aifb.uni-karlsruhe.de/WBS/pci/Publications/orakel_tech.pdf>, 2007.
[19] P. Cimiano, M. Erdmann, G. Ladwig, Corpus-based pattern induction for a knowledge-based question answering approach, in: Proceedings of the 1st IEEE International Conference on Semantic Computing (ICSC), 2007, pp. 671–678.
[20] A. Copestake, Robust Minimal Recursion Semantics (working draft). <http://www.cl.cam.ac.uk/aac10/papers/rmrsdraft.pdf>, 2006.
[21] A. Copestake, K.S. Jones, Natural language interfaces to databases, Knowledge Engineering Review 5 (4) (1989) 225–249 (Special issue on the Applications of natural language processing techniques).
[22] H. Cunningham, K. Humphreys, R. Gaizauskas, Y. Wilks, GATE – a General Architecture for Text Engineering, in: Proceedings of Applied Natural Language Processing (ANLP), 1997, pp. 29–30.
[23] S. Decker, M. Erdmann, D. Fensel, R. Studer, Ontobroker: ontology based access to distributed and semi-structured information, in: Database Semantics: Semantic Issues in Multimedia Systems, Kluwer, 1999, pp. 351–369.
[24] P. Dekker, Existencial disclosure, Linguistics and Philosophy (16) (1993) 561–587.
[25] E. Bozsak et al., KAON – towards a large scale semantic web, in: Proceedings of the Third International Conference on E-Commerce and Web Technologies (EC-Web), Springer Lecture Notes in Computer Science, 2002.
[26] C. Fellbaum, WordNet, an Electronic Lexical Database, MIT Press, 1998.
[27] A. Frank, H.U. Krieger, F. Xu, H. Uszkoreit, B. Crysmann, B. Jörg, U. Schäfer, Question answering from structured knowledge sources, Journal of Applied Logic 5 (1) (2007) 20–48 (Special issue on Questions and answers: theoretical and applied perspectives).
[28] N. Fuchs, K. Kaljurand, G. Schneider, Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces, in: Proceedings of the International Conference of the Florida Artificial Intelligence Research Society (FLAIRS), 2006.
[29] B. Grosz, D. Appelt, P. Martin, F. Pereira, Team: An experiment in the design of transportable natural language interfaces, Artificial Intelligence 32 (1987) 173–243.
[30] P. Haase, J. Broekstra, A. Eberhart, R. Volz, A comparison of RDF query languages, in: Proceedings of the 3rd International Semantic Web Conference (ISWC), 2004.
[31] A. Joshi, Y. Schabes, Tree-adjoining grammars, Handbook of Formal Languages, vol. 3, Springer, 1997, pp. 69–124.

[32] H. Kamp, U. Reyle, From Discourse to Logic, Kluwer, 1993.
[33] M. Kao, N. Cercone, W.S. Luk, Providing quality responses with natural language interfaces: the null value problem, IEEE Transactions on Software Engineering 14 (7) (1988) 959–984.
[34] M. Kifer, G. Lausen, J. Wu, Logical foundations of object-oriented and frame-based languages, Journal of the ACM 42 (1995) 741–843.
[35] V. Lopez, M. Pasin, E. Motta, Aqualog: an ontology-portable question answering system for the semantic web, in: Proceedings of the European Semantic Web Conference (ESWC), 2005, pp. 546–562.
[36] V. Lopez, E. Motta, V. Uren, Poweraqua: fishing the semantic web, in: Proceedings of the European Semantic Web Conference (ESWC), 2006, pp. 393–410.
[37] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, Ontology library (final), WonderWeb deliverable D18, 2003.
[38] D. McGuiness, F. van Harmelen OWL Web Ontology Language Overview. W3C Recommendation. <http://www.w3.org/TR/owl-features/>, 2004.
[39] R. Montague, On the proper treatment of quantification in ordinary english, in: R.H. Thomason (Ed.), Formal Philosophy: Selected Papers of Richard Montague, 1974, pp. 247–270.
[40] R. Muskens, Talking about trees and truth-conditions, Journal of Logic, Language and Information 10 (4) (2001) 417–455.
[42] W. Ogden, P. Bernick, Using natural language interfaces, in: M. Helander (Ed.), Handbook of Human–Computer Interaction, Elsevier, 1996.
[43] A. Pease, I. Niles, J. Li, The suggested upper merged ontology: a large ontology for the semantic web and its applications, Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web, 2002.
[44] A. Popescu, O. Etzioni, H. Kautz, Towards a theory of natural language interfaces to databases, in: Proceedings of the International Conference on Intelligent User Interfaces (IUI'03), 2003, pp. 149–157.
[45] E. Prud'hommeaux, A. Seaborne, Sparql query language for rdf, W3C Working Draft 4. <http://www.w3.org/TR/rdf-sparql-query/>, 2006.
[46] C. Rosé, C. Pai, J. Arguello, Enabling non-linguists to author advanced conversational interfaces easily, in: Proceedings of the International Conference of the Florida Artificial Intelligence Research Society (FLAIRS), 2005, pp. 572–577.
[47] Y. Schabes, A. Abeille, A. Joshi, Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars, in: Proceedings of the International Conference on Computational Linguistics (COLING'88), 1988, pp. 578–583.
[48] Y. Schabes, A. Joshi, An earley-type parsing algorithm for tree adjoining grammars, Technical Report MS-CIS-88-36/LINC LAB 113, University of Pennsylvania, 1988.
[49] H. Schmid, Probabilistic part-of-speech tagging using decision trees, in: Proceedings of the International Conference on New Methods in Language Processing, 1994.
[50] B. Shneiderman, C. Plaisant, Designing the User Interface, Pearson/Addison-Wesley, 2005.
[52] C. Thompson, R. Mooney, L. Tang, Learning to parse natural language database queries into logical form, in: Proceedings of the Workshop on Automata Induction, Grammatical Inference and Language Acquisition, 1997.
[53] P. Warren, D. Alsmeyer, Applying semantic technology to a digital library: a case study, Library Management 26 (4/5) (2005) 190–195 (Special issue: Semantic web).
[54] L. Zadeh, The concept of a linguistic variable and its application to approximate reasoning, Information Sciences (1975) 8–9.

Philipp Cimiano is a senior researcher and project leader at the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe. He obtained his Ph.D. in the field of ontology learning from text in 2006 from the University of Karlsruhe. His major research interests include: natural language interfaces, information extraction, ontology learning, knowledge acquisition from text, natural language processing, computational semantics and ontologies. Currently, he is local project leader of the EU IST project X-Media and the DFG-funded project MULTIPLA, both concerned with information extraction, information retrieval and question answering. Philipp has co-authored over 50 publications in the fields of natural language processing, text mining, ontologies and Semantic Web.

Peter Haase is a senior researcher at the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe, where he obtained his Ph.D. in 2006. Before joining the AIFB, he worked in the Silicon Valley Labs of IBM in the development of DB2 until 2003. His research interests include ontology management and evolution, decentralized information systems and Semantic Web.
At the AIFB, he previously worked in the EU IST projects SWAP (Semantic Web and Peer-to-Peer) and SEKT (Semantically Enabled Knowledge Technologies) and is now working as a project leader for the EU IST project NeOn (Lifeycle Support for Networked Ontologies).

Jörg Heizmann graduated in computer science (diploma, M.Sc.) at University of Karlsruhe in 2007. His student research project was on the topic "Conjunctive Queries in Semantic Peer-to-Peer Networks based on Adaptive Overlays". His graduation project was carried out in cooperation with British Telecom in the SEKT project on the topic: "Semantic-Based Digital Libraries - architecture, implementation and evaluation". Currently, he is working at ontoprise GmbH, a leading software provider of ontology-based solutions. There, he is involved in the halo project which is an effort by Vulcan Inc. towards the development of a "Digital Aristotle".

Matthias Mantel is a computer science student at the University of Karlsruhe since 2001. He has been working for the Institute AIFB as a student assistant at the group of Prof. Studer since 2003 as developer of the ORAKEL project. In particular, he has been in charge of the development of FrameMapper and of ORAKEL's lexicon model. Recently, he has finished a student research project on the topic of representing ORAKEL's lexica as OWL ontologies.

Rudi Studer is Full Professor in Applied Informatics at the University of Karlsruhe, Institute AIFB. His research interests include knowledge management, Semantic Web technologies and applications, ontology management, text mining and semantic web services. He is also member of the board of the FZI Research Center for Information Technologies at the University of Karlsruhe and one of the directors of the Research Group "Information Process Engineering" at the FZI; he is also a co-founder of the spin-off company ontoprise GmbH that develops semantic applications. He has been an associate editor of ACM TOIT (Transaction on Internet Technology) since 2002 and became a member of the advisory board of "IEEE Intelligent Systems" in 2005. From 2003 to 2007 he was Editor in Chief of the "Journal of Web Semantics: Science, Services and Agents on the World Wide Web". He is currently also technical director of the EU funded Integrated Project NeOn (Lifecycle Support for Networked Ontologies) and president of the Semantic Web Science Association.