Porting Natural Language Interfaces between Domains – An Experimental User Study with the ORAKEL System –

Philipp Cimiano, Peter Haase, Jörg Heizmann
Institute AIFB
University of Karlsruhe
{cimiano,haase,johi}@aifb.uni-karlsruhe.de

ABSTRACT

We present a user-centered model for porting natural language interfaces (NLIs) between domains efficiently. The model assumes that domain experts without any background knowledge about computational linguistics will perform the customization of the NLI to a specific domain. In fact, it merely requires familiarity with the underlying knowledge base as well as with a few basic subcategorization types. Our model is iterative in the sense that the adaption of the NLI is performed in several cycles on the basis of the questions which the NLI failed to answer, thus iteratively increasing the coverage of the system. We provide experimental evidence in form of a user study as well as a case study involving a real-world application corroborating that our model is indeed a feasible way of customizing the interface to a certain domain.

ACM Classification: H5.2 [Information interfaces and presentation], H3.3. [Information Search and Retrieval]

Keywords: Natural Language Interfaces, Portability, Evaluation

General terms: Experimentation, Human Factors

Introduction

Due to the steady growth of the amount of information and services available on the Web or within intranets as well as to the proliferation of ubiquitous and mobile devices such as cell phones or PDAs, the need for intuitive and effective user interfaces becomes more and more crucial. Because of the limited input/output functionality of mobile devices, natural language interfaces (NLIs) have recently received renewed interest [21] for querying information. Natural language interfaces essentially take as input a question formulated in natural language and return an answer to this question from a given knowledge base. Besides, their potential for being used for querying information, they have also further important applications within dialog and tutoring systems, which users typically communicate with in plain English.

However, it is well-known that robust natural language interfaces are very difficult to realize as they have to handle difficult problems inherent in the task of automatically interpreting natural language (compare [1] and [9]). A further obstacle for the applicability of natural language interfaces is the fact that they are typically difficult to port to other domains. In this paper, we present a user-centered model for porting NLIs between domains, also showing that our system is reasonably precise and robust.

Our user-centered model for porting the NLI merely requires very basic knowledge about subcategorization frames, but no background in computational linguistics. Subcategorization frames are essentially linguistic argument structures, e.g. verbs with their arguments, nouns with their arguments, etc. As in the TEAM natural language interface [12], we also assume that a user with general expertise about computer systems will perform the customization. In our view, this is a crucial requirement which many NLIs fail to meet. The main task of the person in charge of customizing the system is to create a domain-specific lexicon mapping subcategorization frames to relations as 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 (natural language processing) experts to create an appropriate domain-lexicon efficiently and effectively. In particular, we show that the results obtained with lexica customized by non-NLP experts do not substantially differ from the ones created by an NLP expert. 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 obtain a reasonable linguistic coverage.

The paper is structured as follows: in the next section, we briefly describe the natural language interface. In section *Domain Adaption* we present our model of domain adaption and the graphical user interface used in the experiments is introduced in section *Graphical User Interface*. The results of our user study as well as the conclusions drawn from applying our model to a case study involving a real-world application are presented in section *Experiments*. Before concluding, we discuss related work.

The Natural Language Interface

Our natural language interface implements a compositional semantics approach (see [18, 4]) to construct the logical formula corresponding to the input question. Compositional means here that the query to the database or knowledge base - i.e. its meaning - is recursively computed on the basis of the meaning of every single word in the input sentence. 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. Thus, the logical query representing a question is constructed en par with the syntactic analysis of the question. We use in particular the lambda calculus in our compositional approach and reuse the implementation described in [4]. Such a compositional approach to interpretation requires relatively rich lexical resources specifying the logical meaning of each word. This is exactly where our user-centered model for NLI adaption fills a gap as the rich semantic lexicon is generated in the background as a byproduct of the interaction of the user with the system's lexicon acquisition frontend, called *FrameMapper* (see section *Graphical User Interface*). Details about the semantics of each word remain completely hidden to the user. Indirectly, the user 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 below.

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. Roughly, the meaning of the diverse syntactic units in the input can be expressed as follows in functional lambda notation:

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 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 and *'Berlin'* expects 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 Figure 1 and evaluating the tree in a standard bottom-up fashion, we would first carry out the functional application:
$\lambda u\ (\lambda Q\ Q(Berlin))((\lambda x\, \lambda y\ flow\_through(x, y))(u))$,
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:
$?x\ (river(x) \wedge flow\_through(x, Berlin))$.

The underlying syntactic theory of our system is an LTAG-inspired[1] lexicalized formalism called Logical Description Grammars (LDGs) (compare [19]). In our approach, we reuse a version of LDG augmented with selectional restrictions specified with respect to an ontology as described in [5]. This ontological information is used to impose restrictions on the arguments of a predicate and thus for disambiguation.

The structures used in the formalism are essentially trees with nodes labeled with syntactic information as depicted in Figure 1. In this formalism parsing boils down to identifying positively and negatively marked nodes, respecting category information and surface order of words. 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. For more details about the formalism the user is referred to [6].

The ORAKEL system implements a procedural version

[1]Lexicalized Tree Adjoining Grammars (LTAGs) are essentially tree-rewriting systems with two operations: substitution and adjoining which exceed the computational power of context-free grammars and have been successfully used to model certain (syntactic) natural language phenomena (compare [14]).

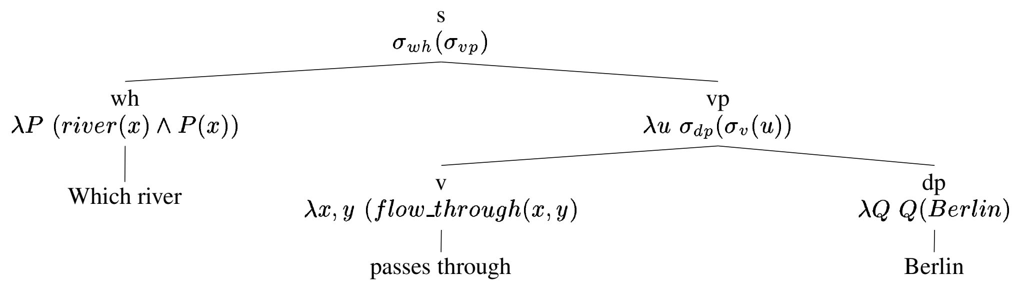

s
$\sigma_{wh}(\sigma_{vp})$
wh
$\lambda P\ (river(x) \wedge P(x))$
Which river
vp
$\lambda u\ \sigma_{dp}(\sigma_v(u))$
v
$\lambda x, y\ (flow\_through(x, y)$
passes through
dp
$\lambda Q\ Q(Berlin)$
Berlin

Figure 1: 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.

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 [22]. Firstly, appropriate elementary trees for each word in the input are selected from the lexicon. Secondly, these elementary trees are combined to yield a parse of the sentence (compare [22]). As it is not the focus of this paper, we refrain from a more detailed description of our implementation of LDG and refer the interested reader to [6].

In order to increase its flexibility, the system has been designed to be domain-independent on the one hand and independent of the specific knowledge representation and query languages used in the background on the other. Domain-independence is achieved by separating the general and domain lexica as is typically done for portable NLIs (compare [12]). 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. Queries in this FOL-like language can then be translated to any logical language by a translation component. Hereby, the translation is specified declaratively and is thus exchangeable. Currently, our system supports two formalisms used in the Semantic Web, the Web Ontology Language (OWL)[2] with the query language SPARQL[3] as well as F-Logic as ontology language together with its corresponding query language [15]. 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 extra-logical predicates for counting and for numerical comparisons.[4]

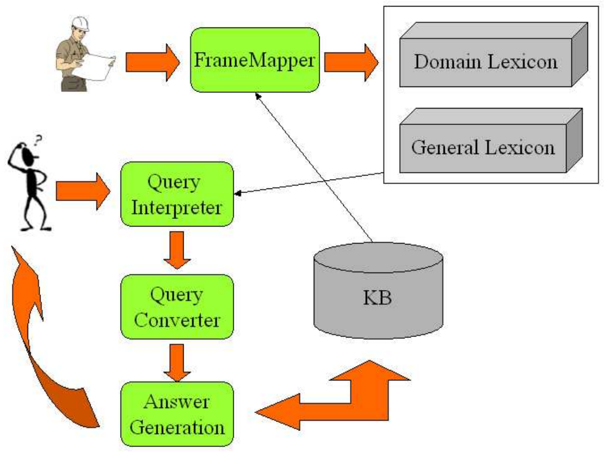


Figure 2: Overview of the ORAKEL system

Figure 2 gives an overview of the system. It shows that there are two lexicons: a domain-specific and a general one. The domain-specific lexicon is generated by a domain expert with the help of the *FrameMapper* lexicon creation frontend, which is described in detail in section *Graphical User Interface*. The domain lexicon is partially generated from the knowledge base. The query of an end user is interpreted by the *query interpreter*, which transforms it into an FOL-like query as described above. The *query converter* is a Prolog program which then transforms the FOL formula into a query specified in the target query language.

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. Factoid in this context means that answers are ground facts as typically found in knowledge or data bases, but not complex answers to *why*- or *how*-questions asking for explanations, the manner in which something happens or the cause of some event. From a syntactic point of view, the

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

system allows relative clauses, coordination of determiner phrases (DPs) and noun phrases (NPs), (numerical) quantification and negation.

After having given an overview of our natural language interface as well as explained how it applies the principle of compositional semantic construction to interpret users' questions, in the next section we describe in detail the crucial process by which a domain lexicon is constructed in interaction with the user.

Domain Adaption

In our system we pursue an approach in which the domain lexicon is constructed in interaction with the user, whose task is to map relations in the knowledge base to appropriate subcategorization frames for verbs and nouns, as well as adjectives. Before explaining in detail the underlying model which allows a user to create a domain-specific lexicon and thus customize the system to a certain domain, it is important to mention that the overall lexicon of the system has a tripartite 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
- an *ontological lexicon*, containing lexical entries and the semantics of instances and concepts which are typically represented linguistically as proper nouns and nouns, respectively.

The only lexicon component which has to be generated by the user is the *domain-specific lexicon* in which verbs, adjectives and relational nouns are mapped to corresponding relations specified in the domain ontology. 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 representation of the words in this domain-independent lexicon thus makes reference to domain-independent categories as given for example by a foundational ontology such as DOLCE [17]. This obviously assumes 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 benefits of such a modularized approach can be found in [8]. The *ontological lexicon* is derived fully automatically from the domain ontology loaded into the system. 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 relies on 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 concepts, we use a lexicon with morphological information to generate the appropriate plural form.

The user is thus only involved in the creation of the domain-specific lexicon, which is actually the most important lexicon as it is the one containing the mapping of linguistic 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. In fact, the domain-specific part of the grammar is generated for every domain. The overall grammar consists exactly of the trees in the domain-independent lexicon, the ontological lexicon and the domain-specific lexicon. Thus, the task of the user is to actually provide a domain-specific grammar to the system. As this is a difficult task, 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 are organized in a type hierarchy, such that only structures of compatible arity are mapped onto each other. As shown in Figure 3, in our type system we distinguish between binary, ternary and quaternary subcategorization frames which can be mapped to binary, ternary and quaternary relations, respectively.

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

- transitive, e.g. *X borders Y*
- intransitive + PP-complement, e.g. *X flows through Y*
- noun + pp, e.g. *X is capital of Y*

For example, the user could create the mappings:

location(of: pcomp(city)) $\rightarrow$ location(city,loc) (1)
inhabitants(of: pcomp(city)) $\rightarrow$ inhabitants(city,integer) (2)
capital(of: pcomp(state)) $\rightarrow$ capital_of(city,state) (3)
length(of: pcomp(river)) $\rightarrow$ length(river,integer) (4)
flows(subj: river,through: pobj(river)) $\rightarrow$ flow_through(river,city) (5)
pass(subj: river,through: pobj(river)) $\rightarrow$ flow_through(river,city) (6)

height(of: pcomp(mountain)) → height(mountain,integer) (7)
border(subj: loc, obj:loc) → borders(loc,loc) (8)

Though these mappings may seem trivial, they are indeed crucial as a full domain-specific grammar mapping linguistic expressions to appropriate semantic representations will be generated on their basis. It is important to mention that 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 subcategorization frame. Therefore, it is necessary that the user also specifies the order in which the relation's arguments map to the ones of the subcategorization frame. For example, if the method *authorOf* is mapped to the transitive verb *write*, the order of arguments is reversed i.e. x writes $y \leftrightarrow y\ authorOf\ x$. For the noun subcategorization frames, the argument of the relation which has not been mapped to the *pcomp* position is stored separately from the actual frame as it will normally be expressed in form of a copula[5] construct such as *"What is the length of the Rhein?"*. Further, in case the preposition is *'of'*, the system also generates trees allowing to ask for the corresponding relation using the verb *'have'* (see the examples below). On the basis of the above example mappings, the system then generates elementary trees, such that it is able to interpret the following questions:

What is the length of the Rhein? (4)
What is the capital of Baden Württemberg? (3)
Which river flows through the capital of Baden Württemberg? (3+5)
Which rivers flow through a capital? (3+5)
Which river flows through the most cities? (5)
Which river flows through a state which borders Baden Württemberg? (5+8)
What is the height of the Zugspitze? (7)
Which countries does Baden Württemberg border? (8)
Which countries are bordered by Baden Württemberg? (8)
Which countries border Baden Württemberg? (8)
Which country borders the most states? (8)
How many inhabitants does Baden Württemberg have? (2)

In particular, for methods such as *capital_of*, which do not have a datatype such as a string or an integer as range, and which have been mapped to a noun+pp, the lexicon generation algorithm does not only create elementary trees for 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 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.

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

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 [12]). The positive/negative distinction is thus 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.

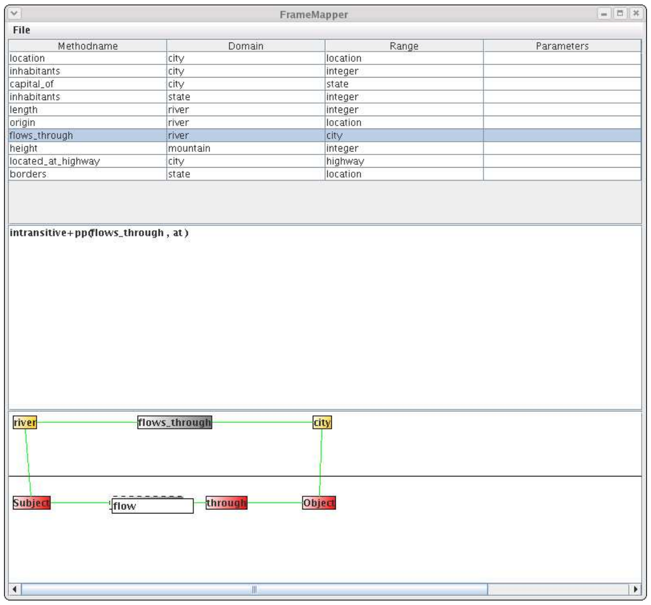
For details about the process generating grammar trees out of subcategorization frames, the interested reader is referred to [6]. The important aspect here is actually the fact that the domain-specific grammar necessary for understanding domain-specific expressions is generated in the background as a byproduct of a user 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.

Overall, the mapping model is not only restricted to binary relations as suggested in the type hierarchy shown in Figure 3. 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 ($2 \times$ 2-Join in the Figure), 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 ($2 \times$ 2-Join' in the Figure) or to a join of 3 binary methods ($3 \times$ 2-Join in the Figure), etc. The reason for introducing such a more or less 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.

In order to map relations defined in the ontology to appropriate subcategorization frames, users 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 user has assigned all the relations to corresponding subcategorization frames or adjectives, he can export the lexicon. It can then be used by the natural language interface to answer users' questions against the knowledge base. In our model, we do not expect a domain expert 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 in fact is that with such an iterative method the quality of the lexicon can be improved constantly. We will present experimental evidence for this hypothesis in section *Experiments*. An obvious alternative would consist in allowing the engineer to test his lexicon after each session without involving other users. However, during our experiments we found that a single user has often trouble in thinking of different variations for asking for the same fact, such that this approach can not be expected to achieve a high coverage. Nevertheless, this would be an interesting hypothesis to verify in future work.

Graphical User Interface

Figure 4 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 introduced by the preposition 'through'. The figure shows the three main panes of *FrameMapper*. In the top pane, the user sees the relations specified in the ontology. In the second pane, he can see the different subcategorization frames assigned to the active relation. In the third pane, he sees a visualization of the subcategorization frame and the selected relations as a graph. He can then graphically map the arguments of the frame to the ones of the selected relation(s). In the example, the lexicon engineer has 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 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 generate all the grammatical variations of the intransitive verb *'flow'* in the background, thus allowing to ask for the *flow_through* relation in a variety of ways. In order to add a further verb, e.g. *'pass'*, the user has to instantiate a new subcategorization 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. Note that the user can also select various relations and carry out joins between these to specify more complex mappings involving more than one relation. In general, the user can export the lexicon which can then be loaded into the ORAKEL natural language interface, but he can also import an already created mapping lexicon to add more subcategorization frames, thus creating the lexicon in several iterations.



Methodname	Domain	Range	Parameters
location	city	location	
inhabitants	city	integer	
capital_of	city	state	
inhabitants	state	integer	
length	river	integer	
origin	river	location	
flows_through	river	city	
height	mountain	integer	
located_at_highway	city	highway	
borders	state	location	

intransitive+pp(flows_through , at)

Figure 4: GUI of FrameMapper

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 system can be ported between domains without major efforts. First of all, 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 figures demonstrating that the system potentially has a reasonable linguistic coverage. Finally, we 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.

User study

The aim of the user study was to show that computer scientists without any NLP expertise can indeed generate reasonable domain-specific lexica 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

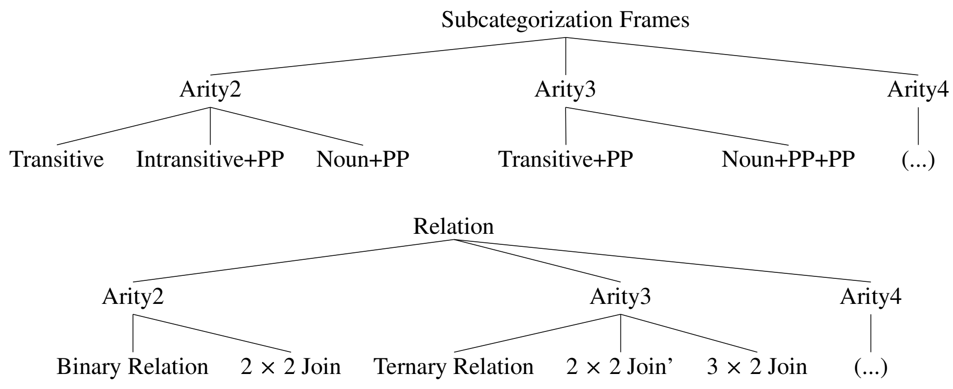


Figure 3: Type hierarchies of linguistic templates and relations

geographical facts about Germany. In particular, it contains states, cities, rivers and highways in Germany as well as the names of neighboring countries. It is a small knowledge base handcrafted by students at our department. 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. The relations defined in the ontology are the following:

city[location => 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 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 paper, as well as 26 additional test persons from 4 different institutions, both academic and industrial. Of these, 25 were computer scientists and 1 a graphic designer, all without any background in computational linguistics and thus suitable for our experiments. The role of the author as well as two of the other participants was to construct a lexicon each, while the rest played the role of end users of the system. We will refer to the author as A and the other 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 FrameMapper tool as well as 10 minutes 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 minutes. 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 4 sessions with different users. They were asked to complete the lexicon on the basis of the failed questions. Overall, they thus had one 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 for each user. 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. 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[6]. Table 1 shows these results for each of the

[6]It is important to emphasize that we are using precision and re-

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

Table 1: Results for the different lexica

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. 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 and precision to the lexicon developed by A. This shows that our lexical 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 constantly improve the lexica. However, this is subject to further analysis in future work.

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. A detailed manual analysis of the questions showed that with our basic subcategorization frames *transitive*, *intransitive+pp*, *np* and *adj* as well the constructions automatically generated from these, we get a linguistic coverage of more than 90%, i.e. more than 90% of the sentences could be handled from a grammatical and lexical point of view. This 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 opaque to the user. A more detailed presentation of this question analysis can be found in [6].

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[7]. In this case study the aim is to enhance the access to BT's digital library, which mainly contains metadata and full-text documents of scientific publications, by a natural language interface. 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[8], 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 also evaluated by taking into account the metadata in the database. The knowledge base contains metadata about 67015 persons, 17174 topics and 33501 documents (journal articles, conference papers, conference proceedings, periodicals and books). Further, there are 66870 instances of the *AuthorOf* relation and 165089 instances of the *isAboutTopic* relation. As the data size is indeed several orders of magnitude bigger compared to our geography domain, in the context of this use case it was totally unfeasible to generate a grammar entry for each instance in the database. Therefore, we performed a slight change to the ORAKEL system to allow to dynamically create grammar 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 syntax trees at runtime. The ontological 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 PhD student spent overall approx. 6 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. Though we have not carried out an extensive evaluation involving various users in this setting, the application to the BT digital library showed on the one hand that, given certain straightforward modifications, our approach can actually scale to much larger knowledge and data bases. On the other hand, 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 [7] for further details about the case study at British Telecom.

call here in line with Popescu et al. [21] and not in the standard information retrieval sense.

[7]http://www.sekt-project.com/

[8]http://proton.semanticweb.org/

Related Work

There is a vast amount of work related to the customization of NLIs to a certain domain. Discussing all the different approaches is out of the scope of this paper. For a detailed review of natural language interfaces, the interested reader is referred to the overviews in [9] and [1] as well as to the evaluation survey in [20]. Due to space limitations, we will only briefly discuss four well-known older and seminal systems (LADDER, TEAM, PARLANCE and ASK) as well as a few more recent approaches.

To overcome the difficulty in porting systems based on semantic grammars such as LADDER [13], systems such as TEAM, PARLANCE and ASK focused on supporting the portability of NLIs across domains by database experts. TEAM's [12] approach consisted in asking questions to a user to acquire linguistic knowledge about certain words, i.e. verbs, nouns, etc. as well as their relation to database fields. ASK [23] and PARLANCE [2] allowed users to teach new words and concepts even during execution time. However, to our knowledge, the only system of the above which has been evaluated in terms of the time taken to be customized is PARLANCE. According to [2], porting PARLANCE takes between 6-8 person weeks for databases with between 32 and 75 fields. Such an effort is enormous compared to the one presented in this paper.

More recently, the PRECISE system [21] has focused on the reliability of NLIs and presented a system which is formally proved to be 100% precise, given an appropriate domain-specific lexicon. In fact, PRECISE requires no additional customization nor domain-specific knowledge other than an appropriate lexicon. The precision of PRECISE of almost 100% on real data is certainly impressive, but in contrast to our approach it only produces conjunctive (SQL) queries. Furthermore, ORAKEL handles arbitrary quantification as well as negation and has also been demonstrated to be very reliable as the precision ranges between 74% and 85%. 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 for a NLI.

Customization to a domain in the system of Thompson et al. [24] is achieved by training a parser using ILP techniques on the domain in question. 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 usage point of view. The system is evaluated on two domains (jobs and geography), and achieves very decent accuracy levels between 25 - 70% for the geography domain and between 80 - 90% accuracy for the job domain, depending on the amount of training data used.

The approach in the QETAL system [10] implements the mapping from a question to a query via three intermediary stages: i) construction of a Robust Minimal Recursion Semantics (RMRS) representation, ii) mapping to FrameNet types and roles as well as iii) construction of so-called *proto-queries*. The approach implements a hybrid technique to interleave shallow and deep processing. An evaluation of the system shows that it achieves similar precision rates as our system, i.e. 74.1%. Customization is achieved through hand-written rewriting rules transforming FrameNet-like structures to domain-specific structures as provided by the domain ontology.

The recently presented AquaLog system [16] essentially transforms the natural language question into a triple-representation and then relies on similarity computations to map the triples to appropriate relations defined in the ontology. Some basic support for disambiguation is provided in this way. An obvious benefit is that AquaLog does not rely on any sort of customization.

Finally, the system presented by Bernstein et al. [3] builds on a controlled language approach as implemented by the ACE (Attempto Controlled English) framework (see [11]). It requires a transformation from the parser output structures – DRSs actually – as produced by the Attempto Parsing Engine (APE) to PQL queries formulated with respect to the relations and concepts of 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.

Conclusion

We have presented a new model for user-centered adaption of natural language interfaces to a certain domain. In line with TEAM [12], no knowledge about computational linguistics or NLP is required to customize the NLI in our model. Our experiments have shown that the very rudimentary knowledge about subcategorization frames needed can be quickly acquired by domain experts. Our extensive experiments clearly corroborate the hypothesis that our model represents a feasible approach for domain adaption. 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 accuracy of the system. Further, according to our lexicon developers, an iterative model is actually useful, but we plan to further test the effect of several iterations on the quality of the created lexica.

Acknowledgments This research has been supported by the following projects: the BMBF project SmartWeb, financed by the German Ministry of Education and Research as well as the projects Dot.Kom, SEKT and X-Media, funded by the European Union under FP6. Special credit goes to Matthias Mantel for help in implementing the system. 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 and Sofia Pinto for taking part in the experiment as end users.

REFERENCES

1. I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.
2. M. Bates. Rapid porting of the parlance natural language interface. In *Proceedings of the Workshop on Speech and Natural Language*, pages 83–88, 1989.
3. A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying ontologies: A controlled english interface for end-users. In *Proceedings of the 4th International Semantic Web Conference*, pages 112–126, 2005.
4. P. Blackburn and J. Bos. *Representation and Inference for Natural Language - A First Course in Computational Semantics*. CSLI Publications, 2005.
5. 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)*, pages 401–406, 2004.
6. P. Cimiano, P. Haase, J. Heizmann, and M. Mantel. Orakel: A portable natural language interface to knowledge bases. Technical report, Institute AIFB, University of Karlsruhe, 2007. to appear.
7. P. Cimiano, P. Haase, Y. Sure, J. Völker, and Y. Wang. Question answering on top of the BT digital library. In *Proceedings of the World Wide Web conference (WWW)*, pages 861–862, 2006.
8. P. Cimiano and U. Reyle. Towards foundational semantics - ontological semantics revisited -. In *Proceedings of the International Conference on Formal Ontology in Information Systems FOIS)*. IOS Press, 2006.
9. A. Copestake and K. Sparck Jones. Natural language interfaces to databases. *Knowledge Engineering Review*, 1989. Special Issue on the Applications of Natural Language Processing Techniques.
10. A. Frank, H.-U. Krieger, F. Xu, H. Uszkoreit, B. Crysmann, B. Jörg, and U. Schäfer. Question answering from structured knowledge sources. *Journal of Applied Logic, Special Issue on Questions and Answers: Theoretical and Applied Perspectives*, 2006. to appear.
11. N.E. Fuchs, K. Kaljurand, and G. Schneider. Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In *Proceedings of FLAIRS'06*, 2006.
12. B.J. Grosz, D.E. Appelt, P.A. Martin, and F.C.N. Pereira. Team: An experiment in the design of transportable natural language interfaces. *Artifi cial Intelligence*, 32:173–243, 1987.
13. G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978.
14. A.K. Joshi and Y. Schabes. Tree-adjoining grammars. In *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, 1997.
15. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.
16. V. Lopez and E. Motta. Aqualog: An ontology-portable question answering system for the semantic web. In *Proceedings of the International Conference on Natural Language for Information Systems (NLDB)*, pages 89–102, 2004.
17. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology library (final). WonderWeb deliverable D18.
18. R. Montague. On the proper treatment of quantification in ordinary english. In R. H. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, pages 247–270. 1974.
19. Reinhard Muskens. Talking about trees and truth-conditions. *Journal of Logic, Language and Information*, 10(4):417–455, 2001.
20. W.C. Ogden and P. Bernick. Using natural language interfaces. In M. Helander, editor, *Handbook of Human-Computer Interaction*. Elsevier, 1996.
21. A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of IUI'03*, pages 149–157, 2003.
22. Y. Schabes, A. Abeille, and A.K. Joshi. Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars. In *Proceedings of COLING'88*, pages 578–583, 1988.
23. B.H. Thompson and F.B. Thompson. ASK is transportable in half a dozen ways. *ACM Transactions on Offi ce Information Systems*, 3(2):185–203, 1985.
24. C. Thompson, R. Mooney, and 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.