Transportable Natural Language Interfaces for Taxonomic Knowledge Representation Systems

Lutz Prechelt, Finn Dag Buø, Rolf Adams
(prechelt|finndag|adams@ira.uka.de)
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe, W–7500 Karlsruhe, Germany

Appeared in the proceedings of the
Ninth IEEE Conference on
Artificial Intelligence for Applications
Orlando, Florida, March 1-5, 1993

A general approach is presented for building transportable natural language interfaces for question answering systems based on a KL-ONE-like knowledge representation. An example system, YAKR*, is described: The YAKS knowledge representation of concepts and relations is annotated with minimal syntactic information to generate a semantic case frame grammar with inheritance of cases. The generated grammar directs a case frame parser, which processes written input into instantiated case frames. These instantiatiations are easily translated into knowledge base queries. The same method is applicable to other object-oriented knowledge bases and other parsing techniques. The original contribution of this work is to show an approach with which natural language interfaces can with low effort be adapted to work with any new knowledge base: While most other systems require a complete model of the domain for the natural language interface knowledge representation, we derive most of this information from the application's knowledge base. This technique reduces the amount of work needed to create the interface to about additional 15 percent after building the knowledge base for the application kernel.*

AI topic: knowledge acquisition, knowledge representation, natural language interface
Language/Tool: Unix, C++
Status: implementation complete, evaluation in progress
Effort: about 4 person years
Impact: quick development of restricted natural language interface for certain classes of knowledge-based applications (15 % additional knowledge acquisition work).

1 Introduction

There have been several attempts to create natural language interfaces for databases [15, 10, 5]. Many of these natural language interfaces to databases are transportable, that is, they can be adapted to new domains by changing the lexicon and the domain-dependent knowledge. They usually rely on relational databases or Prolog databases. This basis influences the types of knowledge to be acquired and how this knowledge is represented. Generally, the effort to acquire the knowledge needed to use the natural language interface for a new domain is high.

In this paper we provide a framework of how natural language interfaces with low-effort knowledge acquisition may be realized for object-centered knowledge representation systems, in particular those similar to KL-ONE. The key property we strived for with our approach was cost-effective knowledge acquisition for the natural language interface. The way we achieved this property was basically to exploit as much as possible the knowledge about syntactic and semantic constraints that is implicit in the knowledge base.

The next sections describe the YAKS knowledge representation system, the SARA natural language processing component, and the knowledge acquisition. Then the limitations of both our current system and the approach in general are discussed. The approach is compared to related work.

2 The YAKS Knowledge Representation System

The YAKS knowledge representation language (in a former version named KRS [1]) distinguishes between terminological knowledge ("vocabulary" to be used to

express facts) and assertional knowledge (facts about *individuals (instances)* in the application domain).

The terminological knowledge consists of concept definitions and role definitions. A *concept* can be thought of as an abstract set of individuals. The concrete individuals that belong to a concept are called the *instances* of that concept. A *role* is a binary relation from a concept A to a concept B, i.e., a set of pairs of instances. A is called the *domain* of the role and B is called the *range* of the role. Concepts and roles are defined with constructors that each describe a subset of the set of all possible individuals or individual pairs, respectively.

The key property of YAKS in respect to the construction of the natural language interface is that the concepts in a YAKS knowledge base are implicitly arranged into a taxonomy, i.e., a hierarchy with multiple inheritance. The deductive capabilities of YAKS aim at deciding the subsumptional relations in this hierarchy. Since the YAKS language is very powerful, the deductions are not completely computable. Yet, we have not found a single case where this has been a problem in practice.

YAKS has a uniform language for data modeling (describing concepts and roles) and querying, i.e., all constructs that can be used to model a concept or role can also be used to query for one.

For a detailed description of YAKS see [2].

3 The Natural Language Processor

Since the effort to implement a sophisticated natural language processor is high, we chose to implement a case frame parser [16, 7], which is comparatively simple. The main disadvantage of case frame parsing is that complex syntactical constructions cannot be handled adequately. Note, that case frame parsing does not represent a state of the art technique in natural language processing — but this does not matter much for our purpose: We do not aim at building an especially powerful natural language interface, but one for which knowledge acquisition is especially simple.

However, the syntactic coverage of our case frame parser SARA [13], which targets German, is quite broad compared to other parsers using this technology; it handles the following syntactic phenomena: Noun phrases with determiners, adjectives, adverbs, and restricted numbering; simple conjunctions and disjunctions of noun phrases; all types of sentences (Yes/no–questions, Wh–questions, declaratives, imperatives, infinitives, to-be–sentences, to-have–sentences) in all tenses; modal verbs, immediate relative clauses, simple sentence-level negations, meaningless words, and subclauses starting with a conjunction (e.g. *...,because...*). Not handled are general and/or–conjunctions, general numbers, quantifiers, and many complex syntactical constructions.

Case frame parsers convert written input into instantiated case frames as an intermediate representation; no surface structure is generated. The parser is directed by built-in syntactic knowledge and a set of case frames, representing semantic knowledge. Thus, the parsing is semantically directed. A case frame represents an utterance by its central concept, called the *head* of the case frame, plus a set of *cases*, each of which describes (a) a certain semantic relation to the head and (b) the set of what may appear in that relation (the *fillers*). Each case frame represents a whole class of utterances, because some of the cases may or may not be present, cases can appear in any order, each case can have several different possible fillers, and each semantic relation can have several different possible grammatical representations.

In YAKR, case frames are never explicitly written by a user. Instead, we use the concept hierarchy of YAKS to build a corresponding case frame hierarchy, along which cases are implicitly inherited. A similar technique is used for the fillers: Instead of individual words, concepts are stated as allowed fillers. With any concept all of its subconcepts are legal fillers, too. For each of these concepts, a whole set of words or phrases can be given as natural language representations. Case frames instantiations can be nested. The grammatical representations that are possible for each of the semantic relations are listed in a separate *case table*. Entries in this case table have separate parts for occurrences of the case in nominal and in verbal context, if applicable, which enables to parse either a complete sentence or the corresponding nominalization with the same case frame.

4 Knowledge Acquisition and Generation of Case Frames

In our approach, the knowledge acquisition for the natural language interface consists of adding annotations to the knowledge base. These annotations are simple

and small, but suffice to construct all information the natural language interface needs. In this section we describe the type of information the annotations contain and how it is used in YAKR to generate the case frames. See [6, 14] for a detailed description.

There are two main types of information present in the annotations: (1) information about individual words and (2) information about grammatical constructions.

The word information associates each concept or role with its natural language synonyms (if any) and with simple phrases that represent this concept or role. The variety of the phrases covered by the annotations to a concept or role X is increased by using the inheritance mechanism to derive parts of these phrases from the annotations of superconcepts or superroles of X.

The information about grammatical constructions enables the generation of the case frames or other parser control information. This annotation is needed for roles only. In YAKR, it describes which cases must be inserted into which case frames. Nothing more than the name of a case table entry has to be given in it, since the exact set of case frames to put a case into and the corresponding filler for each insertion can be deduced from the YAKS modeling.

The annotations in YAKR are now described one by one; please refer to the example given in figure 1.

4.1 Word Annotations

The simplest form of annotation to a concept or role X is the synonym list: The `VERB` and `NOUN` annotations give a list of verbs and nouns, respectively, that can denote the concept or role X they are annotated to; the same word can be annotated to multiple concepts or roles. For instance, Zugriff (access) refers to the concept *access*. Similarly, abfragen (to query) refers to *read-access*. The latter is a verb with a separable prefix and is therefore given in two parts.

Often nouns can be specialized by prefixing an adjective. The `ADJECTIVE` annotation to X expresses this prefixing: It lists a number of adjectives that can be used to specialize a noun in order to denote X. The suitable nouns for this are all nouns that are annotated to any superconcept or superrole of X. This shows one of the ways inheritance is used in the construction of case frames: Whenever an annotation to X specifies a part P of a natural language construct to be *added* to another construct A, then A is inherited from the superconcepts or superroles of X. For instance, sequentieller Zugriff (sequential access) refers to *seq-access*, where Zugriff is inherited from the annotation of the direct superconcept *access*.

```
PRIM-CONCEPT access =
  AND(action  ;(not defined here)
      SOME (Agent, user)
      SOME (Theme, file)).
  VERB ((greifen zu)) ; to access
  NOUN (zugriff).     ; an access
  ROLES (Agent : user;
         AT-DOMAIN (agent-case)
         Theme : anything;
         AT-DOMAIN (thema-case)).

PRIM-CONCEPT read-access = access.
  VERB (lesen,         ; to read
        (fragen ab))   ; to query
  NOUN (lesen abfrage); a read, a query
  ADJECTIVE (lesend)   ; reading
  PREFIX (Lesen)       ; read-

PRIM-CONCEPT write-access = access.
  VERB (schreiben      ; to write,
        aendern)       ; to change
  NOUN (schreiben      ; a write
        aenderung)     ; a change
  PREFIX (Schreiben)   ; write-
  ADJECTIVE (schreibend) ; writing

PRIM-CONCEPT random-access = access.
  ADJECTIVE(wahlfrei, : random,
            direkt).  ; direct

PRIM-CONCEPT seq-access = access.
  ADJECTIVE(sequentiell). ; sequential

PRIM-CONCEPT user =
  ;(rest of definition left out)
  NOUN (benutzer).     ; a user

PRIM-CONCEPT file =
  ;(rest of definition left out)
  NOUN (datei).        ; a file
```

Figure 1: An annotated YAKS terminology

Compound nouns are written as a single word in German, so they could all be put into the dictionary and just annotated in the `NOUN` list. However, this would be extremely tedious, since compound nouns are very versatile and ubiquitous in German. To solve this problem, the `PREFIX` annotation to X lists possible prefix nouns that can be prepended to the nouns annotated with the superconcepts of X in order to form a reference to X. This annotation works analogously to the way the `ADJECTIVE` annotation works. For instance, Lesezugriff (read access) refers to *read-access*.

Only the words Zugriff and Lesen need to be in the dictionary, the compound is algorithmically broken into these components.

4.2 Grammatical Construction Annotations

All the above annotations merely describe phrases that represent individual concepts; no case frames are built from them. The source of cases for the case frames are the roles. This is where the information about grammatical constructions is used, which tells what cases to insert where. Similar Annotations could be used to generate the information that is needed to control other natural language processors.

As an example, the *Agent* role describes who performs an action, e.g., it relates an *access* to the *user* who performs it. Usually, actions are expressed by the main verb of a sentence, e.g., schreiben (to write) in the sentence der Benutzer schreibt (the user writes). The agent is the subject of such a sentence, e.g., der Benutzer (the user). What *"subject of the sentence"* means is described in the case table (not shown here) in an entry named *agent-case*. In this situation, the following cases must be generated: In the case frame of any concept X that specifies a filler F for the *Agent* role, an *agent-case* case with allowed filler F must be inserted. In the example given above, this happens for the concept *access* only; the generated case is an *agent-case* with filler *user*. This case is then inherited to the case frames of subconcepts.

In general, whenever the relationship expressed by a role can be represented as a case in a case frame, appropriate cases have to be inserted into the case frames of either the domain concept or the range concept of the role, sometimes into both. *What* to insert has to be given in the annotations: A case table entry is required that lists the correct syntactical forms for the case; the name of this entry has to be annotated to the role X as either `AT-DOMAIN`, meaning that the case is inserted in the case frames of the domain concept of X or as `AT-RANGE`, to insert into the case frames of the range concept of X. The filler in the `AT-DOMAIN` case is the range concept and vice versa. These cases are inherited via the case frame formalism, unless a concept explicitly specifies a filler when referring to a role: This more specialized filler is then used in the case frames of this concept and its subconcepts, resulting in stronger semantic restrictions for the parser.

In the example, the annotations at *Agent* and *Theme* result in case frames for *access* and its subconcepts that allow to parse, among others, the following inputs (none of the punctuation is necessary in YAKR):

Welche Benutzer lesen Datei A ?
(Which users read file A ?)
Schreibzugriffe auf Datei A ?
(Write accesses to file A ?)
Wird die Datei B sequentiell gelesen ?
(Is file B read sequentially ?)
Lesen der Datei C ?
(Reading of file C ?)
Benutzer, die Datei D schreiben ?
(Users that write file D ?)
Zugriffe zum Lesen auf Datei E ?
(Accesses for reading to file E ?)
Von welchen Benutzern wird Datei A gelesen ?
(By which users is file A read ?)

Not all concepts and roles can be handled the same way. For instance, concepts describing a property require case frames with to-be and to-have as head, while other concepts do not. To handle this problem we divide the concepts and roles into certain *syntactic categories*: Each concept or role is assigned to one of two concept categories (action/object, attribute) or one of four role categories (synonymless, theme, has-attribute, has-action/has-object). For these categories, the rules that apply for case frame generation are different, but static. To which category a certain concept or role belongs, can be computed from the taxonomic information and the word annotations. See [6, 14, 2] for details.

5 Interpretation of Parser Output

For an answering system, the case frame instantiations generated by the parser have to be interpreted in order to build a knowledge base query that corresponds to the user input.

Most systems need a complicated semantic analysis in order to interpret the output of the natural language parser. Such an analysis is not needed for YAKR: most of the restrictions are already enforced by the parser itself, since it is semantically directed; the remaining ambiguities are resolved in the query answering, because queries that make no sense will of course not return any result.

In YAKR, the interpretation of the parser output is a simple recursive conversion: Upon generation of the case frames, we add some information to every case that is used in this conversion process. This information describes a part of a YAKS query expression,

that has to be used in a query if the case containing it has been filled. Most of the instantiations can simply be processed in a top-down manner, only the instantiations for certain grammatical constructions require some more complicated processing. In any case, this processing is purely mechanical. No further semantic processing is necessary. Most ambiguities that remain after parsing need not be explicitly resolved, anyway, since the wrong interpretations result in queries that return no answer at all.

When a parsing technique other than case frame parsing shall be used, the interpretation of parser output gets more complicated, but it will always be possible to derive the information necessary to do it from the taxonomic knowledge base so that no additional knowledge acquisition for the natural language interface is needed.

6 Experiences, Limitations, Generality

A medium-sized example we made to test YAKR, took about 400 person hours for the domain modeling. YAKS was still unstable at that time and we did not know ourselves how to use it properly, so it is more realistic to calculate cost with a development time of 200 hours. The example consists of about 160 concepts, 130 roles, and 1000 instances and models C++ (which proved to be a very complicated domain) and a small part of the internal view of the NIH Class Library [9]. We needed less than 10 hours to add the annotations; thus, the additional effort was less than 5 percent.

A second example models the use and purpose of 69 Unix user commands [12]. It consists of 290 concepts (90 of which are actions), 170 roles, and 430 instances most of which are unnamed and are used to describe the actions. The development time of this model was about 110 person hours. Unix proved to be a less complicated domain, but the modeling was still difficult, especially since control of the model gets increasingly difficult with increasing size. Adding the annotations took 30 hours, which is about 27 percent additional effort. This difference in the relative cost of annotating compared to the first example has two reasons: One is that the annotator of the C++ examples was experienced in the annotation system while the annotator of the Unix example was not. The other is that the Unix domain was simpler, which saved time when modeling, but did hardly affect the work needed for the annotations.

Our experiences were that the natural language knowledge acquisition (i.e. adding the annotations) is indeed unproblematic, so the adaptability of the interface is high. But on the other hand the syntactic and semantic coverage of YAKR is too limited in practice.

In addition, the following effect occurs: Consider the (imaginary) set W of ways to express a certain abstract query in German natural language. Suppose we partition W into the set W^+ of inputs that can be handled by a given modeling M plus annotation, and the set W^- that cannot. Then we find, no matter how good the annotations are, that the structure of W^+ reflects the structure of M. In other words, in order to maximize a user's chance of being understood by the system, this user should know about the structure chosen for the internal knowledge model. This effect is of course present in any natural language system but may be more important for YAKR since its great power and flexibility makes it difficult to "guess" the internal knowledge structure correctly.

It is not quite clear how easily the annotations could be extended to cover more complex natural language utterances. Many extensions will be straightforward: For example, SARA is currently much more limited than the query language provided by YAKS, since, e.g., conjunctions cannot be handled; a corresponding extension would be useful. It would be no problem to replace our natural language processing component with a more sophisticated one, without any need to change the annotations at all. On the other hand, it will, for example, be difficult to describe classes of paraphrases by annotations.

7 Related Work

Two kinds of approaches for transportable natural language interfaces have to be discussed: (1) Integrated systems that are built for a certain application and (2) stand-alone interfaces that can be adapted for use with any application. We give one example of each plus a short discussion of an interface to another taxonomic knowledge base system.

TEAM [10] is a natural language interface to relational databases. TEAM makes use of four knowledge sources: the augmented phrase structure grammar, the lexicon, the conceptual schema, and the database schema. The grammar is fixed; the other knowledge sources have to be updated for a new database. The lexicon contains word information for morphological, syntactic, and semantic processing. With our ap-

proach, the lexicon can be reduced to a dictionary (without semantic information); the semantic information can be derived via the annotations. The conceptual schema consists of sort information and constraints on the arguments of nonsort predicates. With our approach, no such schema is necessary at all; the information can completely be deduced from the knowledge base itself. The database schema consists of information that enables the mapping of the intermediate representation to a query expressed in a relational query language. With our approach, as described above, this mapping can be done based on information that is also derived from the knowledge base using the annotations.

The Core Language Engine CLE [3], is a sophisticated general purpose natural language processor that can be coupled to any application. The syntactic processing works stand-alone, while the semantic processing must interact with the application to resolve ambiguities. Our approach can be used to integrate the CLE with an application that is based on a taxonomic knowledge base. To do that, for example the case frames could be interpreted as sets of semantic constraints. The main difficulty is that the conversion of parsing output into knowledge base queries (or other operations) gets more complicated.

Another integration of a natural language interface with a KL-ONE-like knowledge base is described rather briefly in [8]. The natural language component is realized by customizing the natural language processor TELI [4]. The knowledge structures of TELI are similar to those used in SARA, but there is no inheritance hierarchy for the case frames. Case frames have to be constructed by hand. The integration does not work with general modelings and restricts to frames describing actions.

8 Conclusion

If a natural language interface shall be added to an application that is based on a taxonomic knowledge base, it is not necessary that this natural language interface requires a costly and complicated knowledge acquisition process. By making effective use of the information contained in the knowledge base, only minimal additional information is necessary, in order to control a natural language parser and to interpret its outputs. This additional information has to provide a mapping from words to concepts and relations in the knowledge base and from grammatical constructions to relations in the knowledge base.

Since knowledge acquisition in taxonomic knowledge bases is a complicated process, the additional effort required for the natural language process is small; our experience indicates that the relative effort is about 15 percent, if modeling is done by hand. This makes cost-effective engineering of restricted natural language interfaces possible for a large class of knowledge based systems.

References

[1] Rolf Adams. KRS — a hybrid system for representing knowledge in knowledge–based help systems. In *[11]*, pages 129–133, 1991.

[2] Rolf Adams. *Wissensrepräsentation und -akquisition in einem natürlichsprachlichen Software-Informationssystem.* PhD thesis, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, November 1992.

[3] H. Alshawi, D.M. Carter, J. van Eijck, R.C. Moore, D.B. Moran, F.C.N. Pereira, S.G. Pulman, and A.G. Smith. Research programme in natural language processing: Final report. Technical report, SRI International, Cambridge Research Centre, July 1989.

[4] Bruce Ballard. A lexical, syntactic, and semantic framework for a user-customized natural language question-answering system. In Martha Evens, editor, *Lexical-Semantic Relational Models*, pages 211–236. Cambridge University Press, 1988.

[5] M. Bates, M. Moser, and D. Stallard. The IRUS transportable natural language database interface. In Larry Kerschberg, editor, *Expert Database Systems*, pages 617–630. The Benjamin/Cummings Publishing Company, 1986.

[6] Finn Dag Buø. A natural language interface for KRS. Master's thesis, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, D-7500 Karlsruhe, Germany, 1991.

[7] J. Carbonell and P. Hayes. Natural-language understanding. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 660–677. John Wiley & Sons, 1987.

[8] P. Devanbu, P. Selfridge, B. Ballard, and R. Brachman. A knowledge based software information system. In *International Joint Conference on Artificial Intelligence*, pages 110–115. Morgan Kaufmann Publishers, 1989.

[9] Keith E. Gorlen, Sanford M. Orlow, and Perry S. Plexico. *Data Abstraction and Object-Oriented Programming in C++*. Wiley, Chichester, 1991.

[10] B. Grosz, D. Appelt, P. Martin, and F. Pereira. TEAM: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32(1987):173–243, 1987.

[11] IEEE Computer Society. *The Seventh Conference on Artificial Intelligence Applications*, Miami Beach, Florida, February 1991. IEEE Computer Society Press.

[12] Matthias Ott. Modellierung von UNIX-Kommandos mit YAKR. Studienarbeit, Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation, D-7500 Karlsruhe, Germany, November 1992.

[13] Lutz Prechelt. Ein Fallschablonenzerteiler für Deutsch. Master's thesis, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, D-7500 Karlsruhe, Germany, 1989.

[14] Lutz Prechelt. The SIS project: Software reuse with a natural language approach. Technical Report 2/92, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, D-7500 Karlsruhe, Germany, June 1992.

[15] D. Warren and F. Pereira. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3-4):110–122, July-December 1982.

[16] Patrick Henry Winston. *Artificial Intelligence*. Addison Wesley, 1981.