The SIS Project:
Software Reuse with a Natural Language Approach

Lutz Prechelt (prechelt@ira.uka.de)
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe
Postfach 6980
D-7500 Karlsruhe, Germany

Technical Report 2/92
June 16, 1992

Abstract

The SIS (Software Information System) project investigated a new approach to one part of the software reuse problem. The problem is how to find and use a reusable component from a repository. The approach is:

(1) to provide a knowledge representation system that describes the components in the repository with user-defined semantic categories;
(2) to achieve ease of understanding and creation of knowledge by a natural language approach;
(3) to complement the formal query capabilities in this knowledge representation system.


Contents

1 Introduction
  1.1 Underlying Assumptions ........................................ 4
  1.2 Requirements .................................................. 4
  1.3 Basic Design .................................................... 5

2 The YAKS Knowledge Representation System ......................... 6

3 YAKS Knowledge Acquisition ........................................ 8
  3.1 Our Experimental Modeling ..................................... 9
  3.2 A Knowledge Acquisition Tool ................................ 9

4 The SARA Natural Language Processor ................................ 10

5 SARA Knowledge Acquisition ....................................... 11
  5.1 Information about Individual Words .......................... 13
  5.2 Information about Cases ...................................... 14
  5.3 Information about Explicit Inheritance of Words .......... 15

6 Integration of YAKS and SARA ...................................... 15
  6.1 Generation of Case Frames .................................... 15
    6.1.1 Action-or-Object Concepts ............................... 16
    6.1.2 Attribute Concepts .................................... 16
    6.1.3 Has-Action-or-Object Holes ............................ 16
    6.1.4 Has-Attribute Holes .................................. 16
    6.1.5 Theme Holes ........................................... 16
    6.1.6 Synonymless Holes .................................... 16
  6.2 Generation of Queries ....................................... 16

7 Experiences and Limitations ...................................... 17
  7.1 Usability ...................................................... 17
  7.2 Practicability ............................................... 17
  7.3 Acceptance .................................................... 17
  7.4 Scalability .................................................. 17
  7.5 Applicability ............................................... 17

8 Related Work ................................................................ 17
  8.1 Finding Reusable Components ................................ 17
  8.2 Transportable Natural Language Interfaces .................. 17

9 Results ...................................................................... 17

10 Further Work ................................................................ 17

A Other knowledge sources of YAKS .................................. 17
  A.1 SARA dictionary .................................................. 17
    A.1.1 Verbs ....................................................... 17
    A.1.2 Nouns ....................................................... 17
    A.1.3 Adjectives .................................................. 17
    A.2 SARA role table .............................................. 17

B List of Generated Case Frames ..................................... 17

C User interface commands ........................................... 17

D Example Session ..................................................... 17
CONTENTS

E. Possible Problem with Natural Language Interfaces 32

Bibliography 36
1 Introduction

The software component reuse problem can be split into two parts: (1) how to create and collect reusable components and (2) how to actually reuse them. The SIS project was only concerned with the second part.

The problem of how to reuse components can be further split into three parts: (a) how to find a suitable component for a given problem, (b) how to adapt it for the problem if necessary, and (c) how to use it correctly. The SIS project attempted to build a system that mainly addresses (a), and a system is called YAKR.

This subsection describes the assumptions YAKR is based on, the requirements that follow, and how these requirements are shaped into a concrete system design.

Assumptions

The assumptions that underly the design of YAKR:

- When looking for a reusable software component for a certain task, often no complete functionality of that component is available; the user's conception of what a component needed for a task X in a terminology that is not from the software components instead of reusing existing ones, unless reuse be maintained with enough care in the long run, if the database be kept consistent with a complicated schema.

Observations of many scientists in the area of software [2], although especially A4 is often not addressed at the system.

- Non-redundancy: Requirements as guidelines for the design

- Adequate and inaccurate specifications.

- The same request.

- Components must be easy to use.

In the SIS project, we picked one of them outlined in the following
1.3 Basic Design

R1 is realized with a specially designed knowledge representation language, YAKS (Yet Another Knowledge representation System), which has similarities to KL-ONE [8]. The constructs of the language are directly targeted to the description of software components and allow to define suitable terminology for software from any domain. This terminology is arranged in a taxonomy, which allows complete as well as inaccurate queries to be answered: they just retrieve elements that are more in respect to the taxonomy.

Satisfied by a natural language interface. Natural language is the most versatile way of expression. Natural language interfaces have the disadvantage that they are expensive to construct and maintain. We have found a way to minimize the work that is needed to construct the natural interface, too. Since natural language is the easiest way for a non-expert, especially if its conception is still fuzzy, the tendency not to use the system.

A maintenance problem in our system first, it must be easy to add new objects to the system. This is approached by letting the knowledge representation divide into a terminological part and an assertional part. The terminological part describes the actual objects in the repository in terms of the terminology that has once been defined. Thus, the base is easy; the repository.

This is only a
section discussing related work and a section that summarizes our results follow. Several appendices provide additional details.

2 The YAKS Knowledge Representation System

The YAKS knowledge representation language (in a former version named KRS [1, 17]) has well-defined model-theoretic semantics and distinguishes between assertional and terminological knowledge. The terminological knowledge defines a "vocabulary" to be used to express facts. The assertional knowledge
comprises facts about **individuals** 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 are defined with constructors that each describe a subset of the set of all possible individuals. Each constructor thus represents a restriction that an instance must adhere to in order to belong to the concept that is defined by the constructor. Rules are defined with constructors, too. A distinction between primitive concepts or roles (partial descriptions, describing conditions necessary), defined concepts or roles (exact descriptions, describing conditions that are both and sufficient), and derived concepts or roles (describing conditions that are sufficient, but

language in N3 is quite expressive, allowing value restriction, number restriction for
value maps, conjunction, disjunction, and negation. The concepts form a hierarchy
in which rules, restrictions, and parameterization. The role language allows conjunction, disjunction, domain
restriction, negation, and inversion.

The following definitions, excerpted from the larger example on page 11:

```
?ons = AND(Functions SOME(reads)).

a-Objects;).

?es = Objects.
```

and concept, i.e. exactly all those instances that obey the restrictions
seven belong to the concept.

that instances must obey, in order to belong to the concept _Input-
to the concept _functions_ and (b) they must have at least one

at least _Input-Functions_ and whose range contains at
of the role at other concept definitions may modify

those instances that obey the restrictions given
but we cannot know whether they really

_objects_ (whose definition is not shown

roles. The most important
determination whether a
descriptions all
fect placement of a
ter concepts
cepts
it belongs to. Finally, retrieval determines for a given concept description the set of all individuals that are instances of the concept. Retrieval is analogously defined for roles, producing a set of pairs of instances. The resulting taxonomy after classification of all concepts and realization of all instances for the larger example on page 11 is shown in figure 2 on page 13. Since the YASL language is so powerful, the inferences are not completely computable. Yet, we have not found a single case where this has been a problem in practice.

As a simple deduction example, look at the following definition, also excerpted from the larger example on page 11:

```
ANCE fgetc = AND(
  Functions
  VALUES(reads, [character-c])
  VALUES(has-Parameter, [filepointer-stream])
  VALUES(has-synonym, ['fgetc'])
).
```

definition means the following:

1. **fgetc** is an instance (individual).
2. **fgetc** belongs to the concept **Functions**.
3. **fgetc** has the instance **character-c** (whose definition is not shown, but which belongs to the class **Data-Objects**) as filler of the role **reads**.
4. **fgetc** has the instance **filepointer-stream** (whose definition is not shown, but which belongs to the class **Data-Objects**) as filler of the role **has-Parameter**.
5. String instance "**fgetc**" (which need not be defined, because it is a string) as filler **synonym**

YASL to infer that **fgetc** belongs not only to **Functions** but also to **Input-Actions**. All both restrictions given in the definition of **Input-Actions** and (i.e. any instance that adheres to all restrictions given in the concept) 

- **Killing** (describing concepts and roles) and **querying**, i.e., any concept or role can also be used to query for one. Thus, YASL is more expressive than a relational database.

- Interface, because the query language can be interpreted and how they can be

stages: First, define the

and, create the actual

ology that has
3.1 Our Experimental Modeling

This process resembles object-oriented design of a software system. First the classes (concepts) have to be described, i.e., “find out which kinds of objects exist and which of them are special cases of which others”. The better this modeling is, the easier the second stage will be: Define all the actual objects (instances) by picking a class (concept) for each of them and instantiating all its attributes (assigning its role fillers).

In practice, just as in object-oriented design, some backtracking will usually be necessary in order to get the terminology right. Our experience indicates that modeling in YAS is about as difficult as class design in an object-oriented programming language: If the task is complicated, modeling is a challenging task. But once the modeling is right, everything looks simple and clear.

An example of what a YAS modeling may look like will be given below in section 5.

1 Our Experimental Modeling

To learn about how modeling actually works and how good our system behaves on a medium-sized have modeled a part of the internal view of the NH Class Library [19], which is written complicated part of this task was to model the constructs of the C++ programming terminology contains about 160 concepts and 130 roles in 40 Kbytes of knowledge base for NHCL models only the top three classes of the (and the rest very roughly), but nevertheless contains almost concepts in 105 Kbytes of YAS source code.

model, although it must be mentioned that YAS was as well as changing. We also first had to learn the options and design flaws turned out to be

...
has-specification, has-specifier, has-subclass, has-superclass, has-synonym has-virtual-base-class

includes, inherits, is-data-type-of, is-declared-in, is-defined-in, is-enumeral

is-friend-of, is-included-by, is-inherited-by, is

class-of, is-private-re

However, broad compared

Noun phrases with simple conjunctions and questions, declaratives, imp

dal verbs, immediate relations starting with a conjunction: general numbers, general quer

Case frame parsers convert writt:

e no surface structure is gener

frames, representing semantic knowledge represents an utterance by its central co

each of which describes (a) a certain semant

relation (the fillers). There are verbal case fra

and nominal case frames describing noun phrases (wit

a whole class of utterances, because some of the cases may i

in any order, each case can have several different possible fillers, several different possible grammatical representations.

In our system case frames are never explicitly written by a user. Instea

of VAS to build a corresponding case frame hierarchy (there are also case

which cases are implicitly inherited. A similar technique is used for the fill

words, concepts are stated as allowed fillers. With any concept all of its supero

are legal fillers, too. For each of these concepts, a whole set of words or phrases of

language representations. Case frames can be nested when parsing: If, for example, ther

in a case frame is a noun that has a case frame associated with it, a complete instantiation

frame can be filled into that case. The grammatical representations that are possible for

semantic relations are listed in a separate case table. Entries in this case table have separate

occurrences of the case in nominal and in verbal context, if applicable, which enables to parse

complete sentence or the corresponding nominalization with the same case frame. This representation avoids tedious repetition and makes the representation compact and almost free of redundancy.
5 SARA Knowledge Acquisition

One reason why most natural language interfaces are not successful is that it requires too much work to adapt them to a new domain. In the design of YARR, we paid special attention to the problem of knowledge acquisition for the natural language interface. Ideally, nothing more should be necessary to acquire than the words which can be used to refer to each concept, role or instance. If this ideal is not approximated closely enough, it is necessary to specify complex grammatical descriptions; at least in this case the natural language interface will not be successful in practice.

YARR is very close to the ideal: The knowledge acquisition for the natural language interface in YARR consists of adding short annotations to each concept definition or role definition in the YARS. There are three main types of information present in the annotations: (1) information about words, (2) information about cases, and (3) information about explicit inheritance.

Associates each concept or role with its natural language synonym and with this concept or role. The variety of the phrases covered by the annotation increased by using an inheritance mechanism to derive parts of these concepts or explicitly stated syntactical superroles of X.

The generation of the case frames themselves. It describes which nouns. This annotation is needed for roles only. Nothing more is given, since the set of case frames to put the case information can be deduced from the YARS modeling.

For roles and for derived concepts in order phrases for these concepts or roles, although (a concept) from which the noun could be the following example. It should be shown how YARR works. It does (up-to-date) description
AT-DOMAIN(attribut).

PRIM-CONCEPT Functions = Objects.
  NOUN(unterprogramm funktion prozedur).
  ROLES(has-Parameter : Data-Objects;
  NOUN(parameter)).

DEF-CONCEPT Input-Functions = AND(Functions
  SOME(reads)).
  PREFIX(eingabe einlesen lesen input)
  SYN-CASE((zweck reads))
  ADJECTIVE(einlesend lesend).
  ROLES(reads : Data-Objects;
  VERB(lesen (lesen ein))
  NOUN(lesen eingabe einlesen input)).

DEF-CONCEPT Output-Functions = AND(Functions
  SOME(writes)).
  PREFIX(ausgabe ausgeben schreiben output)
  SYN-CASE((zweck writes))
  ADJECTIVE(ausgebend).
  ROLES(writes : Data-Objects;
  VERB(schreiben (geben aus))
  NOUN(schreiben ausgabe ausgeben output)).

PRIM-CONCEPT Data-Objects = Objects.
  NOUN(daten)
  PREFIX(daten).

DEF-CONCEPT Parameters = AND(Data-Objects
  SOME(is-Parameter-of)).
  NOUN(argument parameter)
  ADJECTIVE(uebergeben).
  ROLES(is-Parameter-of = INV(has-Parameter);).

PRIM-CONCEPT Characters = Data-Objects.
  NOUN(zeichen char character).

PRIM-CONCEPT Lines = Data-Objects.
  NOUN(zeile).

PRIM-CONCEPT Files = Data-Objects.
  NOUN(datei file).

INSTANCE character-c = Characters.
INSTANCE string-s = Lines.
INSTANCE filepointer-stream = Files.

INSTANCE fgetc = AND(Functions
5.1 Information about Individual Words

VALUES(reads, [character-c])
VALUES(has-Parameter, [filepointer-stream])
VALUES(has-synonym, ["fgets"]).

INSTANCE fputc = AND(Functions
VALUES(writes, [character-c])
VALUES(has-Parameter, [character-c filepointer-stream])
VALUES(has-synonym, ["fputc"]).

INSTANCE fgets = AND(Functions
VALUES(reads, [string-s])
VALUES(has-Parameter, [string-s filepointer-stream])
VALUES(has-synonym, ["fgets"]).

INSTANCE fputs = AND(Functions
VALUES(writes, [string-s])
VALUES(has-Parameter, [string-s filepointer-stream])
VALUES(has-synonym, ["fputs"]).

Note that the concepts actions and Call-Actions and their accompanying roles are not really used in this example; they are present for explanation purposes only.

The hierarchy that results from this example is depicted in figure 2. The roles calls and with are not shown in this picture, because they are not actually used in the example anyway.

Figure 2: Taxonomy of the example knowledge base

5.1 Information about Individual Words

The simplest form of annotation to a concept or role $X$ is the synonym list: The VERB and NOUN
lists give a list of verbs and non-compound nouns, respectively, that can denote the concept or
role. The same word can annotate multiple concepts or roles, resulting in ambiguities.
for that word. For instance, the German words *Funktion* (function) and *Unterprogramm* (subprogram) both refer to the concept *Functions*. Similarly, *lesen* (to read) and *einlesen* (to read in) refer to the role *reads*. *Einlesen* 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 specialization are all nouns that annotate any superconcept of X. The *ADJECTIVE* annotation shows one of the ways inheritance is used in the construction of case frames: wherever an annotation to X specifies a part P of a natural language construct to be *added* to other construct A, then A is inherited from the superconcepts or superroles of X. For instance, *die Funktion* (reading function) refers to *Input-Functions*, where *Funktion* is inherited from the annotation of the direct superconcept *Functions*. The word *lesen* is a present participle, but can be used as an adjective in our system. The nouns that can be used need not be annotated at direct superconcepts: *lesen des* Objekt (reading object) could be used as well to denote *Input-Functions*. In a real knowledge base this phrase would most probably be ambiguous, because *lesen* might annotate other superconcepts of *Objects* as well, but ambiguity a good result in this case, because the phrase is indeed very vague.

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 prefix nouns that can be prepended to the nouns annotated with the superconcepts of a reference to X. This annotation is analogous to the *ADJECTIVE* annotation. For *Funktion* (read function) refers to *Input-Function*. Only the words *lesen* and *Funktion* in the dictionary, the compound is algorithmically broken into these components by the adjectives, inheritance is possible from concepts that are more than one level above.

A adjective or noun prefix to specialize a noun, it is often possible to use a pre- that is placed right behind the noun. This possibility can be expressed with the (*PREFIX*) annotation to X: It gives a case (with a filler that, when used together to a superconcept of X, denotes X. This annotation is an extension because it produces a case that does not get inserted into a case filler or head instead. For instance *Funktion zum lesen* (function for *functioning*). The possible grammatical form for this reference are listed in the case appendix A.2; its relevant entry for this phrase is "preposition zum with". This type of annotation need not be used, because the same handling capability can be getting an additional role into the model: *zweck* means purpose and could have been "as-purpose" that has *Functions* in its domain. But a model may be somewhat arts; then the *SYN-CASE* annotation is a simple way to increase the coverage of surface.

*Cases* are phrases that represent pure concepts or roles; no case frames or the case frames are the roles. Information about cases is to insert where.

*MAIN(sc)* at a role R creates a case in the case frame this case is the range concept of R; it has to appear
in a grammatical form described by the role table entry \( s c \) (see Appendix A.2). The dual form of this annotation is AT-RANGE(\( sc \)). Given at a role \( R \), the AT-RANGE annotation creates a case in the case frame of the range concept of \( R \). The allowed filler for this case is the domain concept of \( R \); it has to appear in a grammatical form described by the role table entry \( s c \). Examples can be found in Section 6.1. Note that the cases are inherited by subconcepts of the concepts they originally target at.

Not all cases in all case frames are created from such annotations. Some cases can be added without annotations and some case frames can be built completely automatically. These details are explained in 6.1 below.

### Information about Explicit Inheritance of Wids

In a hierarchy in WiS, there are no superroles for a role and it is necessary to have ADJECTIVE annotations, because they rely on inheritance. For concepts, because, since a derived concept \( D \) is described by \( s \) (which includes \( s \)), no concept can ever be guaranteed to be a superconcept of \( D \). To overcome this problem there is the SUPERR annotation to concepts. To annotate SUPERR(S) at a concept \( c \) from \( S \), although this form of annotation may seem at first annotation of complete phrases, because it is the annotations work as possible. SUPERR annotated concepts work as possible. SUPERR annotations are used across multiple levels.

In our system, all Wids are registered as their...
domain. The theme roles have verb synonyms. All these conditions are necessary and sufficient (otherwise the modeling is incorrect). The has-action-or-object roles have actions or objects as their range.

For each of these categories there are fixed rules that describe which cases and case frames must be generated; the case frame generator module implements these rules. A case frame consists of a head and a set of cases. Here are two examples:

Call-Actions  [C-Call-Actions]  
  agent has-agent (C-Functions),
  theme calls (C-Functions),
  attribut with (C-Parameters),
  benennung has-synonym (C-Call-Actions))

has-Parameter_HADO [R-has-Parameter]  
  agent DR (C-Functions) 21,
  theme RR (R-has-Parameter) 21)

Call-Actions and has-Parameter_HADO are the name of the first and second case frame, respectively. Call-Actions/has-Parameter is the head of the first/second case frame (marked to be a concept/role). Each of the indented lines is one case. The components of a case are, in the order shown, the syntactical role (i.e., the name of an entry in SARN's role table; see appendix A.2), the expression to be used to generate that part of the YAKS query from the instantiated case frame is to this case (if it is filled), the list of allowed fillers for this case (which must often one element), and optionally the priority mark, which is 20 (and not shown then) by 21 marks a case as mandatory, i.e. it must appear in an instantiation or else that case will be dropped. See appendix B for a complete listing of the case frames that are generated.

If the relevant concept and role categories what cases and case frames are generated:

concepts

Every action-or-object concept AO. It contains at least one case of

(C-AO)

in noun phrases such as die Funktion "f", where it catches the to AT-DOMAIN/AT-RANGE annotations: All roles R with an action-or-object concept as their domain and a concept rmg

E(sc) annotation that have an action-or-object concept

generate a case of the form

frame.
6.1 Generation of Case Frames

annotation have an attribute concept as their range. Thus there are usually no cases in the case frame of an attribute concept except the one to catch a name.

6.1.3 Has-Action-or-Object Rules

For each has-action-or-object role three case frames are generated. These are named after the role with additional suffixes _HAC_, _HAC_, and _HAD_. As an example, assume the domain Functions of the role has-Parameter has been annotated with the noun Funktion, the range Data-Objects with Datenobjekt, and the role itself with Parameter, then

- the has-Parameter _HAC_ case frame serves to parse nominal phrases that mention the role itself as a relation with the Parameter "X" von Funktion "F",
- has-Parameter _HAC_ parses to be phrases of the kind der Parameter ist das Datenobjekt "R" (using has-Parameter _HAC_ to catch the Parameter), and
- has-Parameter _HAD_ parses to have phrases of the kind die Funktion "F" hat den Parameter "P" (using has-Parameter _HAD_ to catch the Parameter "P"), which is the most natural of the relation described by the role itself.

Of course the actual inputs will usually not be declarative sentences. The _R-HAO_ case role _R_ with domain _do m_ and range _rng_ uses the role itself as its head and has the two cases

- benennung has-synonym (C-rng)
- gen von INV(R) (C-do m)

The _R-HAO_ case frame of a role _R_ with range _rng_ uses the word sein as its head at has the two cases

- agent RR (R-R) (21)
- definition RR (C-rng) (21)

Were the allowed filler of the agent case means that the _R-HAO_ case frame of the sentence be used and the syntactic role definition stands for the grammatical case "nominative" are mandatory (i.e. must be filled for an instantiation to be legal). The _R-HAO_ case frame _R_ with domain _do m_ uses the word haben as its head at parsing time and has the two cases

- agent DR (C-do m)
- thema RR (R-R)

Were the allowed filler of the theme case means that the _R-HAO_ case frame of the same role must be used and the syntactic role thema stands for the grammatical case "accusative". DR stands for "domain restriction" and RR for "range restriction"; the instantiated case frame is transformed into a role expression by the query generator.

6.1.4 Has-Attribute Rules

has-attribute roles are handled much as has-action-or-object roles with the following differences: (a) the suffixes of the case frame names are _HAA_, _HAC_, and _HAD_ and (b) since concrete specifiers usually appear as adjectives, the definition case in the _HAC_ case frame is not always sufficient to catch the range of the role; it is therefore complemented by another case with the syntactic role adj adv (adjective or adverb) and exactly one of these two cases must be filled by an input sentence.

\footnote{Once upon a time, a _HAP_ case frame existed, too, but it has been merged into the _HAC_ case frame now}
6.1.5 Theme Roles

Theme roles generate one simple case frame that contains exactly two cases: one for the domain $D$ of the role and one for its range $R$. These cases have always the same syntactical roles and YASAT's associated with them; they look like the following:

\[
\begin{align*}
& \text{Dr} & & (C-D) \\
& \text{Ra} & & (C-R)
\end{align*}
\]

Dr stands for “domain restriction” and Ra for “range restriction”.

Synonymless Roles

Asume says, synonymless roles have no word annotations. Thus it is not possible for a case frame to be activated by a certain word in an input sentence. Consequently there are synonymless roles, but a synonymless role always has an AT-DOMAIN annotation to the frame of an action concept.

The instantiated case frames generated from an input sentence by the translation, as well as all of its fillers are considered and combined into a query expression — in many cases the computation of the restrictions is recursive; the elements of the case frame instantiation that pertain to roles and role queries. First of all it must be any given case frame. Role queries can be in the answer. We thus try to get an idea about which pairs might be situations that have sein or haben already annotated at a role, following:

"Relation of JIAO to and these are filled or the ran a synonymof a role, because a case filled by"
5. Any other case with WKS expression \( expr \) and filler \( X \) is translated into
\[
\text{SOME}(expr, X)
\]

Somewhat different handling is necessary for relative clauses and for the construction of appropriate role queries for W questions. This handling is sketched in the following paragraphs.

The restriction that is defined by a relative clause has to be put into the restriction that is returned for the noun to which the relative pronoun refers. This is straightforward if the relative clause maps into a concept expression. But if it would normally map into a role expression (because the head of instantiation for the relative clause is a role \( R \)), it has to be converted into a equivalent concept expression. It is possible to do so, because we know whether the relative pronoun filled (a) a case with \( DR \) or (b) one marked with \( RR \): let the translation of the filler of the other case in the case be \( F \), then return \( R(F) \) for (a), or \( (\text{INV}(R), F) \) for (b).

If the restriction is necessary to defer the generation of the query terms that correspond to the rest of the query is known. Therefore propagate markers for the rest of the query towards the uppermost level of the recursive process. Then we can then propagate the role query from \( R(F) \) and the rest of the query restrictions as \( RR(F) \)

It is also possible to build correct queries from explicit questions for pairs, between the query exists in the modeling by composing the two roles that the question asks for.

### Limitations

An efficiently big database the usefulness of WKR for an end-user is limited in some, because the syntactic and semantic limitations of natural language interface are not high enough in the first attempt, it can be difficult for the thing that the user specifically learn the restrictions of conditions are not by our.
sophisticated in this respect, 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.

2 Practicality

Practicality of the implementation of a system such as YAR is good: The software is of moderate size and could be produced by a small team in some months. It took about 4 person years (including implementation) to produce the system. Even our prototype is neither large nor difficult to use.

Database maintenance is difficult to predict in general. If additions to the database are made, it is likely that they do not require changes in the terminology; they are likely to be simple additions. For any other database with a non-trivial structure, additions will be added that require new terminology, some knowledge of the domain, and coherence. Another problem is the maintainability of the database: it contains adequate descriptions of software.

YAR would be high as far as the maintainability of database practicality is concerned, to use the
some human-readable documentation for them. The repository we target consists of components for which no formal input/output specification is available and which do not necessarily use common data structures, common processing models, or common modularization strategies. Thus the informality of the availability of the components shows up in our system in the informality of the interface we use.

and, reuse could also take place in a more controlled and formalized environment, and cataloging) of reusable components and the production of new software by a common formal framework. In this case other methods to access might be superior, namely those that use the information that is

Even in this case, however, YAS may be a good tool to should be extended.

what is usually understood as software reuse;

process of changing a software system too.

One could of course be considered software

There is of section

SS.
The software information system that is most similar to ours is LaSSIE/COB BASE [14, 32]. LaSSIE initially used a frame-based knowledge representation language called KANOR, which was later replaced by a language called Classic. In LaSSIE all information in the knowledge base had to be represented using Classic. In COB BASE, a lot of information is acquired by an automatic form-filling and is then stored in a database which is queried on demand. The user interface is a natural language parser plus a graphical browser for navigating.

Significantly, some powerful constructs are missing; for example, two roles, the inversion of roles, union of concepts or descriptions, negation of concepts or roles, and the like, are avoided in the whole system. COB BASE is intended to be LaSSIE enhanced.
 syntactic coverage; the other knowledge sources have to be updated for a new database. The lexicon contains word information for morphological, syntactic, and semantic processing. The conceptual schema consists of sorts information and constraints on the arguments of non-sorted predicates. Finally, the database schema consists of information that enables the mapping of the intermediate representation to a query expressed in a relational query language.

The acquisition process differs from the one we use. Our approach relies on specifying lexical and knowledge in the lexicon and annotated knowledge structures. In TEAM lexical and knowledge structures and the answers of the user. Verbal case frames are given by the knowledge engineer and questions about correctness. This kind of acquisition is motivated by the aim that non-unique interface to a new database. Similar acquisition tasks are currently more difficult. However, building such tasks is deduced when using NLP, for example.

This requires less syntactical and semantical information.
4. The practical efficiency of the deductions varies much with a large knowledge base. Many queries
return within less than a second, some others take minutes.

5. It is possible to build a natural language interface for a specific knowledge base with only minimal
additional work (less than 10 percent) for the knowledge engineer.

A natural language interface to a repository of software components is useful to have, even if it
tactically restricted.

Further Work

There are a lot of possibilities to improve our system. The most important ones would be
abilities of the natural language interface (syntactic and semantic), to complement
face with menu and windowing techniques (for instance to access the source
to speed up those deductions that are now very costly, and to avoid
to virtual memory. We do not currently plan to follow any of

instance: Is this semantic modeling plus natural language

ter than, for instance, much cheaper information
on documentation files of the components?
Part of the program described in chapter 1
our system will then be compared.
We expect that the result of
es” and “unsuccessful
to “near misses”.
idea where our
A Other knowledge sources of YAKR

Apart from the YAKR specification of the knowledge base, there are two other sources of knowledge needed by the parser: the dictionary of word forms and the table of syntactic role formats and semantics are described in the following two subsections.

Dictionary

All the words that SARA shall be able to recognize. It is implemented partly with an algorithmic word form analysis. The information it delivers (part-of-speech label, time, casus, numerus, genus, person, for most types of words, dictionary information rarely needs for verbs, nouns, and adjectives additions are already contained about 10000 words with about 25000 implicitly recognizable word forms. The format definitions are by example for ease of understanding.

Sara.uberben, their form can be deduced from and below:

its wb v and look like

:ung } utf) :um } prefix () :um }

n ab aus vor) :ung }
y and must be the infinitive form of the

ma-separated prefix word parts is sample ackern and beackern).
back but beackern --beackert
remaining “do not prepend ge even where other prefixes

of separable prefixes is example fertigen
case prefix is cut
uch abfertige etwas.
cept if the empty fertigen --gefertigt
d list eliminates the ge of aufgeaddert.
explicitly generated in the

analysis.
The last part of a verb entry is either :m (which is an abbreviation for the also possible
regelmaessige) or :ung. :m designates the verb as a regular one. :ung does the same, but ad-
verbially results in the generation of another nominalization. For all verbs (whether regular or not),
infinitive forms are automatically put into the dictionary as a noun, too (e.g. ackern — das Ackern).
Verbs second noun entry is made, in which the en ending of the infinitive form is replaced
fertigen — das Fertigen, die Fertigung.

Partizip I and Partizip II forms of all verbs are automatically also available as adjectives and as

Nouns

For nouns may look like

litt :substantiv :typ (Ss, Pe) }
   :sub :typ (S, Pn) }
   :sub :geschlecht (s) :typ (Ss, Per) }
thms :sub :geschlecht (m) :typ (S) }
thms :sub :stamm algorithmen :geschlecht (m) :typ (P) }
   :sub :geschlecht (s) :typ (Ss, P) }}
satz :sub :stamm zeichensatz :typ (Ss, PUs) }

The first part of the entry is the word name, which must be the base form of the noun. :sub (or
substantiv) is the key word that assigns the part-of-speech.

precedes the list of inflectional types of the noun. The available types are S, Ss,
Pn, Per, Pe, Pss, Pi, Pue, PU, PUs, PUsen, PUser. The S-types describe how
is formed from the base form either by appending nothing (S), as in die
by appending s or es (Ss), as in das Bild — des Bild(e)s, or by appending
Mensch — des Menschen. From this S-type assignment all singular noun forms
into the dictionary (nominative, genitive, dative, and accusative case).

The nominative plural is formed from the base form
appending nothing/e/n/en/er/s,
Putting the first or @-marked vowel into the corresponding unaccented plus

Inse,
for us into, and

mimoen.

automatically assumed to be female, all others are assumed to be
for about 80 percent of all nouns. For the rest, gender
geschlecht follows by a parenthesized list (!) of the
for neutral. Multiple genders can be assigned to a

Singular different stems for singular and plural form.

withus above.
A 1.3 Adjectives

Entries for adjectives may look like

```
{ absolut  : adj }  
{ bedeuten  : adj : steigerungsstamme (bedeutend, -)  
  (bedeutenst, bedeutenst) }  
{ public   : adj : ungebeugt }  
```

The first part of the entry is the word name, which must be the base form of the adjective. :adj (or alternatively :adjektiv) is the key word that assigns the part-of-speech. If the comparison endings are not er, est, the complete base form for positive, comparative, and superlative can be given as a list of words (or word lists for alternative form) after the keyword :steigerungsstamme. It is also possible to specify that the word should be considered to be an adjective in an input sentence even if it appears without an inflectional ending by giving the keyword :ungebeugt in the entry last. This is needed to handle German usage of English adjectives.

Adjectives are analyzed algorithmically and are not put into the dictionary as full forms. All adjectives are automatically also available as adverbs.

Role table

Associates the names of syntactic roles with a set of grammatical constructions and a set that can be used to refer to this syntactic role. The standard set of syntactic roles in the SARA package has 33 entries. Although it does not need to be changed, a role table entry is the following:

```
;'ich' schlage keinen Hund  
;'von mir' wird kein Hund geschlagen  
;'von Nachbarn' werden alle Hunde geschlagen  
;'durch mich' werden keine Hunde geschlagen  
;das Klagen 'des Nachbarn'  
;das Klagen 'von mir'  
;das Klagen 'von Nachbarn'  
;das Klagen 'durch den Nachbarn'  

;'wer' fragt mich  
;'was' krabbelt meinen Rucken hinauf  

;'von wem' werde ich gefragt  
)
)

(sinnvoll, wenn die Agenten SW-Objekte sind.)
This entry can be read as follows: \textit{agent} is the name of the entry (as to be used in \textsc{AT-DOMAIN} and \textsc{RANGE} annotations). The following string is merely a free form description of the entry. All of the following is optional, except the keyword \texttt{fragen}.

\textbf{aktiv} means “the following appearance form are valid for verbal phrases (i.e. clauses) in active voice only”. \textbf{nominativ} means “one possible appearance of the \textit{agent} role is a noun in pure nominative case (i.e. without a preposition)”. The part of the line after the semicolon is a comment and gives an example explanation for the other appearance entries are analogous.

\textbf{nur aktiv} keyword. This had meant that they should be used for purely syntactic reasons, to ease parsing the meaning of a sentence indicates that the sentence may be given and must all appear in exactly one of the cases of the entry.

The listing
adj_adv ?? (C-anything) (21),
agent ?? (C-anything) (21))
Data-Objects [C-Data-Objects] (  
  benennung has-synonym (C-Data-Objects),
gen_von INV(has-Parameter) (C-Functions))
has-Parameter_HA0d [R-has-Parameter] (  
  agent DR (C-Functions) (21),
  thema RR (R-has-Parameter) (21))
has-Parameter_HA0c [R-has-Parameter] (  
  agent RR (R-has-Parameter) (21),
  definition RR (C-Data-Objects) (21))
has-Parameter_HA0a [R-has-Parameter] (rollensynonym) (  
  benennung no-KRS-role (C-Data-Objects),
gen_von INV(has-Parameter) (C-Functions))
Files [C-Files] (  
  gen_von INV(has-Parameter) (C-Functions),
  benennung has-synonym (C-Files))
sein1 [W-sein] (  
  definition ?? (C-anything) (21),
  agent ?? (C-anything) (21))
Objects [C-Objects] (  
  benennung has-synonym (C-Objects))
GF [C-anything] (  
  definition no-KRS-role (C-anything) (21),
  agent no-KRS-role (C-anything) (21))
Input-Functions [C-Input-Functions] (  
  benennung has-synonym (C-Input-Functions))
Output-Functions [C-Output-Functions] (  
  benennung has-synonym (C-Output-Functions))
haben1 [W-haben] (  
  thema ?? (C-anything) (21),
  agent ?? (C-anything) (21))
writes [R-writes] (  
  agent DR (C-Output-Functions),
  thema RR (C-Data-Objects))
Lines [C-Lines] (  
  gen_von INV(has-Parameter) (C-Functions),
  benennung has-synonym (C-Lines))
actions [C-actions] (  
  benennung has-synonym (C-actions),
  agent has-agent (C-Functions))
Characters [C-Characters] (  
  gen_von INV(has-Parameter) (C-Functions),
  benennung has-synonym (C-Characters))

Fallschablonen zu ’sein’: (GF, has-Parameter_HA0c)
Fallschablonen zu ’haben’: (has-Parameter_HA0d)
sein1, sein2, and haben1 are the case frames that are used internally to parse all inputs with sein or haben as main verb. Instantiations of these case frames are then converted into instantiations of the appropriate JPA, JPO, JPC, and JPH case frames by a unification algorithm. This method is the extremely long running time of the parser that would result if all sein/haben cases were used for parsing. GF is the so called general frame that is used to parse inputs of the A sind B where both A and B are object concepts.

**Interface commands**

A command interpreter looks like this:

```
Sitzung
auf
sbasis (kann #krsincluden enthalten)
Wissensbasis
neutrafahren/Konzepte
ischablonen
sbasis auf Konsistenz
ischablonen
in (ein/aus)
berungen anzeigen (ein/aus)
bedateinamen bei w,k,r,f,t (ein/aus)
em
berungen auf EDGE-Datei schreiben (ein/aus)
berungen einmal auf EDGE-Datei schreiben
rdeingabe um
basis nach less
en Eingabesatz
urpretierer aufrufen (Verlassen mit 'quit')
ieren Woerterbuches aus `/tmp/sara.wb.bin.Z`
```

Described now

are files as a SARA knowledge file, which may contain:
1. A dictionary of other SARA files, and
2. #krsincluden of WWS files.
3. A knowledge base file.
4. Corresponding dictionary entries, concepts, and base. The output should be more or less
   either R-, C- or I- to display that
   WWS. Bible concepts are further
   is a synonym, then, has-
   concepts may be further prefixed
   attribute, or other concept.
   ents whose names contain
   ly empty (i.e. just a
"<" reads the binary form of the dictionary from a file whose name is given in the file .sararc; this is very much faster than to parse the source form of the dictionary, but is only possible as long as the dictionary is completely empty. This is usually the very first command issued in a session.

">" writes the complete dictionary onto a file whose name is also given in the file .sararc, for later use with the command <. To avoid accidents, it is clever not to have the same name in .sararc for input file as for the input file.

Example Session

tcontains a short example session with YAKR, using the example knowledge base from above. This typeface while user input is in this typeface. Comments are indented.

YAKR                     ? fuer Hilfe

in.Z : sis.wb................

the SARA dictionary has been read in.

basis ein.
A-Wissensbasis: bsp

The contents of the file bsp are
like "sara.std"
include "daten/bsp_yaks"

a lot of output is generated that shows the names of the objects in the as they are created while the knowledge base file is being read. This is here.

announces that all knowledge files have been read in and generates output not shown here is generated that shows the names created.

ausgehen

angeben et was aus

string-s
character-c

neue Funktion ein Zeichen einlesen ?
etc reads character-c

SARA: Ausgabe funktionen

fputc  fputs

SARA: Funktionen, die ausgeben

fputc  fputs

SARA: i

Instanzierungen anzeigen wird EINGeschaltet
This command toggles the displaying of the instantiated case frames and the YAS queries generated from them.

ionen, die ausgeben

ionen, die ausgeben" ->

G4 C-Functions,
   [Gw Fngda Np P3 substantiv funktion]
L2 G24 R-writes, Relativsatz Np, P3, (aktiv,praesens,indikativ,Nebensatz)
rolle L2 G2 agent/DR
   [:I L2 G2 C-Functions,
      [Gw Fna Np P3 substantiv funktion]]]]]]

EPT ?cvar = AND(Functions
ME(writes)). /* Concept */

ifputs

q

ss !

Possible Problems with Natural Language Interfaces

Real dialog may serve to illustrate some of the difficulties that may arise in the use of natural language interfaces. Actually, this projection was made in the use of a slow printing terminal. Please take care to understand the initial steps. To provide the best possible service to You, my computer has the capability to converse in basic English as well as the ability to understand the keyboard.

Steps must start with //
your type in, type *** and press the CARRIAGE RETURN key.
you type out, depress the MATCH RELEASE key.
your type in, backspace and type over the erroneous characters.
your type in, type ! or ?.
your type in, type " finished! ".
will be billed for the time used at the end of the month.
What is your credit card number ?
  //372 AC-2631 AFX. !
What is your name ?
  //J. Wm.ley Rogers III.
!
Credit card number validated.
What may I serve you, Mr. Rogers?

//Do you provide library reference services?
Yes. I have access to \(1.27 \times 10^7\) volumes of books and \(3.26 \times 10^7\) publishers of magazines.

//1.27*10E+7?
"1.27*10E+7?" meaning unclear. Please clarify.

//Sorry. What does 1.27*10E+7 mean?
1.27*10E+7 is number. Other legal notations are 12700000 decimal or 11000011010100000000000 binary.

//Oh, I see. Can you give me all your references on Tarot cards?

Tarot cards?

//Tarot cards!
Tarot cards?

//Cards used by gypsies to foretell the future!
I have major subject indices for subjects "card", "gypsy", "gypsy", and "future" displayed?

//I want to know about Tarot cards.
!

//!

Tarot cards?

//OK! You win. I va
"OK" meaning unclear. Please clarify.

//Listen you goddam***INPUT CANCELLED***
//Please display your index for subject "Gypsy".

//!

Books or magazines?

//Both!

Which first?

//Books?

"Books?" meaning unclear. Please clarify.

//Books!

Do you wish the listings in random order, alphabetic order by grouped by publisher, or ordered by date of publication?

//Whatever.

//!

"Whatever." meaning unclear. Please clarify.

//Present the data in whatever order you
wish the listings in randomorder, alphabetic order by title, alphab***TYPEOUT INTERRUPTED***

/By date of publication!
Earliest or latest first?
///Latest!
First or last?
///Either***INPUT CANCELLED***
///First!
?
!
///!

Yes.
///Yes?
"Yes?" meaning unclear. Please clarify.
///"Yes" what?
Yes, I can give you all my references on Tarot cards.
///Why the Hel***INPUT CANCELLED***
///Why didn't you say so in the first place?
Why didn't I say what?
///Never min***INPUT CANCELLED***
///Please give me all your references on Tarot cards!
Do you wish the listings in randomorder, alphabetic order by ti***TYPEOUT INTERRUPTED***

///RANDOM ORDER
///!
Several references are listed as classified/restricted information. If you wish access to them enter access code.
///What's classified about Tarot cards?
Tarot cards?
///Tarot cards
///***INPUT CANCELLED***
///Quit the classified references.
///!
Several references are listed as restricted information. If you wish acce**
Quit the restricted information.
!
///!
No input received. Please enter next request.
///Quit the restricted information!
(From DAINAIL ON April 1973, pp 72-73, by Ronald Kenney)

All enjoyment, no work again and read carefully.

References

16. Charles J. Fillmore. The case for case. In [3], Holt, Rinehart


