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 re
tory. The approach is (1) to provide a knowledge representation system that a
the components in the repository with user-defined semantic cat
capabilities in this knowledge representation sys
(3) to complement the formal qu
achieve ease of u
knowl

Contents

1 Introduction 4
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 Roles . . . . .
6.1.4 Has-Attribute Roles . . . . .
6.1.5 Theme Roles . . . . .
6.1.6 Synonymless Roles . . . . .
6.2 Generation of Queries . . . . .

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

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

9 Results

10 Further Work

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

B List of Generated Case Frames

C User interface commands

D Example Session

E Possible Problems 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
able component for a given problem, (b) how to adapt it for the problem, if necessary, and (c) how
y use it correctly. The SIS project attempted to build a system that mainly addresses (a),
nts are only partly covered. This system is called YAKR.

subsections describe the assumptions YAKR is based on, the requirements that follow
ions and how these requirements are shaped into a concrete system design.

ssumptions

ons that underly the design of YAKR:

oking for a reusable software component for a certain task, often no complete
unctionality 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

tware 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.

oservations of many scientists in the area of software
]), although especially A4 is often not addressed at
systems.

owing requirements as guidelines for the design

ial and inaccurate specifications.

the same request.

onents must be easy to use.

S project, we picked one of them
utlined in the following

1.3 Basic Design

R1 is realized with a specially designed knowledge representation language, YAKS (Yet Another Knowledge representation System[1]), 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
omplete as well as inaccurate queries to be answered: they just retrieve elements that are more
in respect to the taxonomy.

isfied by a natural language interface. Natural language is the most versatile way of expression.
tural language interfaces have the disadvantage that they are expensive to construct and
a new domain. We have found a way to minimize the work that is needed to construct
e: only short annotations to each definition in the terminology are needed.

ne natural interface, too. Since natural language is the easiest way for a
on, especially if its conception is still fuzzy, the tendency not to use the
zed.

maintenance problems in our system. First, it must be easy to add
ts. This ideal is approached by letting the knowledge represen-
ogical part and an assertional part. The terminological part
d relations of a domain, but does not state any specific
describes the actual objects in the repository in terms
l case when adding new software components is
terms of the terminology that has once
the domain that has not yet been
s is complicated. Thus,
e base is easy; the
or a repository.

ts. This
nly a

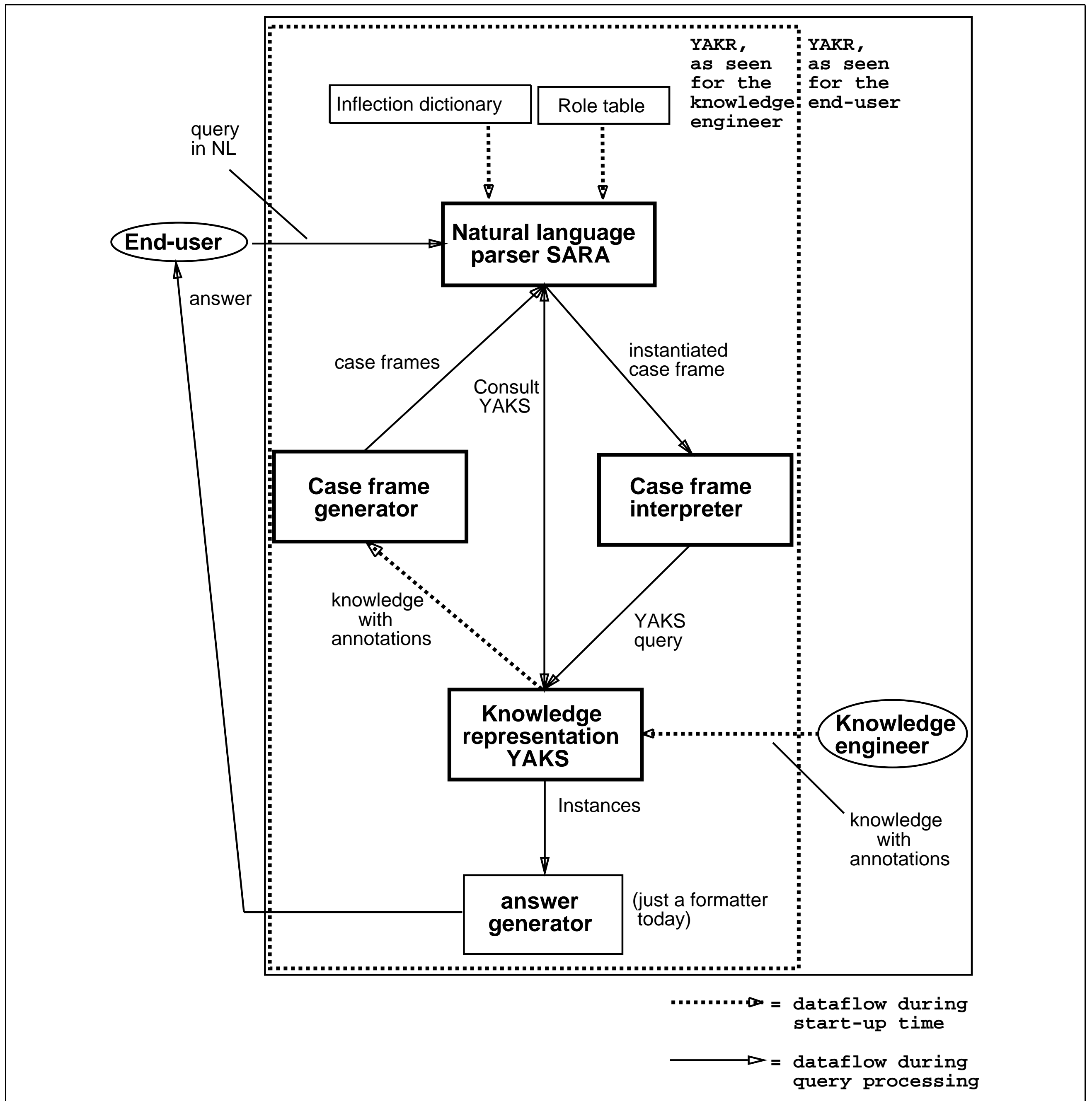

Figure 1: Basic architecture and dataflow of YAKR

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 indivi-
duals. Each constructor thus represents a restriction that an instance must adhere to in order to
ong to the concept that is defined by the constructor. Roles 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
).

nguage in YAKS is quite expressive, allowing value restriction, number restriction for
-value maps, conjunction, disjunction, and negation. The concepts form a *hierarchy*
ple inheritance. The role language allows conjunction, disjunction, domain
ction, negation, and inversion.

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

```
ons = AND(Functions  SOME(reads)).
a-Objects;).

s = Objects.
```

owing:

d concept, i.e. exactly all those instances that obey the restrictions
ion belong to the concept.

hat 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

ose instances that obey the restrictions given
but we can not know whether they really

bjects (whose definition is not shown

roles. The most important
determines whether a
ld descriptions all
ct placement of a
perconcepts
ncepts

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 realisation of all instances for the larger example on page 11 is shown in figure 2 on page 13. Since the YAKS 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
age 11:

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

definition means the following:

getc is an instance (individual).

fgetc belongs to the concept *Functions*.

etc has the instance *character-c* (whose definition is not shown, but which belongs to the
: *Data-Objects*) as filler of the role *reads*.

s the instance *filepointer-stream* (whose definition is not shown, but which belongs to
ata-Objects) as filler of the role *has-Parameter*.

ring instance *"fgetc"* (which need not be defined, because it is a string) as filler
ynonym

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

leling (describing concepts and roles) and querying, i.e.,
ncept or role can also be used to query for one. Thus,
S more expressive than a relational database.
age interface, because the query language
be interpreted and how they can be
om limitations of the modeling

stages: First, define the
nd, create the actual
nology that has

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 YAKS 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.

A tiny example of what a YAKS modelling 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 NIH 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 Kilobytes of
knowledge base for NIHCL models only the top three classes of the
(and the rest very roughly), but nevertheless contains almost
ncepts in 105 Kilobytes of YAKS source code.

odel, although it must be mentioned that YAKS was
as well as changing. We also first had to learn
ptions and design flaws turned out to be

t part of the instance descriptions
eclarations and cross references.
on C++ programs. It generates
of C++. Only the purpose
actable from the source

header files (140
entation
feed *all*
ation

rotected, declares-private,
defines-protected, defines-public, ends-in-line, ex-
, has-base-file, has-basetype, has-cast, has-class-type, has-constructor,
e, has-datatype, has-default, has-derived-class, has-descendant-class, has-destructor, has-
dimension, has-directory, has-enumerator, has-friend, has-function-call, has-implementation, has-initial
has-inner-block, has-length, has-linkage, has-linkagetype, has-local-variables, h
name, has-number, has-outer-block, has-owner, has-parameter, ha
protected-member, has-public-base-class, ha

has-specification, has-specifier, has-subclass, has-superclass, has-synonym, has-virtual-base-class
cludes, inherits, is-datatype-of, is-declared-in, is-defined-in, is-enumera
is-friend-of, is-included-by, is-inherited-by, is
class-of, is-private-

f
c
cannot

However,
broad compared
Noun phrases with
simple conjunctions and
questions, declaratives, im
dal verbs, immediate relati
ses starting with a conjuncti
general numbers, general q

Case frame parsers convert writ
tion; no surface structure is gener
frames, representing semantic knowle
represents an utterance by its central co
each of which describes (a) a certain sema
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
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. Inste
of YAKS to build a corresponding case frame hierarchy (there are also case
which cases are implicitly inherited. A similar technique is used for the fil
words, concepts are stated as allowed fillers. With any concept all of its superc
are legal fillers, too. For each of these concepts, a whole set of words or phrases c
language representations. Case frames can be nested when parsing: If, for example, t
in a case frame is a noun that has a case frame associated with it, a complete instantiati
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 separat
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 representat
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 require too much work to adapt them to a new domain. In the design of YAKR, we thus paid special attention to the problem of knowledge acquisition for the natural language interface: Ideally, nothing more should be necessary o acquire than the words which can be used to refer to each concept, role or instance. If this ideal not approximated closely enough, it is necessary to specify complex grammatical descriptions; A4 s that in this case the natural language interface will not be successful in practice.

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

associates each concept or role with its natural language synonyms and with nt this concept or role. The variety of the phrases covered by the annota- increased by using an inheritance mechanism to derive parts of these erconcepts or explicitly stated syntactical superroles of X.

the generation of the case frames themselves. It describes which ames. This annotation is needed for roles only. Nothing more given, since the set of case frames to put the case into rtion can be deduced from the YAKS modeling.

sed for roles and for derived concepts in order hrases for these concepts or roles, although erconcept) from which the nouns could be

he following example. It should be show how YAKR works. It does y 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
```

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

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 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.

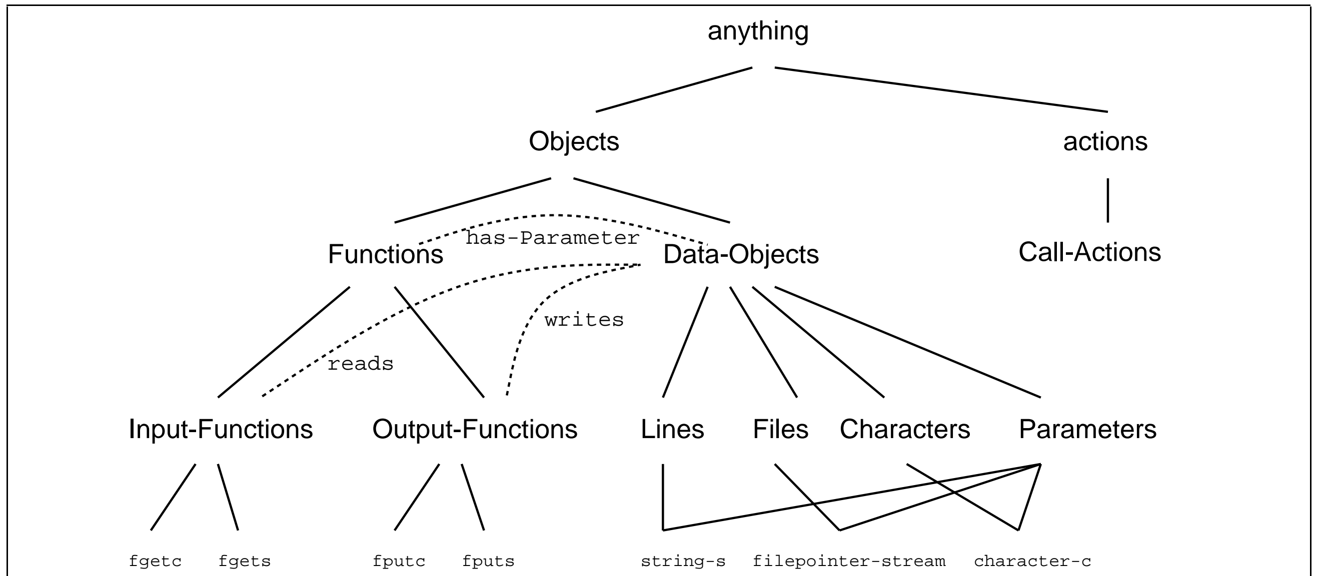

Figure 2: Taxonomy of the example knowledge base

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 annotations give a list of verbs and non-compound nouns, respectively, that can denote the concept or role X they annotate. 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) refe *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 this prefixing: It lists a number of adjectives that can be used to specialize a noun in order X. The suitable nouns for this specialization are all nouns that annotate any superconcept of X ADJECTIVE annotation shows one of the ways inheritance is used in the construction of case frame 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, lesende Funktion (reading function) refers to *Input-Functions*, where Funktion is inher annotation of the direct superconcept *Functions*. The word lesend is a present partic be used as an adjective in our system. The nouns that can be used need not be annotated at superconcepts: lesendes Objekt (reading object) could be used as well to denote *Inp* real knowledge base this phrase would most probably be ambiguous, because lesend n other subconcepts of *Objects* as well, but ambiguity a good result in this case, because the indeed very vague.

Compound nouns are written as a single word in German, so they could all be put into the dictionar 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 is analogous to the ADJECTIVE annotation. For instance, Lesefunktion (read function) refers to *Input-Function*. Only the words Lesen[2] and Funktion need to be in the dictionary, the compound is algorithmically broken into these components by the parser. As with adjectives, inheritance is possible from concepts that are more than one level above.

Instead of using an adjective or noun prefix to specialize a noun, it is often possible to use a prepositional phrase that is placed right behind the noun. This possibility can be expressed with the SYN-CASE (synonym case) annotation to X: It gives a case (with a filler) that, when used together with a noun that is annotated to a superconcept of X, denotes X. This annotation is an extension of the usual case frame formalism, because it produces a case that does not get inserted into a case rame instantiation, but modifies a filler or head instead. For instance Funktion zum Lesen (function for reading) refers to *Input-Functions*. The possible grammatical forms for this reference are listed in the case table entry *zueck* (see appendix A.2); its relevant entry for this phrase is *"preposition* zum *with dative case"*. This type of annotation need not be used, because the same handling capability can be achieved by integrating an additional role into the model: zweck means purpose and could have been modeled with a role *has-purpose* that has *Functions* in its domain. But a model may be somewhat in some parts; then the SYN-CASE annotation is a simple way to increase the coverage of nguage interface.

mation about Cases

e annotations merely describe phrases that represent pure concepts or roles; no case frames hem. The source of cases for the case frames are the roles. Information about cases is only and tells what cases to insert where.

nt case annotations: AT-DOMAIN(sc) at a role R creates a case in the case frame R. The allowed filler for this case is the range concept of R; it has to appear

t to have the *verb* lesen in the dictionary, because its nominalization is automatically

in a grammatical form described by the role table entry *sc* (see appendix A.2). The dual f
annotation is AT-RANGE(sc). Given at a role R, the AT-RANGE annotation creates a case i
frame of the range concept of R. The allowed filler for this case is the domain concept of R; i
appear in a grammatical form described by the role table entry *sc*. Examples can be found in sectio
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 section 6.1 below.

5.3 Information about Explicit Inheritance of Words

Since roles are not arranged into a hierarchy in YAKS, there are no superroles for a role and it is
not directly possible to use the PREFIX and ADJECTIVE annotations, because they rely on inheritance.
A similar statement is true for derived concepts, because, since a derived concept D is described by
conditions that are sufficient but not necessary, no concept can ever be guaranteed to be a superconcept
of D (i.e. to include all instances that D includes). To overcome this problem, there is the SUPERR
annotation to roles (and the analogous SUPERC annotation to concepts). To annotate SUPERR(S) at a
ole R means, that the role R shall inherit nouns from S. Although this form of annotation may seem
unsatisfactory, it is still much better than explicit annotation of complete phrases, because it is
ore compact and retains as much of the usual way the annotations work as possible. SUPERR
notations are not inherited, i.e. they cannot be used across multiple levels.

ration of YAKS and SARA

ove, the knowledge acquisition for the natural language interface in our system consists
ns to the knowledge base. In this section we describe how these annotations are
e frames and how instantiated case frames are converted into a YAKS query

onent in YAKR for each of these tasks: The case frame generator
the annotations and the YAKS knowledge base itself. The case
e frames as produced by the parser into queries that are
e, answers are the results of such queries.

ase and the annotations are used to generate
enerated case frames look like.

ce, concepts describing a property
s head, while other concepts do not. Thus
ory of a concept or role; the following

or *integer*, respectively; no case
ecifiers. All other concepts are
category *action-or-object*

ve *actions* 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 set of cases. Here are two examples:

```
ctions  [C-Call-Actions]  (
t has-agent (C-Functions),
a calls (C-Functions),
ibut with (C-Parameters),
nnung has-synonym (C-Call-Actions))

rameter_HAOd  [R-has-Parameter]  (
t DR (C-Functions) (21),
a RR (R-has-Parameter) (21))
```

ctions and has-Parameter_HAOd are the name of the first and second case frame, respec-
ll-Actions/has-Parameter is the head of the first/second case frame (marked to be a
Each of the indented lines is one case. The components of a case are, in the order
tical role (i.e. the name of an entry in SARA's role table; see appendix A.2), the
used to generate that part of the YAKS query from the instantiated case frame
(if it is filled), the list of allowed fillers for this case (which most often
and optionally the priority mark, which is 20 (and not shown then) by
as mandatory, i.e. it must appear in an instantiation or else that
ix B for a complete listing of the case frames that are generated

concept and role categories what cases and case frames are

s

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

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

E(sc) annotation that have an action-or-object concept
generate a case of the form

frame.

same way as action-or-object concepts, but usually no roles with
ute concept as their domain and no roles with an AT-RANGE

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 Roles

For each has-action-or-object role three case frames are generated. These are named with additional suffixes *_HAOa*, *_HAOe*, and *_HAOd*[3]. 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_HAOa* case frame serves to parse nominal phrases that mention the role itself as a relation such as `der Parameter "X" von Funktion "f"`,
- *has-Parameter_HAOe* parses to-be phrases of the kind `der Parameter ist das Da` (using *has-Parameter_HAOa* to catch the `Parameter`), and
- *has-Parameter_HAOd* parses to-have phrases of the kind `die Funktion "f" ha` (using *has-Parameter_HAOa* to catch the `Parameter "P"`), which is th of the relation described by the role itself.

Of course the actual inputs will usually not be declarative sentences. The *R* role *R* with domain *dom* and range *rng* uses the role itself as its head and has t

```
benennung   has-synonym   (C-rng)
gen_von     INV(R)        (C-dom)
```

The *R_HAOe* case frame of a role *R* with range *rng* uses the word `sein` as its he has the two cases

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

Where the allowed filler of the *agent* case means that the *R_HAOa* case frame of the s be used and the syntactic role *definition* stands for the grammatical case "nominative". Bot are mandatory (i.e. must be filled for an instantiation to be legal). The *R_HAOd* case frame of a ro *R* with domain *dom* uses the word `haben` as its head at parsing time and has the two cases

```
agent   DR   (C-dom)
thema   RR   (R-R)
```

Where the allowed filler of the *thema* case means that the *R_HAOa* 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 Roles

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*, *_HAe*, and *_HAd* and (b) since concrete specifiers usually appear as adjectives, the *definition* case in the *_HAe* 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.

[3] Once upon a time, a *_HAOb* case frame existed, too, but it has been merged into the *_HAOa* 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 YAKS expressions associated with them; they look like the following:

```
agent   DR   (C-D)
thema   RR   (C-R)
```

Again, DR stands for "domain restriction" and RR for "range restriction".

6.1.6 Synonymless Roles

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

6.2 Generation of Queries

This subsection describes how the instantitated case frames generated from an input sentence by the parser are translated into YAKS queries. A detailed description of this translation can be found in [9].

Generally speaking, the head of a case frame instantiation, as well as all of its fillers are considered restrictions. These restrictions are computed and combined into a query expression — in many cases simply by applying the AND operator to them. The computation of the restrictions is recursive; the ursion ends at individual words (to be precise: at those elements of a case frame instantiation that rom individual words) that denote a concept, role, or instance.

e two possible forms of queries: concept queries and role queries. First of all it must be l which of these types should be generated from any given case frame. Role queries can ormation, since they show associations of objects in the answer. We thus try to s as often as possible, i.e., whenever we have an idea about which pairs might be s are generated from (a) all case frame instantiations that have **sein** or **haben** wh-questions, and (c) pure simple noun phrases that are annotated at a role, *Oa* and *_HAu* case frames that have no case filled.

ecursive process of building a query are the following:

ations translate into themselves.

ns translate into themselves, except for instantiations of *_HAOa* and and for the range of the role whenever any cases are filled or the So if the input consisted of nothing more than a synonym of ery, instead of a query for the range of that role, because

l YAKS expression *has-synonym*, i.e. a case filled by

anslated into

5. Any other case with YAKS expression *expr* and filler X is translated into
   `SOME(expr,` X`)`

Somewhat different handling is necessary for relative clauses and for the construction of appropriate role queries for WH-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 to a concept expression. But if it would normally map into a role expression (because the head of stantiation for the relative clause is a role R), it has to be converted into a equivalent concept sion. It is possible to do so, because we know whether the relative pronoun filled (a) a case th *DR* or (b) one marked with *RR*. Let the translation of the filler of the other case in the be F, then return

F`)` for (a), or

`(`R`),` F`)` for (b).

e empty, we use the most general concept *anything* instead.

t is necessary to defer the generation of the query terms that correspond to the until the rest of the query is known. We therefore propagate markers for the F towards the uppermost level of the recursive process. There we can then uery from R, F, and the rest Q of the query restrictions as

`)`

htly more efficient equivalent specialized form

ible to build correct queries from explicit questions for pairs, between the modeling by composing the two roles that the question asks for.

Limitations

notated, sufficiently big database the usefulness of YAKR for an end-user is epository with a short query that is easy to formulate without mastering The recall of such a query is usually high, since a taxonomy and precision is not high enough in the first attempt, it can be (e.g. adding or specializing an attribute), which is easy

ever, is reduced, because the syntactic and semantic out in [5], a limited natural language interface ay of expressing the thing wanted and the user he user can easily learn the restrictions of ow well these conditions are met by our gh experience yet.

isticated natural language r more complex natural example, the re- ery language eful.

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.

7.2 Practicability

The practicability of the implementation of a system such as YAKR 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 the design); a re-implementation could probably done in half the time. Even our prototype is neither
too buggy nor too slow to use it.

e practicability of the database maintenance is difficult to predict in general. If additions to the
base are homogeneous in the sense that they do not require changes in the terminology, they are
maintenance in this case is about as difficult as for any other database with a nontrivial
he other hand, components will be added that require new terminology, some expertise
o keep the terminology clean and coherent. Another problem is the masterability
which is needed if the database contains adequate descriptions of software
t kinds.

cceptance of a system such as YAKR would be high as far as the
ned, but may be low as far as the practicability of database
vious sections). It may, however, make sense, to use the
re or less fixed knowledge base.

ed by the restrictions of its syntactic and semantic

oo difficult to master a big terminology; we do not
val allows to keep both precision and recall
emantically and can be differentiated with
xonomies tend to grow more in width
he number of instances that
l size of the knowledge

e currently use a
brary we have
etween

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
uctures, common processing models, or common modularization strategies. Thus the informality
ormation that is available about the components shows up in our system in the informality
interface we use.

nd, reuse could also take place in a more controlled and formalized environment,
and catalogization) of reusable components and the production of new software
by a common formal framework. In this case other methods to access
ght be superior, namely those that use the information that is
Even in this case, however, YAKS may be a good tool to
should be extended.

d what is usually understood as software reuse:
e process of changing a software system, too.
ems could of course be considered software

nd reusable components. There is of
fferent approaches. This section
ach taken in YAKR.

ponents is that
e components:
one of a
ty of

The software information system that is most similar to ours is LaSSIE/CODE-BASE [14, 32]. LaSSIE
originally used a frame based knowledge representation language called KANDOR, which was later
ed by a language called Classic. In LaSSIE all information in the knowledge base had to be
In LaSSIE's successor, CODE-BASE, a lot of information is acquired by an automatic
REF, and is then stored in a database which is queried on demand. The user
sts of a natural language parser plus a graphical browser for navigating

ts design, some powerful constructs are missing; for example
tion of two roles, the inversion of roles, union of concepts or
Thus KANDOR's expressive power is considerably smaller
educed with the use of Classic; but some gaps
is allowed only for roles with at most one
f roles, negation of concepts or roles, and

as not and avoids storing the whole
LaSSIE/CODE-BASE is
sion of the LaSSIE

[39]. They
he com

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 sort information and constraints on the arguments of nonsort predicates. Finally,
e database schema consists of information that enables the mapping of the intermediate represen-
o a query expressed in a relational query language.

sition 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
s acquired by means of an acquisition dialogue using menus and windows. The
owledge structures and the answers of the user. Verbal case frames are
e given by the knowledge engineer and questions about correctness
This kind of acquisition is motivated by the aim that non-
dapt the interface to a new database. Similar acquisition
s are currently more difficult. However, building such

of TEAM does. Furthermore, information
e deduced when using YAKS. For
c fields just as relations
n process requires less

icted syn-
rical
c

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.

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

10 Further Work

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

striking importance: Is this semantic modeling plus natural language
nts really better than, for instance, much cheaper information
keyword search on documentation files of the components ?

is issue: A part of the programs described in chapter 1
on and recall of our system will then be compared
se programs. We expect that the result of
uivalent queries" and "unsuccessful
pportunities for "near misses".
ss give us an idea where our

lso implemented
nted

A Other knowledge sources of YAKR

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

A.1 SARA dictionary

The dictionary contains all the words that SARA shall be able to recognize. It is implemented partly
full form dictionary and partly with algorithmic word form analysis. The information it delivers
parser are (where appropriate): part-of-speech label, time, casus, numerus, genus, person,
on, and contained prefix. For most types of words, dictionary information rarely needs
see the file saralib/sara.std). But for verbs, nouns, and adjectives additions are
tly, although the dictionary already contains about 10000 words with about 25000
orms plus several times as many implicitly recognizable word forms. The format
described here. All explanations are by example for ease of understanding.

defined in the file saralib/sara.uverben, their format can be deduced from
at for regular verbs as described below.

collected in the file saralib/sis.wb.v and may look like

```
xe (be)  :halbpraefixe ()         :ung }
xe (-)   :halbpraefixe (auf)      :rm }
xe (- inter re)  :halbpraefixe () :rm }
xe (ver) :halbpraefixe (an ab aus vor) :ung }
```

e *name* of the dictionary entry and must be the infinitive form of the

he part-of-speech.

esized list of blank- or comma-separated prefix word parts is
the same stempart (in the example ackern and beackern).
for prefixed verbs: ackern → geackert but beackern → beackert
he dash designates the empty prefix that has the meaning "do not prepend ge
addiert instead of geaddiert. It can be given even where other prefixes
usually necessary for verbs with foreign origin.

following parenthesized blank- or comma-separated list of separable prefixes is
es even more other verbs with the same stempart (in the example fertigen
fference between separable prefix and full prefix is that a separable prefix is cut
form in present tense, e.g., ich fertige etwas ab but not ich abfertige etwas
the Partizip II form is again different for verbs with separable prefix (except if the empty
present) because the ge is inserted *between* the prefix and the stem: fertigen → gefertigt
nfertigen → angefertigt (and not geanfertigt). The dash in the :praefix list eliminates the ge
also in the Partizip II forms with separable prefix, e.g., aufaddiert instead of aufgeaddiert.

Only the Partizip I and Partizp II forms of verbs with separable prefix are explicitly generated in the dictionary as full forms; all others are recognized via algorithmic word form analysis.

The last part of a verb entry is either `:rm` (which is an abbreviation for the also possible `:regelmaessig`) or `:ung`. `:rm` designates the verb as a regular one. `:ung` does the same, but additionally results in the generation of another nominalization: For all verbs (whether regular or not), the infinitive form is automatically put into the dictionary as a noun, too (e.g. ackern → das Ackern). For `:ung` verbs second noun entry is made, in which the en ending of the infinitive form is replaced by ung, e.g., fertigen → das Fertigen, die Fertigung.

The Partizip I and Partizip II forms of all verbs are automatically also available as adjectives and a adverbs.

A.1.2 Nouns

Entries for nouns may look like

```
{ abschnitt :substantiv :typ (Ss, Pe) }
{ adresse   :sub    :typ (S, Pn) }
{ bild      :sub    :geschlecht (s)  :typ (Ss, Per) }
{ algorithmus :sub  :geschlecht (m)  :typ (S) }
{ algorithmus :sub  :stamm algorithmen   :geschlecht (m)  :typ (P) }
{ solo      :sub    :geschlecht (s)  :typ (Ss, Pi) }
{ zeichensatz :sub  :stamm zeichens@atz  :typ (Ss, PUe) }
```

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

The keyword `:typ` precedes the list of inflectional types of the noun. The available types are `S, Ss, Sn, P, Pe, Pn, Pen, Per, Ps, Pss, Pi, Pue, PU, PUe, PUen, PUer`. The `S`-types describe how the genitive singular is formed from the base form: either by appending nothing (`S`), as in die Adresse → der Adresse, by appending s or es (`Ss`), as in das Bild → des Bild(e)s, or by ap n or en (`Sn`), as in der Mensch → des Menschen. From this `S`-type assignment all singul can be (and are) entered into the dictionary (nominative, genitive, dative, and accusative

The `P`-types describe how the nominative plural is formed from the base form:
for `P/Pe/Pn/Pen/Per/Ps` by appending nothing/e/n/en/er/s,
for `PU/PUe/PUen/PUer` by turning the first or @-marked vowel into the corresponding Umlaut ; appending nothing/e/en/er[5],
for `Pss` by turning an ending s into sse,
for `Pi` by turning the latin ending o or us into i, and
for `Pue` by turning the latin ending um into en.

All nouns with singular type `S` are automatically assumed to be female, all others are assumed to male. These heuristics find the correct gender for about 80 percent of all nouns. For the rest, gender can be announced explicitly with the keyword `:geschlecht` followed by a parenthesized list (!) of the letters `m` (for male), `w` or `f` (for female), and `s` or `n` for neutral. Multiple genders can be assigned to a single word.

If the noun is irregular, it may be necessary to assign different stems for singular and plural forms. How to handle this case is shown with the example algorithmus above.

[5] e.g. der Zeichensatz, die Zeichensätze

A.1.3 Adjectives

Entries for adjectives may look like

```
{ absolut   :adj }
{ bedeutend :adj :steigerungsstaemme (bedeutend, -,
                                      (bedeutenst, bedeutendst)) }
{ public    :adj :ungebeugt }
```

The first part of the entry is the word name, which must be the base form of the adjective. `:adj` (o alternatively `:adjektiv`) is the key word that assigns the part-of-speech. If the comparation endings are not **er**, **est**, the complete base forms for positive, comparative, and superlative can be given as a list of words (or word lists for alternative forms) after the keyword `:steigerungsstaemme`. 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 a inflectional ending by giving the keyword `:ungebeugt` in the entry last. This is needed to handle german usage of english adjectives.

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

A.2 SARA role table

The role table associates the names of syntactic roles with a set of grammatical constructions and a set of question phrases that can be used to refer to this syntactic role. The standard set of syntactic roles resides in the file `saralib/sara.std` and has 33 entries. Although it does not need to be changed very often, its format is described here.

A very general example of a role table entry is the following:

```
<< agent "das Subjekt"
    :nur_aktiv
      :nominativ         ;'ich' schlage keinen Hund
    :nur_passiv
      :dativ von         ;'von mir' wird kein Hund geschlagen
      :dativ vom         ;'vom Nachbarn' werden alle Hunde geschlagen
      :akkusativ durch   ;'durch mich' werden keine Hunde geschlagen
    :nur_nominal
      :genitiv           ;das Klagen 'des Nachbarn'
      :dativ von         ;das Klagen 'von mir'
      :dativ vom         ;das Klagen 'vom Nachbarn'
      :akkusativ durch   ;das Klagen 'durch den Nachbarn'
  :fragen
    :nur_aktiv
      :frage (wer)       ;'wer' fragt mich
      :frage (was)       ;'was' krabbelt meinen Ruecken hinauf
    :nur_passiv
      :frage (von wem)   ;'von wem' werde ich gefragt
      :frage (von was)
      :frage (durch wen)
      :frage (durch was)
      :frage (wo)        ;(sinnvoll, wenn die Agenten SW-Objekte sind.)
>>
```

This entry can be read as follows: agent is the name of the entry (as to be used in AT-DOMAIN and AT-RANGE annotations). The following string is merely a free form description of the entry. All of the following is optional, except the keyword :fragen.

:nur_aktiv means "the following appearance forms are valid for verbal phrases (i.e. clauses) in active voice only". :nominativ means "one possible appearance of the 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 for this appearance form.

ur_passiv means "the following appearance forms are valid for verbal phrases in passive voice only". iv von means "one possible appearance of the agent role is a noun in dative case preceded by the on von". Again, the part of the line after the semicolon is a comment and gives an example rance form. The explanations for the other appearance entries are analogous.

eans "the following appearance forms are valid for nominal phrases only". There ance cases before the :nur_aktiv keyword. This had meant that they should .e. active and passive verbal phrases as well as nominal phrases.

-optionally) needed for purely syntactic reasons, to ease parsing the

wer at the beginning of a sentence indicates that the sentence maybe y many question words may be given and must all appear in exactly rm.

in the :fragen section of the entry.

s Welche Funktion are implicitly derived from all the keyword, that are valid for verbal phrases.

e Frames

erated from the example modeling on page 11. The listing

```
ons))

)

s))

tions),
s))

))
```

```
  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_HAOd  [R-has-Parameter]  (
  agent DR (C-Functions) (21),
  thema RR (R-has-Parameter) (21))
has-Parameter_HAOc  [R-has-Parameter]  (
  agent RR (R-has-Parameter) (21),
  definition RR (C-Data-Objects) (21))
has-Parameter_HAOa  [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_HAOc)
Fallschablonen zu 'haben': (has-Parameter_HAOd)
```

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 _HAOc, _HAOd, _HAc, and _HAd case frames by a unification algorithm. This method is used to avoid the extremely long running time of the parser that would result if all sein/haben case rames were directly used for parsing. GF is the so called *general frame* that is used to parse inputs of form Welche A sind B where both A and B are object concepts.

User interface commands

The help page of the SARA command interpreter looks like this:

```
   - Quit. Beende die Sitzung
   - Rufe eine Shell auf
   - Lese SARA-Wissensbasis (kann #krsinclude enthalten)
   - Lese reine YAKS-Wissensbasis
 k - Zeige Woerterbucheintraege/Konzepte
 f - Zeige Rollen/Fallschablonen
   - Teste die Wissensbasis auf Konsistenz
   - Erzeuge die Fallschablonen
   - Anfragen anzeigen (ein/aus)
 i - Charts/Instanziierungen anzeigen (ein/aus)
   - Frage nach Ausgabedateinamen bei w,k,K,r,f,t (ein/aus)
   - Dateinamen aendern
 I - Charts/Instanziierungen auf EDGE-Datei schreiben (ein/aus)
   - Konzepthierarchie einmal auf EDGE-Datei schreiben
   - Lenke die Standardeingabe um
   - Dump der Wissensbasis nach |less
   - wiederhole letzten Eingabesatz
   - YAKS-Kommandointerpretierer aufrufen (Verlassen mit 'quit')
   - Einlesen des binaeren Woerterbuches aus ~/tmp/sara.wb.bin.Z
```

e of these commands are described now.

ts for a filename and reads that file as a SARA knowledge file, which may contain (a) dictionary e table entries, (c) #include of other SARA files, and (d) #krsinclude of YAKS files. n as a file name and denotes the standard input.

and reads that file as a YAKS knowledge base file.

r and display, respectively, the corresponding dictionary entries, concepts, rames from the SARA knowledge base. The output should be more or less n the output are prefixed by either R-, C- or I- to display that r instance, respectively, in YAKS. Role concepts are further r ?_ to display that the role is a synonymless, theme, has- e, respectively. Concept concepts may be further prefixed cept is an action-or-object, attribute, or other concept. with a question mark, all objects whose names contain If the string is completely 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 utput file as for the input file.

Example Session

s section contains a short example session with YAKR, using the example knowledge base from above. utput is in `this typeface` while user input is in *this typeface*. Comments are indented.

```
n bei YAKR          ? fuer Hilfe
```

```
.wb.bin.Z : sis.wb......./....................
```

Now the SARA dictionary has been read in.

```
sensbasis ein.
 SARA-Wissensbasis:
```
bsp

rk: The contents of the file `bsp` are

```
clude    "sara.std"
sinclude "daten/bsp_yaks"
```

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

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

n ?

This command toggles the displaying of the instantiated case frames and the YAKS queries generated from them.

SARA: *Funktionen, die ausgeben*

```
"Funktionen, die ausgeben" -->

[:I L3 G4 C-Functions,
        [Gw Fngda Np P3 substantiv funktion]
  [:I 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]]]]]

Q-CONCEPT ?cvar = AND(Functions
    SOME(writes)).  /* Concept */

fputc fputs
```

SARA: *q*

Tschuess !

E Possible Problems with Natural Language Interfaces

The following hypothetical dialog may serve to illustrate some of the difficulties that may arise in the use of intelligent systems with natural language interfaces. Actually, this projection was made in the year 1973, so the setting is with a slow printing terminal. Please take care to understand the initial instructions.

Hello, I am Your utility service computer. To provide the best possible se
programmers have given me the capapbility t
learn.
Enter y

Credit card number validated.
What may I serve you, Mr. Rogers ?
//Do you provide library reference services ?
*Yes. I have access to 1.27*10E+7 v*
//1.

Do you wish the listings in random order, alphabetic order by title, a
INTERRUPTED***

//By date of publication!

Earliest or latest first?

//Latest!

First or last?

//Eithe*

//Fir

!

?

!

/

Published hardbound and undated paperback references for subject "gypsu
paperback publications are included in mag

//I don't want to know a
//

```
z|4Tn:DxTNFTH.VrLB(J22tnbZ‘p0Z*Vr@\tlPFRPB&6b2bR|Z
V(&&x^b:h&‘L(J8XZ*:N\ NdpX(< vnNZD.,FFfN
F80&>XXNJj?
```

(From: DATAMATION, April 1973, pp 72–73, by Donald Kenney)

For full enjoyment, now start again and read *carefully*.

ferences

[1] Rolf Adams. KRS — a hybrid system for representing knowledge in knowledge-based help systems. In *[21]*, 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. to appear.

[3] Emmon Bach and Robert T. Harms, editors. *Universals in Linguistic Theory*. Holt, Rine Winston, Inc, London, Reprint 1972.

[4] Bruce Ballard. A lexical, syntactic, and semantic framework for a use question-answering system. In Martha Evens, editor, *Lexic* Cambridge University Press, 1988.

[5] M. Bates and R. Bobrow. *Natural Language Interfac* chapter 10, pages 179–194. Ablex Publishing Compan

[6] Ted Biggerstaff and Charles Richter. Reusab 4(2):41–49, March 1987. also in

[7] Ted J. Biggerstaff and ACM Press Frontier S

[8] R. J. Brach *Sci*

[9]

[17] Martin Fischer. Realisierung eines Wissensrepräsentationssystems für die Repräsentationsspra
Master's thesis, Universität Karlsruhe, D-7500 Karlsruhe, 1991.

[18] G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumas. The vocabula
communication. *Communications of the ACM*, 30(11):964–971, November 1

[19] Keith E. Gorlen, Sanford M. Orlow, and Perry S. Plexico. *Data A
ming in C++*. Wiley, Chichester, 1991.

[20] B. Grosz, D. Appelt, P. Martin, and F. Pereira. TEAM
natural language interfaces. *Artificial Intellig

[21] IEEE Computer Society. *The Seventh
Florida, February 1991. IEEE Comput

[22] Michael Loren Mauldin. *Inform
Carnegie Mellon Universit

[23] Andy Podgurski
co
May 19

[24] Lutz