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.

bservations 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.

he 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

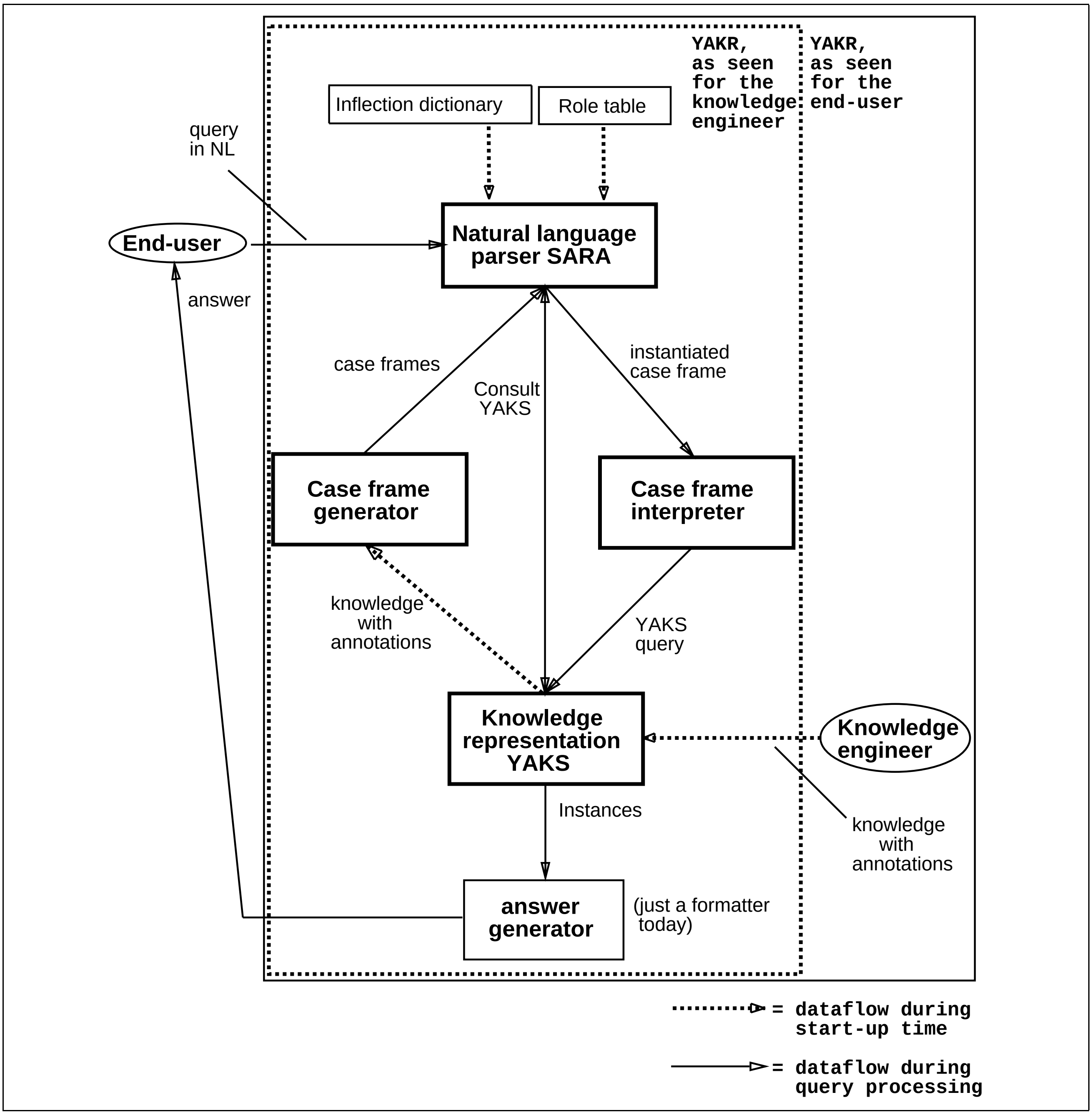

Figure 1: Basic architecture and dataflow of YAKR

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

jects (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:

fgetc 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., cept 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
ctable 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 c
simple conjunctions and
questions, declaratives, im
dal verbs, immediate relati
ses starting with a conjunctio
general numbers, general q

Case frame parsers convert wri
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 t 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.

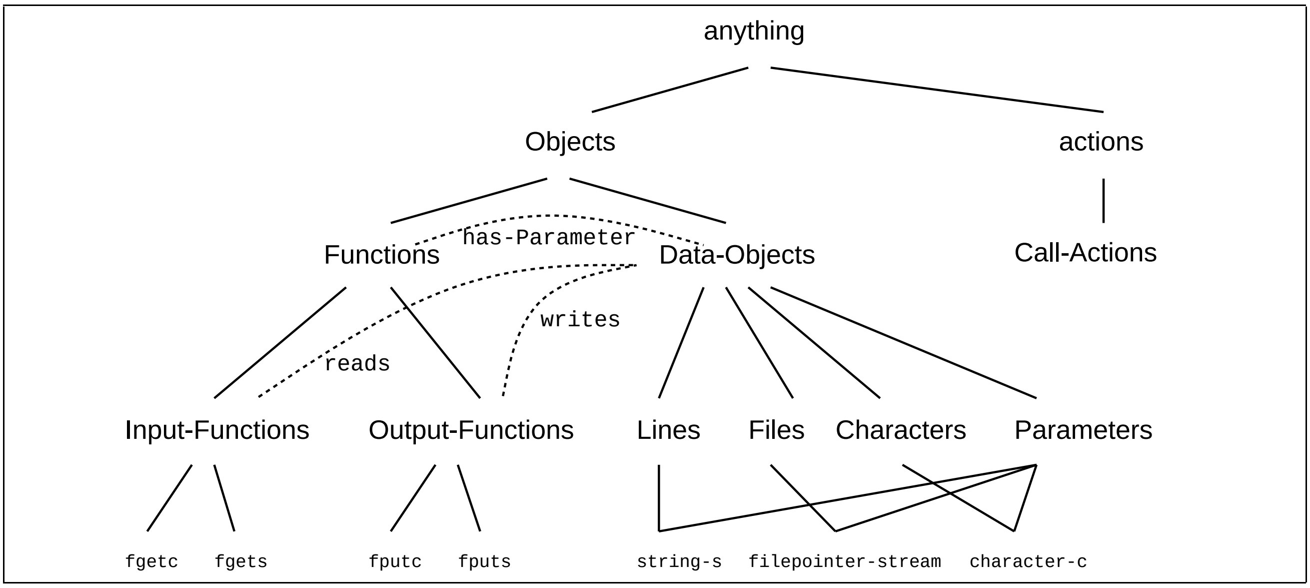

Figure 2: Taxonomy of the example knowledge base

5.1 Information about Individual Words

simplest form of annotation to a concept or role X is the synonym list: The VERB and NOUN
ons give a list of verbs and non-compound nouns, respectively, that can denote the concept or
notate. 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:
enever an annotation to X specifies a part P of a natural language construct to be *added* to
er construct A, then A is inherited from the superconcepts or superroles of X. For instance,
de Funktion (reading function) refers to *Input-Functions*, where Funktion is inherited from the annotation of the direct superconcept *Functions*. The word lesend 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: lesendes 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 lesend might annotate other subconcepts 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
d just annotated in the NOUN list. However, this would be extremely tedious, since compound nouns
versatile and ubiquitous in German. To solve this problem, the PREFIX annotation to X
le prefix nouns that can be prepended to the nouns annotated with the superconcepts of
m a reference to X. This annotation is analogous to the ADJECTIVE annotation. For
ion (read function) refers to *Input-Function*. Only the words Lesen[2] 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.

n adjective or noun prefix to specialize a noun, it is often possible to use a pre-
at is placed right behind the noun. This possibility can be expressed with the
e) annotation to X: It gives a case (with a filler) that, when used together
ted to a superconcept of X, denotes X. This annotation is an extension
sm, because it produces a case that does not get inserted into a case
filler or head instead. For instance Funktion zum Lesen (function for
ctions. The possible grammatical forms for this reference are listed in the
(see 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
rating 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
rts; then the SYN-CASE annotation is a simple way to increase the coverage of
erface.

s

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

OMAIN(sc) at a role R creates a case in the case frame
this case is the range concept of R; it has to appear

ictionary, because its nominalization is automatically in a grammatical form described by the role table entry *sc* (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 *sc*. 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
tations and some case frames can be built completely automatically. These details are explained
n 6.1 below.

rmation about Explicit Inheritance of Words

o a hierarchy in YAKS, there are no superroles for a role and it is
FIX and `ADJECTIVE` annotations, because they rely on inheritance.
ed concepts, because, since a derived concept D is described by
ssary, no concept can ever be guaranteed to be a superconcept
ncludes). To overcome this problem, there is the `SUPERR`
annotation to concepts). To annotate `SUPERR(S)` at a
rom S. Although this form of annotation may seem
it annotation of complete phrases, because it is
e annotations work as possible. `SUPERR`
sed across multiple levels.

n our system consists
annotations are
S query

r

ions 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),
  thema calls (C-Functions),
  attribut with (C-Parameters),
  benennung has-synonym (C-Call-Actions))

has-Parameter_HAOd  [R-has-Parameter]  (
  agent DR (C-Functions) (21),
  thema RR (R-has-Parameter) (21))
```

Call-Actions and has-Parameter_HAOd 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 own, the syntactical role (i.e. the name of an entry in SARA's role table; see appendix A.2), the xpression to be used to generate that part of the YAKS query from the instantiated case frame ls to this case (if it is filled), the list of allowed fillers for this case (which most 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 al. See appendix B for a complete listing of the case frames that are generated ng.

f the relevant concept and role categories what cases and case frames are rator:

ncepts

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

(C-AO)

hind a noun in phrases such as die Funktion "f", where it catches lue to AT-DOMAIN/AT-RANGE annotations: All roles R with an ction-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.

on-or-object concepts, but usually no roles with heir 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 after the role with additional suffixes *_HAOa*, *_HAOc*, 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_HAOc* parses to-be phrases of the kind `der Parameter ist das Datenobjekt “B` (using *has-Parameter_HAOa* to catch the `Parameter`), and
- *has-Parameter_HAOd* parses to-have phrases of the kind `die Funktion “f” hat den Pa` (using *has-Parameter_HAOa* to catch the `Parameter “P”`), which is the most natu of the relation described by the role itself.

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

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

The *R_HAOc* 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)

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” are mandatory (i.e. must be filled for an instantiation to be legal). The *R_HAOd* case frame *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 m be used and the syntactic role *thema* stands for the grammatical case “accusative”. DR stands fo “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*, *_HAc*, and *_HAd* and (b) since concrete specifiers usually appear as adjectives, the *definition* case in the *_HAOc* 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 the role and one for its range R. These cases have always the same syntactical roles and YAKS ons associated with them; they look like the following:

t DR (C-D)

a RR (C-R)

stands for "domain restriction" and RR for "range restriction".

ynonymless Roles

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

stantitated case frames generated from an input sentence by the A detailed description of this translation can be found in

e instantiation, as well as all of its fillers are considered ed and combined into a query expression — in many cases e computation of the restrictions is recursive; the those elements of a case frame instantiation that or instance.

s and role queries. First of all it must be any given case frame. Role queries can ts in the answer. We thus try to dea about which pairs might be tions that have sein or haben t are annotated at a role,

owing:

iations of $\_HAOa$ and es are filled or the an a synonym of t role, because

e. a case filled by

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 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
ession. 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
use be F, then return
R, F) for (a), or
(INV(R), F) for (b).
uld be empty, we use the most general concept $anything$ instead.

uestions it is necessary to defer the generation of the query terms that correspond to the
asked for until the rest of the query is known. We therefore propagate markers for the
its filler F towards the uppermost level of the recursive process. There we can then
te role query from R, F, and the rest Q of the query restrictions as
RR(F))
the slightly more efficient equivalent specialized form

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

d Limitations

ciently big database the usefulness of YAKR for an end-user is
short query that is easy to formulate without mastering
uch a query is usually high, since a taxonomy and
t high enough in the first attempt, it can be
specializing an attribute), which is easy

d, because the syntactic and semantic
imited natural language interface
the thing wanted and the user
ly learn the restrictions of
nditions are met by our
et.

language
tural

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 Practicability

cticability of the implementation of a system such as YAKR is good: The software is of moderate
l be produced by a small team in some months. It took about 4 person years (including
lementation could probably done in half the time. Even our prototype is neither
use it.

latabase maintenance is difficult to predict in general. If additions to the
he sense that they do not require changes in the terminology, they are
e is about as difficult as for any other database with a nontrivial
will be added that require new terminology, some expertise
an and coherent. Another problem is the masterability
abase contains adequate descriptions of software

R would be high as far as the
practicability of database
make sense, to use the

ic and semantic

do not
ll

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.

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

There is of section

ss.

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

ign, some powerful constructs are missing; for example
two roles, the inversion of roles, union of concepts or
NDOR's expressive power is considerably smaller
with the use of Classic; but some gaps
owed only for roles with at most one
, negation of concepts or roles, and

nd avoids storing the whole
ODE-BASE is
he LaSSIE

ey

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.

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

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

rther Work

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

tance: Is this semantic modeling plus natural language
ter than, for instance, much cheaper information
on documentation files of the components ?

rt of the programs described in chapter 1
our system will then be compared
expect that the result of
es" and "unsuccessful
or "near misses".
dea where our

A Other knowledge sources of YAKR

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

tionary

all the words that SARA shall be able to recognize. It is implemented partly
partly with algorithmic word form analysis. The information it delivers
riate): part-of-speech label, time, casus, numerus, genus, person,
For most types of words, dictionary information rarely needs
ra.std). But for verbs, nouns, and adjectives additions are
nary already contains about 10000 words with about 25000
many implicitly recognizable word forms. The format
anations are by example for ease of understanding.

ra.uverben, their format can be deduced from
d below.

sis.wb.v and may look like

```
           :ung }
uf)        :rm }
efixe () :rm }
n ab aus vor) :ung }
```

y and must be the infinitive form of the

ma-separated prefix word parts is
ample ackern and beackern).
kert but beackern →beackert
aning "do not prepend ge
n even where other prefixes

f separable prefixes is
example fertigen
ble prefix is cut
ch abfertige etwas
cept if the empty
ertigen →gefertigt
ix list eliminates the ge
of aufgeaddiert.
explicitly generated in the
analysis.

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

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

Nouns

for nouns may look like

```
itt :substantiv :typ (Ss, Pe) }
e   :sub    :typ (S, Pn) }
    :sub    :geschlecht (s)  :typ (Ss, Per) }
thmus :sub  :geschlecht (m)  :typ (S) }
thmus :sub  :stamm algorithmen   :geschlecht (m)  :typ (P) }
    :sub    :geschlecht (s)  :typ (Ss, Pi) }
nsatz :sub  :stamm zeichens@atz  :typ (Ss, PUe) }
```

art of the entry is the word name, which must be the base form of the noun. :sub (or
ubstantiv) 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,
en, Per, Ps, Pss, Pi, Pue, PU, PUe, PUen, PUer. 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
r Mensch →des Menschen. From this S-type assignment all singular noun forms
nto the dictionary (nominative, genitive, dative, and accusative case).

e nominative plural is formed from the base form
pending nothing/e/n/en/er/s,
ng the first or @–marked vowel into the corresponding Umlaut *plus*

se,
or us into i, and
m into en.

omatically assumed to be female, all others are assumed to be
er for about 80 percent of all nouns. For the rest, gender
eschlecht followed by a parenthesized list (!) of the
n for neutral. Multiple genders can be assigned to a

sign different stems for singular and plural forms.
rithmus above.

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` (or 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 eded to handle german usage of english adjectives.

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

ARA role table

ociates the names of syntactic roles with a set of grammatical constructions and a set hat can be used to refer to this syntactic role. The standard set of syntactic roles ralib/sara.std and has 33 entries. Although it does not need to be changed lescribed here.

a role table entry is the following:

```
   ;'ich' schlage keinen Hund

   ;'von mir' wird kein Hund geschlagen
   ;'vom Nachbarn' werden alle Hunde geschlagen
   ;'durch mich' werden keine Hunde geschlagen

   ;das Klagen 'des Nachbarn'
   ;das Klagen 'von mir'
   ;das Klagen 'vom Nachbarn'
   ;das Klagen 'durch den Nachbarn'


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

   ;'von wem' werde ich gefragt

n)
s)
   ;(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 -RANGE annotations). The following string is merely a free form description of the entry. All of the owing is optional, except the keyword :fragen.

_aktiv means "the following appearance forms are valid for verbal phrases (i.e. clauses) in active nly". :nominativ means "one possible appearance of the agent role is a noun in pure nomi- (i.e. without a preposition)". The part of the line after the semicolon is a comment and e for this appearance form.

ns "the following appearance forms are valid for verbal phrases in passive voice only". one possible appearance of the agent role is a noun in dative case preceded by the the part of the line after the semicolon is a comment and gives an example explanations for the other appearance entries are analogous.

ing appearance forms are valid for nominal phrases only". There e the :nur_aktiv keyword. This had meant that they should assive verbal phrases as well as nominal phrases.

eded for purely syntactic reasons, to ease parsing the

ning of a sentence indicates that the sentence maybe rds may be given and must all appear in exactly

section of the entry.

ion are implicitly derived from all the d for verbal phrases.

page 11. 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_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 ppropriate _HAOc, _HAOd, _HAc, and _HAd case frames by a unification algorithm. This method is the extremely long running time of the parser that would result if all sein/haben case tly used for parsing. GF is the so called *general frame* that is used to parse inputs of che *A* sind *B* where both A and B are object concepts.

terface commands

A command interpreter looks like this:

```
 Sitzung
auf
sbasis (kann #krsinclude enthalten)
Wissensbasis
heintraege/Konzepte
lschablonen
sbasis auf Konsistenz
schablonen
n (ein/aus)
erungen anzeigen (ein/aus)
bedateinamen bei w,k,K,r,f,t (ein/aus)
rn
erungen auf EDGE-Datei schreiben (ein/aus)
e einmal auf EDGE-Datei schreiben
rdeingabe um
basis nach |less
en Eingabesatz
erpretierer aufrufen (Verlassen mit 'quit')
aeren Woerterbuches aus ~/tmp/sara.wb.bin.Z
```

scribed now.

hat file as a SARA knowledge file, which may contain (a) dictionary
ude of other SARA files, and (d) #krsinclude of YAKS files.
the standard input.

S knowledge base file.

he corresponding dictionary entries, concepts,
base. The output should be more or less
either R-, C- or I- to display that
YAKS. Role concepts are further
is a synonymless, theme, has-
cepts may be further prefixed
attribute, or other concept.
ects whose names contain
ely 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 tput file as for the input file.

xample Session

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

```
 YAKR          ? fuer Hilfe

in.Z : sis.wb......./.....................
```

he SARA dictionary has been read in.

```
basis ein.
A-Wissensbasis:
```
bsp

The contents of the file `bsp` are

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

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

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

ausgeben

n geben etwas aus

```
tring-s
haracter-c
```

ndeine Funktion ein Zeichen einlesen?

```
etc reads character-c
RA:
```
Ausgabefunktionen

```
    fputc     fputs
    SARA:
```
Funktionen, die ausgeben

```
            fputc     fputs
            SARA:
```
i

```
             Instanziierungen anzeigen wird EINgeschaltet
```

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

onen, 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 */

fputs
```

q

```
ss !
```

ssible Problems with Natural Language Interfaces

cal dialog may serve to illustrate some of the difficulties that may arise in the
with natural language interfaces. Actually, this projection was made in the
with a slow printing terminal. Please take care to understand the initial

vice computer. To provide the best possible service to You, my
en me the capapbility to converse in basic English as well as the ability to

equests from the keyboard.
sts must start with //
*your typein, type *** and press the CARRIAGE RETURN key.*
t my typeout, depress the MARGIN RELEASE key.
ypein, backspace and type over the errorneous characters.
a typein, type ! or ?.
minate my services, type "finished!".
will be billed for the time used at the end of the month.
What is your credit card number ?

//372AG-2631AFX.!

What is your name ?

//J. Wonsely 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*10E+7 volumes of books and 3.26*10E+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 1100000110101000000000000*
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", "gypsum", and "futu
wish displayed ?

//I want to know about Tarot cards.
!
//!

Tarot cards ?

//OK! You win. I wa
"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 b
grouped by publisher, or odered by date of publicati

//Whatever.
//!

"Whatever." meaning unclear. Please clarify.

//Present the data in whatever order yo

wish the listings in randomorder, alphabetic order by title, alphab`***TYPEOUT INTERRUPTED***`

//By date of publication!

Earliest or latest first ?

//Latest!

First or last ?

//Eithe`***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***`
//Omit the classified references.
//!

*Several references are listed as restricted information. If you wish acce***

Omit the restricted information.
!
//!

No input received. Please enter next request.

//Omit the restricted information!

*Published hardbound and undated paperback references for subject "gypsum" follow. Dated paperback publications are included in maga****TYPEOUT INTERRUPTED***

//I don't want to know about Gypsum I want to know about Tarot cards.
//Cancel
!
!
?
!
//!

"Cancel" meaning unclear. Please clarify.

//Finised!

Word "Finised" not in vocabulary tables. Please provide definition.

//Finished!

Request illegal at this time. Please provide definition of word "Finised".

//"finised", transitive verb, from the french "fini".
//Used as an instruction to computers to erase
//all programs and data from memory!

Definition recorded. Thank you. Enter next request.

//You're quite welcome. Finised!

```
@&‘\J\
8pF.pB4ZL^T\v0^x$n( Zd:‘lv0(TL<D|.XvfNT.6(t^^|4,,Jl|jll(^r>T
||B^NNx"XX(JxZ<L668<pz|.vtx*VL@xP,vZZ ZPDz JzD
X$PTHnjnbd6H428p6n:0Fn,D>4(.jr|P4<ljZ6rjR j,,Z<4hnn|@n<Rlx
v$L<$b(‘^xx.<<\ @x,Jn.((2T6dr"lFrTJNrR|0H|(r&0F\Rd@L48:FpJ
LB"2v
|:|"XxRZHt&^<DjpXd*n.znD4L<6‘ vfn
R|,VP.(2X>@‘D&R\v@pN4v8V&D8R8bJX‘p&V@vTV>Z v:*f0L$V‘x\x2
<&d,$|ld‘8$T|TDr>vXl\D$6D\bTn4:,2tvhlR
8pVh(^@X"vD8 bPdJP"",fphXl
p6fxB6V.>h!!!!!!!!!!&h*0*(.&"<,(N4>.L8‘nFd
$Hz88rl0DBF:H84z6$VpBjlNrJt@ZB&lbJ(*LJ0
VddfR$Tz|N&HX&2<hfNrt.‘L\ ZhD ^D@:Z\zBJPt6,:
Ln@Z6\nLLB<0&*LL&B"8Xz.‘*BB.,z(f$x&tj@0NxTPhTDdp4Xp,n.&
PDLj2P||.LJ&z <^.D8$^dL^l^DRZ$2\ zf8z^^>B8>bv,
\Zbrn>V4r:2r4\0Ln D.lL^RJL@D>"2H8RTPd*,"VJ&2
"bttR,xx2|d@LxTF rzp????V$Lh‘ PzLTNxzZdb>bf46bDN4|4‘ v
f**r,V2PT2v@dj|\jl"4(‘LLjLBDj4X: |:npPn60l\LnZ,4">
^Z$‘PvJFFp
4vb0|@Hv$6|‘f|tFjn2
$nbj‘tLZrzjLJPjXtpbbNVh4JvjzlD"NVH&bld
^,|*LXX&L2^.ht
 <h"hP,\Hr.|&DX6$lzV@h4v\t6$>
>$Hll(.P lD\X88F2,pj68pXd.L ^JP(tFB\jn(vxT v|V:p|F
t42vH2<‘86Z""tv*Lrpd‘r>2dzZ
```

```
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)

ull enjoyment, now start again and read *carefully*.

es

s. KRS — a hybrid system for representing knowledge in knowledge-based help systems. In
ages 129–133, 1991.

f Adams. *Wissensrepräsentation und -akquisition in einem natürlichsprachlichen Software-*
ionssystem. PhD thesis, Institut für Programmstrukturen und Datenorganisation, Universität
sruhe, November 1992. to appear.

mmon Bach and Robert T. Harms, editors. *Universals in Linguistic Theory.* Holt, Rineheart, and
Vinston, Inc, London, Reprint 1972.

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

[5] M. Bates and R. Bobrow. *Natural Language Interfaces: What's Here, What's Coming, and Who Needs It*, chapter 10, pages 179–194. Ablex Publishing Company, 1984.

[6] Ted Biggerstaff and Charles Richter. Reusability framework, assessment, and directions. *IEEE Software*, 4(2):41–49, March 1987. also in [36] and in [7].

[7] Ted J. Biggerstaff and Alan J. Perlis, editors. *Software Reusability — Concepts and Models, Volume I.* ACM Press Frontier Series. Addison-Wesley, Reading, Mass., 1989.

[8] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

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

[10] J. Carbonell. Requirements for robust natural language interfaces: The Language Craft an experiences. In *International Conference on Computational Linguistics*, pages 162–163. Associ Computational Linguistics, 1986.

[11] J. Carbonell and P. Hayes. Coping with extragrammaticality. In *International Conferen tional Linguistics*, pages 437–443, 1984.

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

[13] Jaime G. Carbonell and Philip J. Hayes. Robust parsing using multiple const *Proceedings of the Seventh IJCAI*, pages 432–439, August 1981.

[14] P. Devanbu, R.J. Brachman, P.G. Selfridge, and B.W. Ballard. La information system. *Communications of the ACM*, 34(5):34–49, May 199

[15] P. Devanbu, P. Selfridge, B. Ballard, and R. Brachman. A knowledge In *International Joint Conference on Artificial Intelligence*, pages 110- 1989.

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

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

[18] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumas. The vocabulary problem in human–system communication. *Communications of the ACM*, 30(11):964–971, November 1987.

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

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

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

[22] Michael Loren Mauldin. *Information Retrieval by Text Skimming*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1987. also as Techreport CMU-CS-89-193.

[23] Andy Podgurski and Lynn Pierce. Behavior sampling: A technique for automated retrieval of components. In *Proceedings of the 14th International Conference on Software Engineering*, pages 349–3 May 1992.

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

[25] R. Prieto-Diaz and P. Freeman. Classifying software for reusability. *IEEE So* 1987.

[26] Ruben Prieto-Diaz. *A Software Classification Scheme*. PhD thesis, Department puter Science, University of California, Irvine, CA, 1985.

[27] Ruben Prieto-Diaz. Classficiation of reusable modules. In *in [7]*,

[28] Ruben Prieto-Diaz. Implementing faceted classification fo 34(5):89–97, May 1991.

[29] M. Ratcliffe. Report on a workshop on software reuse h *SIGSOFT Software Engineering Notes*, 12(1):42–47, January 1

[30] Eugene J. Rollins and Jeannette M. Wing. Specifications *of the International Conference on Logic Programming*, Pa

[31] R. C. Schank. Conceptual Dependency: A theory of nat pages 552–631, 1972.

[32] Peter G. Selfridge. Knowledge representation s 134–140, 1991.

[33] S. Shieber, editor. *An Introduction to U* Chicago University Press, 1986.

[34] Tore Syvertsen. CPPREF — an informatio strukturen und Datenorganisation,

[35] Will Tracz. Software reuse: IEEE, IEEE Computer Society

[36] Will Tracz, editor. *Soft* shington, D.C., 1988.

[37] D. Warren and F. Pereir *American Journal*

[38] Patrick Henry Wi

[39] Murray Wood and *Engineering Jo*

[40] Scott N. Woo 4(4):52–59, J