In: Gollan, B., Paul, W.J., Schmitt, A. (eds.): Innovative Informationsinfrastrukturen. Informatik-Fachberichte 184, Heidelberg: Springer, S. 81-100.

SC: Ein intelligentes Hilfesystem für SINIX

Wolfgang Wahlster Matthias Hecking Christel Kemke*

FB 10 Informatik IV

Abstract

The SINIX Consultant (SC) is an intelligent help System for the SINIX operating System. SC represents a help system which combines a passive and an active mode in order to aid the User. In the passive mode, SC answers natural language questions about SINIX objects and commands in a user-oriented tutorial manner. In the active mode, it gives unsolicited advice to the user by detecting and correcting inefficient plans he is using.

To fulfill these tasks, SC has to go beyond typical help system capabilities by providing natural language dialog facilities, a model of the user's knowledge, and a recognition of his actual plans and goals. The basis for performing the required problem solving and explanation tasks is a complex SINIX knowledge base describing commands and actions in the SINIX domain.

1. Intelligente Hilfesysteme

Ohne Hilfe bei der Bedienung ist ein Rechensystem für den Endbenutzer wertlos. Die Akzeptanz komplexer Systeme hängt beim Anwender daher entscheidend von der Qualität der Hilfen ab, die ihm zur Lösung seiner Aufgabenstellung während der Arbeit mit dem System angeboten werden.

Das Spektrum der verfügbaren Hilfeleistungen kann von Handbüchern mit Bedienungsanleitungen bis hin zur persönlichen Beratung durch einen Systemexperten reichen. Das Suchen in Handbüchern ist für den Anwender die aufwendigste Möglichkeit, sein Bedienungsproblem zu lösen, wobei das Ergebnis oft unbefriedigend bleibt, da die gefundenen Hinweise weder auf das individuelle Vorwissen des Benutzers noch auf dessen bisherige Interaktion mit dem System als dem Kontext des Bedienungsproblems abgestimmt sein können. Das andere Extrem ist der stets hilfsbereite Systemkenner, der dem Benutzer das Optimum an Hilfeleistungen bietet, indem er ihm gezielt Hinweise und Ratschläge erteilt, die dem jeweiligen Vorwissen und Problemlösungszustand angemessen sind. Bei der starken Verbreitung von PCs und Arbeitsplatzrechnern ist es die Ausnahme, daß jederzeit auf einen Systemberater zurückgegriffen werden kann, der die Zeit und Geduld hat, alle Fragen des Benutzers zu beantworten.

Schon frühzeitig kam man auf die Idee, das Rechensystem selbst für Hilfestellungen bei akuten Bedienungsproblemen zu nutzen. Für die meisten Rechner gibt es heute Hilfesysteme, die Textpassagen aus den jeweiligen Bedienungshandbüchern über Schlüsselwortsuche oder Menü-Auswahl im Dialogbetrieb zugänglich machen. Solche einfachen Hilfesysteme ersparen im besten Fall das Suchen in Handbüchern; sie sind aber weit entfernt von dem Nutzen und der Effizienz eines Beratungsgesprächs mit einem versierten Fachmann.

Mit *intelligenten Hilfesystemen* versucht man nun, die von Systemexperten erbrachten Beratungsleistungen möglichst weitgehend maschinell verfügbar zu machen. In diesem Beitrag wird das von uns entwickelte intelligente Hilfesystem SC (S1N1X Consultant) für das Betriebssystem SIN1X vorgestellt.

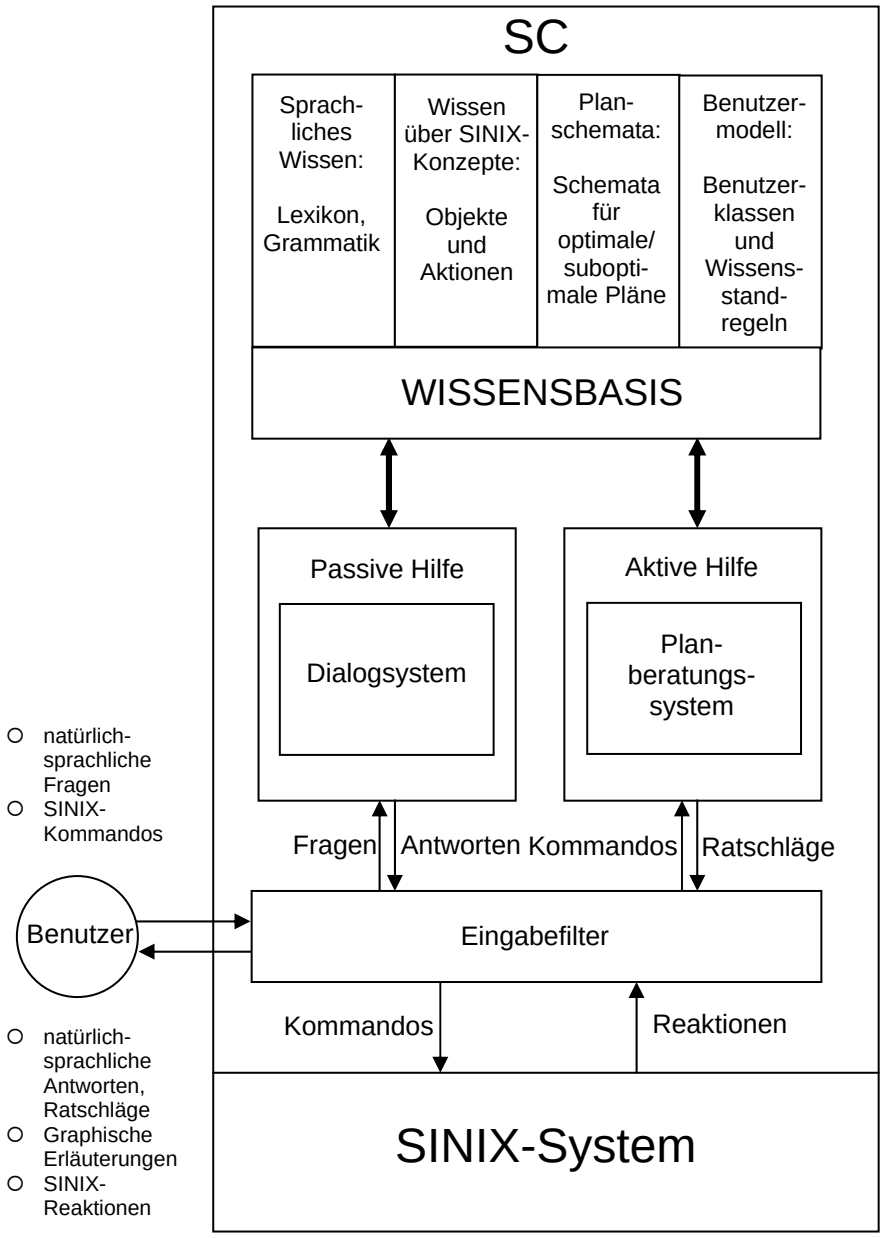
Der heutige Erkenntnisstand im Bereich der Künstlichen Intelligenz (KI) ermöglicht es nicht, die Breite und Tiefe der Hilfeleistungen eines menschlichen Beraters vollständig nachzuahmen. Ein intelligentes Hilfesystem wie SC leistet aber auch dann wertvolle Dienste, wenn es dem Benutzer nicht in allen Situationen weiterhelfen kann. Dem Benutzer bleiben in diesen Fällen aber immer noch die oben genannten konventionellen Informationsquellen.

* *CSnet: <name>%sbsvax.uucp©germany.csnet

Auf dem Gebiet der intelligenten Hilfesysteme wird z. Zt. intensiv geforscht. Die frühesten Arbeiten im Bereich der Hilfesysteme für die UNIX-Domäne wurden an der University of Berkeley im Rahmen des UNIX Consultant (UC) Projektes geleistet (vgl. [WMA*86]). Es folgten zahlreiche weitere Projekte, in denen intelligente Hilfesysteme für UNIX bzw. UNIX-Derivate entwickelt wurden: das AQUA-System (vgl. [QDF86]), das SUSI-System (vgl. [Jer85]), das UCC-System (vgl. [DH82]), das INTERIX-System (vgl. [DGS87]), das Yucca-II-System (vgl. [Heg88]), das OSCON-System (vgl. [MW88]) und das USCSH-System (vgl. [MP88]).

2. Der SINIX Consultant

SC ist ein *wissensbasiertes System,* da sein Verhalten wesentlich von den Einträgen in seiner Wissensbasis bestimmt ist (vgl. Fig. 1).

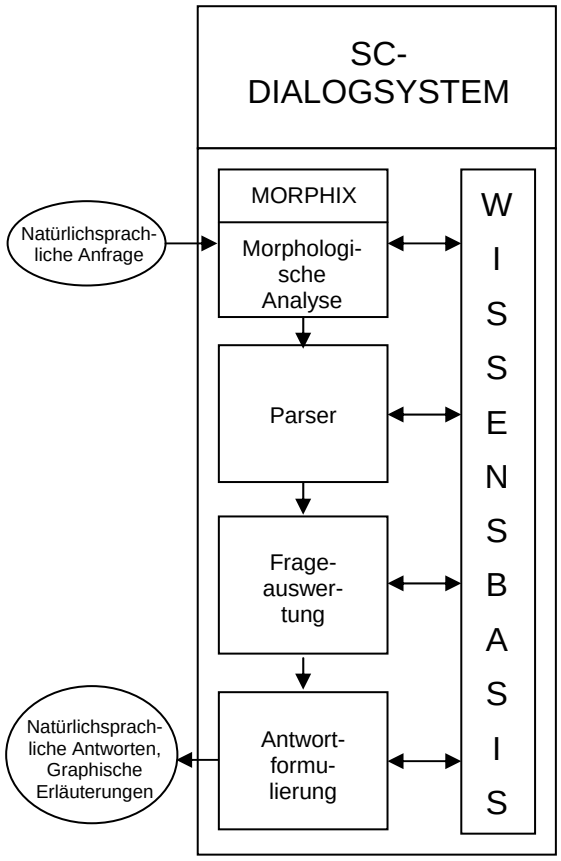


Figur 1: Die Gesamtstruktur des SC-Systems

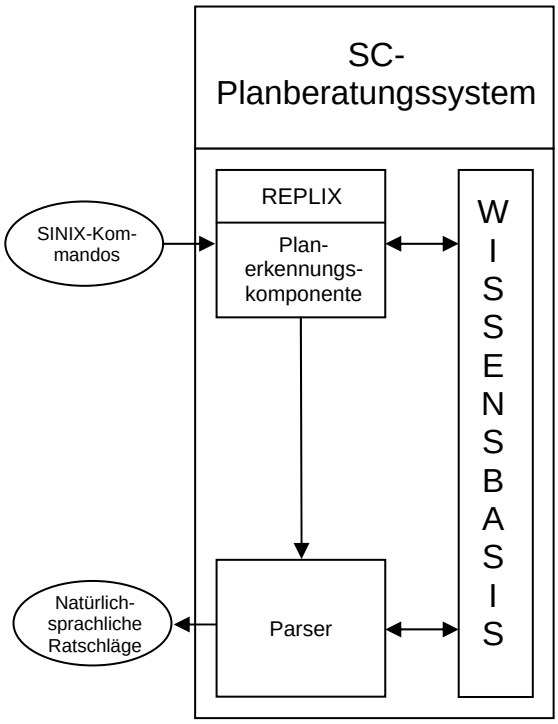
Im Gegensatz zu den bisher üblichen Hilfesystemen ist die beratungsrelevante Information nicht in starren Textdokumenten gespeichert, sondern mithilfe einer Wissensrepräsentationssprache so codiert, daß SC Such- und Inferenzprozesse über den Wissenseinheiten ausführen kann, um ein Beratungsproblem zu lösen und dann seine Ratschläge in einer dem Dialogzustand und dem jeweiligen Benutzer angemessenen Weise zu präsentieren

SC ist ein *natürlichsprachliches System* (vgl. [Wah86]), da Teile seiner Ein- und Ausgaben in deutscher Sprache codiert sind, für die im System Sprachanalyse und -generierungskomponenten existieren (vgl. Fig. 2). Der Zweck eines Hilfesystems wäre verfehlt, wenn der Benutzer wiederum eine neue Kommandosprache lernen müßte, um Hilfeleistungen des Systems in Anspruch nehmen zu können. Um ein 'Hilfesystem für die Bedienung des Hilfesystems' zu umgehen, wird von SC die deutsche Sprache als ein Kommunikationsmittel angeboten, das der Benutzer mühelos beherrscht.

SC ist ein *interaktives System mit gemischter Initiative,* in dem die Dialoginitiative wechselweise vom Benutzer oder vom System ausgehen kann. SC bietet sowohl passive als auch *aktive Hilfe* an. Das Planberatungssystem von SC (vgl. Fig. 3) realisiert eine aktive Hilfekomponente, indem es in Abhängigkeit von Kommandoeingaben des Benutzers unaufgefordert Ratschläge erteilt. Vorbild war hier der SINIX-Kenner, der dem Benutzer 'über die Schulter schaut' und ihm Ratschläge gibt, sobald er feststellt, daß der Benutzer sich fehlerhaft oder ungeschickt verhält, ohne dies selbst sofort zu bemerken. Aktive Hilfesysteme sind auch dann wichtig, wenn der Benutzer so wenig über den Problembereich weiß, daß er gar nicht dazu in der Lage ist, die richtige Frage zum richtigen Zeitpunkt zu stellen. Daneben bietet SC auch eine passive Hilfekomponente, die als natürlichsprachliches Dialogsystem realisiert ist und in der die Initiative vom Benutzer ausgeht (vgl. Fig. 2).



Figur 2: Das SC-Dialogsystem



Figur3: SC- Planberatungssystem

SC ist auch ein *multimodales System*, da es auf der Ausgabeseite mehrere Präsentationsmedien kombiniert. Wie Fig. 1 zeigt, bietet SC neben natürlichsprachlichen Ausgaben auch Graphiken und formatierte SINIX-Reaktionen an. Viele Sachverhalte lassen sich übersichtlicher in Form einer Graphik erläutern und häufig kann ein Verständnis nur durch eine Kombination von Text und Graphik erreicht werden. Derzeit ist SC allerdings noch nicht in der Lage, abhängig vom Dialogkontext und der intendierten Hilfeleistung eine wissensbasierte Präsentationswahl vorzunehmen, sondern die jeweilige Darstellungsart ist für die verschiedenen Reaktionstypen fest vorgegeben.

SC kann auch als kooperatives *Zugangssystem* (vgl. [Wah84]) aufgefaßt werden, da es Fragen des Benutzers nicht nur wörtlich beantwortet, sondern in Form sogenannter *Überbeantwortungen* auch Zusatzinformation in seine Reaktionen aufnimmt, wenn dies aufgrund seines *Benutzermodells* (vgl. Fig. 1) angebracht zu sein scheint. Aufgrund des im Verlaufe der Interaktion aufgebauten individuellen Benutzermodells verfügt SC über Annahmen zum Wissensstand des Benutzers, die es dem System ermöglichen, seine Reaktionen so zu gestalten, daß der Benutzer nicht durch zuviel Information gelangweilt oder durch zu wenig Information überfordert wird. Das 1985 begonnene SC-Projekt hatte von Anfang an zwei Zielrichtungen:

- Die *Grundlagenforschung* im Bereich der intelligenten Hilfesysteme sollte vorangetrieben werden. Durch den vorgegebenen Zeit- und Personalrahmen war von vorneherein eine Konzentration auf wenige Teilfragen wie die Planerkennung, die Repräsentation der Semantik und Pragmatik von Kommandos und die Benutzermodellierung notwendig, um wissenschaftlichen Fortschritt erzielen zu können. Offene Fragen etwa aus dem Bereich der expliziten Modellierung von Zielen und Plänen des Systems, der wissensbasierten Präsentationswahl, der Erkennung von Fehlkonzepten und Mißverständnissen beim Benutzer und der Plangenerierung zur Erfüllung komplexer Benutzerziele mußten ausgeklammert bleiben. Wesentliche Projektergebnisse wurden in einer Reihe von Veröffentlichungen dokumentiert (vgl.[BEG*88], [DGH87], [HKN*88], [Hec88a], [Hec88b], [Hec87], [HH87], [Jun87], [Kem88b], [Kem88a], [Kem87], [Kem86], [Kem85], [Nes87a], [Nes87b]).
- Die *Entwicklung und Implementation* eines einsatzfähigen Funktionsmusters des intelligenten Hilfesystems SC sollte so erfolgen, daß die wesentliche Funktionalität abgedeckt werden kann. Somit konnte nicht in allen Komponenten des Prototypen über den bisherigen Forschungsstand hinausgegangen werden, sondern in mehreren Systemteilen wurden in der KI bewährte Verfahren aufgegriffen und in das Gesamtsystem integriert. Mit dem Abschluß des Projektes stehen Versionen von SC für die Siemens-Rechner MX2, MX500, APS 5815 sowie für VAX-Systeme von DEC zur Verfügung.

Da in den ersten beiden Jahren des Projektes auf den SINIX-Rechnern als den Zielrechnern für SC noch keine geeignete Symbolverarbeitungssoftware zur Verfügung stand, wurde zunächst auf die LISP-Umgebung des Arbeitsplatzrechners APS 5815 zurückgegriffen, die sich bereits in mehreren Kl-Projekten bewährt hatte. In dem parallelen Projekt PORTFIX (vgl. [HSS87]), das am selben Lehrstuhl ebenfalls im Rahmen der KI-Kooperation durchgeführt wurde, konnten die für Projekte wie SC notwendigen Kl-Programmiersprachen und Wissensrepräsentationswerkzeuge erfolgreich auf die SINIX-Rechner portiert werden, so daß im dritten Projektjahr SC auf den Zielrechnern verfügbar gemacht werden konnte.

Im folgenden werden nach einer Erläuterung des Eingabefilters, der globale Kontroll- und Koordinierungsfunktionen hat, die Wissensbasis, das Dialogsystem, das Planberatungssystem und die Benutzermodellierungskomponente genauer beschrieben. Der Bericht schließt mit Hinweisen auf den Implementationsstatus.

3. Der Eingabefilter

Die Entwicklung des SC-Gesamtsystems erfolgte auf einem Siemens APS 5815. Das komplette Softwaresystem wurde anschließend auf einen Siemens MX2 portiert. Die Funktionalität des SC-Kernsystems wurde hierbei vollständig erhalten. Der wesentliche Unterschied der beiden Imple-

mentierungen liegt in der Realisierung des Filter- und Kontrollmoduls. Diese Komponente stellt die Schnittstelle zwischen dem Benutzer, dem SC-Kernsystem und dem SINIX-System dar (vgl. Fig.l).
Der Filter hat daher folgende Aufgaben:

- Einlesen externer Eingaben und Erzeugen externer Ausgaben
- Trennung von natürlichsprachlichen Anfragen und Kommandoeingaben
- Anbindung an das SINIX-System zur Kommandoausführung
- Kontrolle des SC-Verarbeitungsprozesses

Das Filtermodul liest die Eingabe des Benutzers und entscheidet, ob es sich um eine natürlichsprachliche Anfrage oder ein SINIX-Kommando handelt. Im Fall eines natürlichsprachlichen Ausdrucks wird eine Verarbeitung der Eingabe durch die Komponenten des *Dialogsystems* durchgeführt. Kommandoeingaben werden zunächst vom *Planberatungssystem* verarbeitet und anschließend zur Ausführung an das SINIX-System weitergegeben. Vom Filter wird das Endergebnis des Verarbeitungsprozesses ausgegeben und gegebenenfalls Zwischenergebnisse der einzelnen Module, falls das System im Tracemodus betrieben wird, der zu Test- und Vorführzwecken eingerichtet wurde.

Wegen wesentlicher Unterschiede in den Eingabe- und Ausgabemöglichkeiten der beiden Maschinen - Graphik- und Window-Technik des APS 5815 sind auf dem MX2 nicht vorhanden - und des fehlenden Anschlusses an das SINIX-System auf dem APS 5815, das dort innerhalb des Filters emuliert wird, ist diese Komponente für die MX2-Version vollständig neu konzipiert worden (vgl. [BEG*88]).

In der momentanen SC-Implementierung auf dem MX2 werden Kommandos in einer jeweils neu erzeugten Subshell ausgeführt.[1] Diese Lösung führte zu Schwierigkeiten bei Kommandos, die auf Umgebungsvariablen zugreifen, die in der Subshell unterschiedlich zu den entsprechenden Variablen der eigentlichen Shell des Benutzers gesetzt worden sind. So ist z. B. die Wirkung eines cd-Kommandos, das in einer Subshell ausgeführt wurde, nach Verlassen dieser Subshell wieder aufgehoben. Diese Problematik konnte zumindest teilweise durch explizite Simulation der betreffenden Kommandos ohne Starten einer Subshell abgefangen werden. Hierzu wurde auf die in das FranzLisp-System eingebundene Betriebssystem-Schnittstelle zurückgegriffen. Außerdem wurden der Alias- und der History-Mechanismus und partiell auch der Set-Mechanismus nachgebildet. Zielsetzung bei diesem Lösungsweg ist eine Realisierung des Filters in Richtung einer vollständigen Shell.

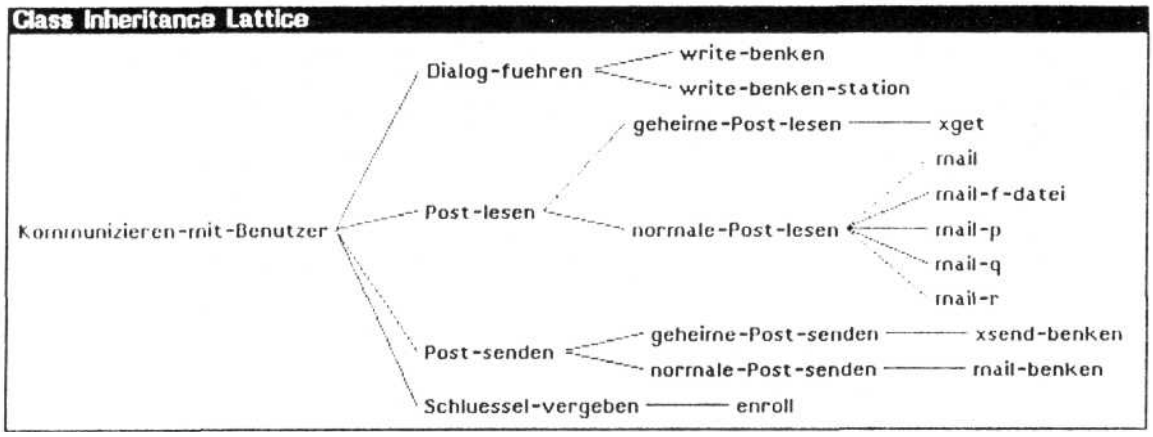
4. Die SINIX-Wissensbasis

Die Aufgabe der SINIX-Wissensbasis ist die Bereitstellung terminologischer Beschreibungen von SINIX-Konzepten in einer deklarativen Form.

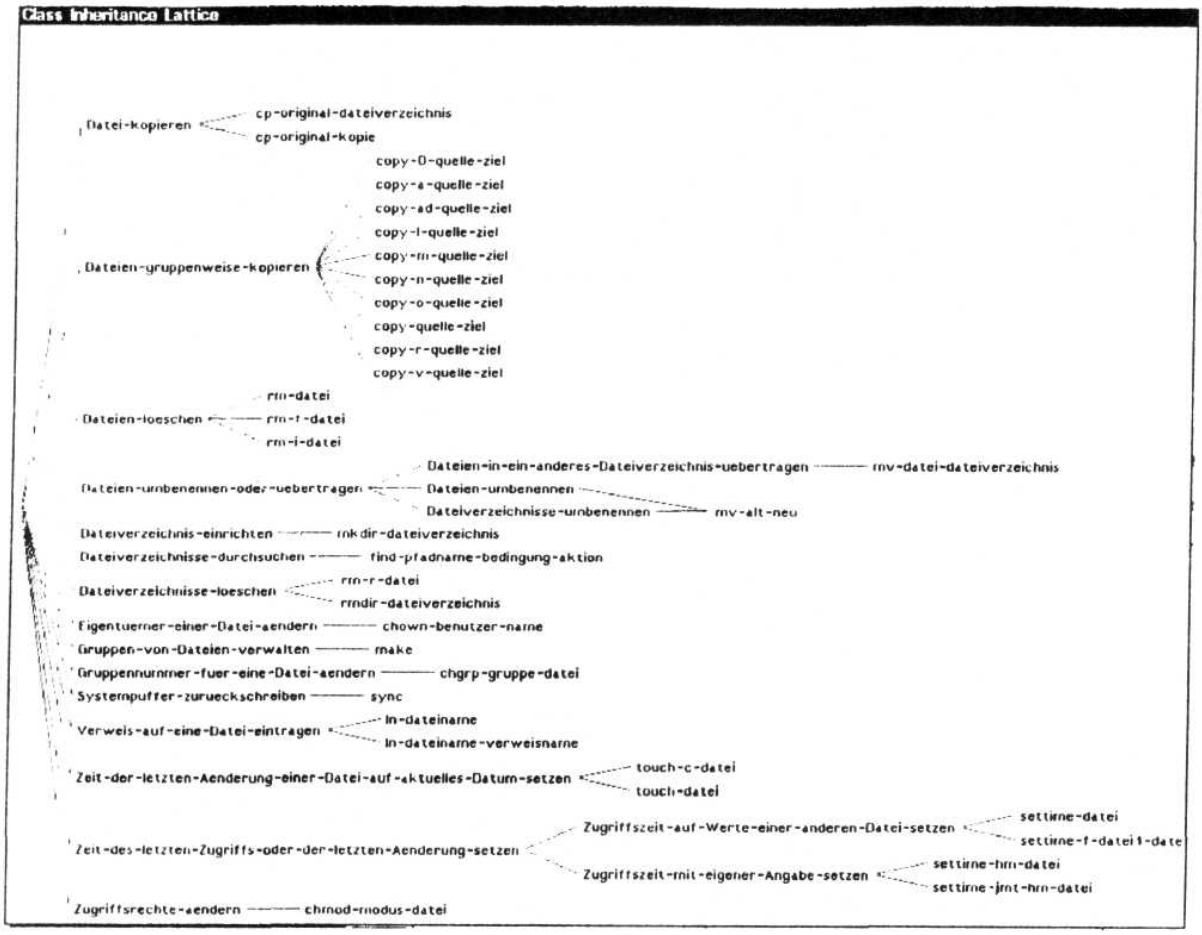
Die SINIX-Wissensbasis ist organisiert als eine taxonomische Hierarchie von Konzepten, die vom Wurzelknoten hin zu den Blattknoten konkreter werden (vgl. Fig. 4 und Fig. 5). Die abstraktesten Konzepte unterhalb des Wurzelkonzepts sind SINIX-Objekt und SINIX-Aktion, die konkretesten Konzepte, also die Blattkonzepte, sind spezifische SINIX-Kommandos, die bestimmt sind durch einen Kommandonamen und ggf. eine Liste von Parametern und einen Schalter -wie `mail -r <Datei>` - oder spezifische SINIX-Objekte - wie z. B. `root-Dateiverzeichnis`, relativ-Pf adname oder Schutzbit. Einzelne Konzepte in dieser Hierarchie sind beschrieben durch Angabe von Attributen, die Eigenschaften des Konzeptes oder Relationen zwischen dem betreffenden Konzept und anderen Konzepten bezeichnen, z. B.

[1] Im Rahmen der Portierung wurden verschiedene Möglichkeiten der Anbindung des SC an das SINIX-System untersucht. Für eine detaillierte Beschreibung vgl. [BEG*88).

das Aenderungsdatum oder der Eigentuemer einer Datei oder der Parameter eines Kommandos. Diese Beschreibungen werden gemäß der hierarchischen Struktur der Wissenbasis von höherliegenden, abstrakteren Konzepten auf untergeordnete, konkretere Konzepte vererbt.



Figur 4: Der Themenbereich kommunizieren-mit-Benutzer



Figur 5: Spezialisierungen der Aktion Dateisystem-organisieren-und-Dateieigenschaften-aendern

Auf der letzten Hierarchiestufe werden diese abstrakten Kommandos zusammengefaßt zu den bereits erwähnten Themen.

Die hierarchische Anordnung von Konzepten in der SINIX-Wissenbasis unterstützt zum einen Suchprozesse, die einer Problemlösung im Fall einer natürlichsprachlichen Benutzerfrage entsprechen (vgl. Frageauswertung); zum anderen ist durch die Einführung 'abstrakter Konzepte', also der Ebenen über den konkreten SINIX-Objekten und -Kommandos, die Möglichkeit gegeben, auch unspezifische Fragen zu beantworten. Solche Fragen werden vom Benutzer aufgrund mangelnden Wissens über die Konzepte des SINIX-Systems gestellt oder sie zielen beabsichtigt darauf, globale übersichtsartige Informationen zu erhalten. Generell reflektieren diese höheren Konzepte mehr oder weniger abstrakte mentale Modelle vom SINIX-System, die sich ebenfalls im Sprachgebrauch niederschlagen, z. B. in dem Ausdruck "Post senden".

Die Betonung bei der Entwicklung der SINIX-Wissensbasis lag auf der Repräsentation von SINIX-Aktionen, da die meisten Fragen des Benutzers die Anwendung von Kommandos zur Durchführung einer bestimmten Aufgabe betreffen.

Es wurden im wesentlichen drei Klassen von Aktionen unterschiedlichen Abstraktionsgrades festgelegt:

- Aktionen auf Themenebene
- abstrakte Kommandos
- konkrete Kommandos

Die Zuordnung von Kommandos zu *Themen* oder *Aufgabenbereichen* wurde direkt aus den SINIX-Handbüchern übernommen; Aktionen auf Themenebene sind z. B. kommunizieren-mit-Benutzer (vgl. Fig. 4), Dateien-verwalten-und-bearbeiten oder Informieren-ueber-Systemdaten.

Die Klassifikation zu abstrakten *Kommandos*, die sich über mehrere Hierarchiestufen erstrecken kann, resultiert aus einer Zusammenfassung von Kommandos mit ähnlicher Funktion. Auf unterster Ebene handelt es sich hierbei um eine Zusammenfassung von Kommandos, die eventuell durch Schalter und zusätzliche optionale Parameter modifiziert sind, zu einem entsprechenden generischen Kommando, das durch einen Kommandonamen und Angabe obligatorischer Parameter festgelegt ist. Ein abstraktes generisches Kommando ist z.B. Dateien-loeschen, das durch den Kommandonamen rm und den obligatorischen Parameter Datei festgelegt ist und das die Kommandos rm-r-datei, rm-f-datei, rm-i-datei und rm-datei subsumiert (vgl. Fig. 5).

Auf den höheren Ebenen werden mehrere abstrakte Kommandos eventuell nochmals zu einer Gruppe zusammengefaßt, falls für diese Gruppe ein spezifischer natürlichsprachlicher Ausdruck existiert, durch den jedes dieser Kommandos referiert werden kann. Beispiele sind das abstrakte Kommando post-senden, das die generischen Kommandos geheime-post-senden (xsend) und normale-post-senden (mail) subsumiert (vgl. Fig. 4), und das abstrakte Kommando Datei-System-organisieren-und-Dateieigenschaften-aendern, das weitere abstrakte Kommandos wie Dateien-umbenennen, Dateiverzeichnis-einrichten oder Zugriffsrechte-organisieren enthält (vgl. Fig. 5). Auf der letzten Hierarchiestufe werden diese abstrakten Kommandos zusammengefaßt zu den bereits erwähnten Themen.

Die Beschreibung einzelner Aktionskonzepte in dieser Hierarchie erfolgt durch eine Reihe von Attributen, die syntaktische, semantische und pragmatische Aspekte der jeweiligen Aktion charakterisieren (s. unten). Ein Teil dieser Attribute ist speziell für die Beschreibung abstrakter Kommandos bzw. Kommandos auf Themenebene vorgesehen, nämlich Standardkommando, wichtige-Kommandos, mögliche-Kommandos und mögliche-Schalter. Andere Attribute sind nur für generische abstrakte Kommandos und konkrete Kommandos definiert, z. B. Kommandoname, aehnliche-Kommandos und zugehoerige-Kommandos.

Die vollständigen Beschreibungen des Themen-Kommandos kommunizieren-mit-

Benutzer, des abstrakten Kommandos Dialog-fuehren und des konkreten Kommandos write-benken-station[2] sehen folgendermaßen aus:[3]

kommunizieren-mit-Benutzer	
Superkonzepte	SINIX-Aktionen
Parameter	<Parameterl>* Benutzerkennung einfache-Datei {<Parameter2>} Datensichtstation
Schalter	{Schalter}
Kommandoaufbau	<Kommandoname> {<Schalter>} <Parameterl>* {<Parameter2>}
Funktionsbeschreibung	unter dem Konzept kommunizieren-mit-Benutzer sind alle SINIX-Kommandos zu finden, die sich in irgendeiner Weise mit der Kommunikation zwischen Systembenutzern beschaeftigen.
moegliche-Kommandos	mail xsend xget write enroll
wichtige-Kommandos	mail write
Standardkommando	mail
Beispielkommando	mail gast

Dialog-fuehren	
Superkonzept	kommunizieren-mit-Benutzer
Kommandoname	write
Parameter	<Empfaenger> Benutzerkennung {<Station>} Datensichtstation
Schalter	nil
Kommandoaufbau	write <Empfaenger> {<Station>}
Funktionsbeschreibung	Nachrichten direkt an einen anderen Benutzer senden.
vorausgesetzte-Konzepte	Der Empfaenger muss aktuell angeschlossen sein.
Beispielkommando	Benutzer
Standardkommando	write gast
zugehoerige-Kommandos	write
aehnliche-Kommandos	mesg who mail
Subkommandos	mail
graphische-Darsteilung	(ENDE-Taste) (CR . CR) BITMAP*...

write-benken-station	
Superkonzept	Dialog-fuehren
Parameter	<Empfaenger> Benutzerkennung <Station> Datensichstation
Kommandoaufbau	write <Empfanger> <Station>
Funktionsbeschreibung	...Nachrichten direkt an einen Benutzer senden. Wird dieselbe Benutzerkennung von mehreren gleichzeitig benutzt, so kann durch Angabe der Datensichtstation der Benutzer eindeutig angesprochen werden.
Beispielkommando	write gast tty09

[2] Die Abkürzung 'benken' steht für 'Benutzerkennung'.
[3] Bei der Angabe von Parametern werden der Parametername und das Konzept, zu dem der Parameter gehört, angegeben.

Die Charakterisierung von Objekten in der SINIX-Wissenbasis erfolgt ebenfalls durch eine Menge von Attributen, die Merkmale des betreffenden Objektes darstellen, wobei diese Merkmale wiederum als Konzepte in der Wissenbasis beschrieben sind. Das spezielle Attribut Aktionen enthält Verweise zu den Kommandos, die auf dieses Objekt anwendbar sind; diese Verbindung zwischen Objekt- und Aktionshierarchie wurde eingerichtet, um die Suche nach einem bestimmten Kommando ausgehend von einem gegebenen Objekt, dessen Zustand verändert werden soll, zu ermöglichen und dadurch Problemlösungs- und Planungsprozesse zu unterstützen. Die Beschreibung des Konzeptes Datei sieht folgendermaßen aus:

Datei

Superkonzept	Dateisystem
Objektbeschreibung	Alles im SINIX-System ist eine Datei. Eine Datei ist eine Folge von Bytes. Das System gibt keine Struktur fuer eine Datei vor und ihrem Inhalt wird keine Bedeutung zugeschrieben. Die Bedeutung der Bytes haengt ausschliesslich von den Programmen ab, die die Datei bearbeiten.
Aktionen	Dateien-verwalten-und-bearbeiten informieren-ueber-Systemdaten
Identilikator	Pfadname Indexnummer
Schutzsystem	Eigentuemer
interne-Repraesent.	Indexnummer
externe-Repraesent.	Dateiname
Aenderungsdatum	Datum
Schutzsystem	Zugriffsschutz
Eigentuemer	Benutzer

Sowohl für Aktionen als auch Objekte sind sogenannte *Funktions-* bzw. *Objektbeschreibungen* angelegt, die kurz die Bedeutung des betreffenden Konzeptes umreißen, also Funktion, Struktur und Zweck eines Objektes oder Kommandos. Für Aktionen soll außerdem eine formalisierte Darstellung ihrer Funktion entwickelt werden. Grundlagen zu einer formalen Repräsentation der Semantik von (SINIX-) Aktionen wurden bereits geschaffen [Kem88a] [Kem88b], eine Implementation erfolgte jedoch noch nicht. Basis für diese formale Semantik-Beschreibung ist eine relativ umfassende Klassifikation von SINIX-Kommandos [Kem87] [Kem88b].

Die SINIX-Wissensbasis umfaßt derzeit ca. 360 Konzepte, davon knapp 300 im Aktionsteil. Die Menge der SINIX-Kommandos ist mit Ausnahme spezieller Systemverwalter-Kommandos vollständig erfaßt. Es sind ca. 180 konkrete SINIX-Kommandos, d.h. durch Kommandoname, Parameter und Schalter festgelegte Kommandos, spezifiziert, von denen ungefähr 130 vollständig beschrieben sind.

5. Das Dialogsystem

5.1. Der Parser

Die passive Hilfe des SC wird über das Dialogsystem realisiert (vgl. Fig. 1). Der Benutzer kann Fragen, die sich auf Konzepte (z. B. `Dateiverzeichnis`, `Post`) oder Kommandos (z.B. `mkdir`, `mail`) des SINIX-Systems beziehen, in deutscher Sprache eingeben. Die *natürlichsprachlichen Anfragen,* die an ein System wie SC gestellt werden, stellen nur einen Ausschnitt des deutschen Sprachumfangs dar. Die Fragen tauchen zumeist in W-Form auf, z. B. "Wie kann ich eine Datei löschen?" oder "Was ist ein Directory?". Darüber hinaus tauchen

Standardphrasen wie "Ist es möglich ...", "Mit welchen Kommandos kann ich ..." oder "Ich möchte ..." auf. Diese Fragen und Phrasen werden von SC korrekt verarbeitet.

Außer wohlgeformten natürlichsprachlichen Ausdrücken werden auch elliptische Sätze verwendet, z. B. "Syntax von ls!". Obwohl diese Anfragen keine vollständigen Sätze darstellen, muß die natürlichsprachliche Schnittstelle sie verarbeiten können, um den Benutzer nicht zu stark in seinem Sprachverhalten einzuschränken. Die Analyse der Anfragen gliedert sich in drei Phasen (vgl. Fig. 2):

1. die morphologische Analyse,
2. das Parsing,
3. die Instantiierung des Kasusrahmens.

Diese Verarbeitungsschritte werden nachfolgend beschrieben.

Die eingegebenen Fragen werden zuerst einer *morphologischen Analyse* unterzogen, in der syntaktische Merkmale (z. B. Genus, Numerus) und der Wortstamm der Wörter bestimmt werden (vgl. [FN86]). Die morphologische Analysekomponente MORPHIX arbeitet mit einem auf die SINIX-Terminologie zugeschnittenen Lexikon, das ca. 2000 Wörter umfaßt.

Nach der morphologischen Analyse wird eine Bedeutungsrepräsentation der Anfrage erstellt. Hierzu bildet der *Parser* mit Hilfe von Grammatikregeln die Anfrage auf *Kasusrahmen* (vgl. [Fil68]) ab. Die zugrundeliegende Grammatik verwendet zwei Typen von Regeln:

- Oberflächenregeln, die die zu erkennenden Satzmuster repräsentieren, und
- Ersetzungsregeln, die die Abbildung terminaler Symbole der Grammatik auf nichtterminale realisieren.

Die Anfrage "Ist es möglich geheime Post einem anderen Benutzer zu schicken?" wird mit Hilfe der folgenden Oberflächenregel verarbeitet:

(< *QueationBegin* > to (verb=< Send >) < *SendAdjective* > (object=< *SendObject* >)*
(address=< *AnotherUser* >) ?)

In < ... > eingeschlossene Namen bezeichnen Nichtterminale. Der String '(verb=< *Send* >)' drückt aus, daß das Wort, das auf das Nichtterminal < *Send* > abgebildet werden kann, an die Variable 'verb' gebunden wird. Das Symbol '*' steht für eine beliebigen Anzahl von Wörtern. Die Strings 'to' und '?' sind terminale Symbole. Zu jeder Oberflächenregel gehört eine Routine zur Instantiierung des Kasusrahmens.

Kann ein Eingabesatz erfolgreich auf eine entsprechende Oberflächenregel abgebildet werden, wird unter Ausnutzung der abgespeicherten Wortstämme der zugehörige Kasusrahmen instantiiert.

Im Fall des obigen Satzes, der erfolgreich vom Parser verarbeitet werden kann, wird der nachfolgende Kasusrahmen kreiert.

```
( sentence    ( Content "Ist es möglich ...")
              ( Illocution    SyntacticQuestion
              ( Verb          ( send (agent    user)
                                     (object   (mail     secrete)
                                     (address  (user     another)))))
```

Außer der wörtlichen Wiedergabe des Inhalts nach dem Schlüsselwort 'Content' spielt die 'lllocution' eine wichtige Rolle. Sie bestimmt, um was für eine Art von Anfrage es sich handelt. 'SyntacticQuestion' ist der Marker für eine Frage nach der Realisierung einer Aktion durch ein Kommando (hier: Verschicken von Post). Über andere Illokutionen kann z. B. nach der Bedeutung eines bestimmten Objektes (z. B.: "Was ist ein Directory?") oder nach der Eigenschaft eines Objektes oder Kommandos (z. B.: "Welche Schalter hat ls?") gefragt werden.

Die verwendete Grammatik, alle Kasusrahmen und alle möglichen Illokutionen sind in [HKN*88] vollständig beschrieben.

Der vom den Parser aufgebaute Kasusrahmen wird an die *Frageauswertung* weitergereicht.

5.2. Die Frageauswertung

Die vom *Parser* erzeugte Repräsentation des Eingabesatzes, der instantiierte Kasusrahmen, wird von der *Frageauswertung* (Question Evaluator) weiterverarbeitet.

Die Aufgabe der Frageauswertung besteht darin, das Konzept der SINIX-Wissensbasis zu bestimmen, auf das sich die Anfrage des Benutzers bezieht. In dieser Komponente erfolgt also der wesentliche Teil des eigentlichen Problemlösungsprozesses. Bei der Bestimmung des gesuchten Konzeptes, das anschließend von der *Antwortformulierung* weiterverwendet wird, wird zunächst in Abhängigkeit von der 'Illocution', dem Typ der Anfrage, ein entsprechender Subprozeß der Frageauswertung angestoßen. Im wesentlichen können zwei Arten von Fragen unterschieden werden:

- Fragen, bei denen das SINIX-Konzept, über das der Benutzer Informationen möchte, bereits in der Frage angegeben ist, und
- Fragen, in denen das Konzept umschrieben wird.

Zur ersten Kategorie gehören z. B. 'ObjectQuestions', die Fragen nach der Erklärung eines Konzeptes bezeichnen wie "Was ist ein Dateiverzeichnis?" oder "Was macht das Kommando f g r e p ?", und 'A t t r i b u t e V a l u e Q u e s t i o n s', die Fragen nach einem speziellen Attributwert eines gegebenen Konzeptes bezeichnen, z. B. die Frage nach Schaltern eines Kommandos oder seiner Syntax. Probleme treten bei diesen Fragen dann auf, wenn der Benutzer aufgrund einer Fehlannahme Konzepte referiert, die nicht existieren, z. B. in der Frage "Was macht xsend -k?". In solchen Fällen wird in der aktuellen SC-Version als Lösung des Frageauswertungsprozes9es das speziellste Konzept der SINIX-Wissensbasis angegeben, das mit der Anfrage konform ist. In dem genannten Beispiel wäre dieses das Konzept g e h e i m e - P o s t - v e r s c h i c k e n, also das generische Kommando x s e n d.

Ein eigentlicher Suchprozeß setzt erst bei der zweiten Fragekategorie ein, in der das gesuchte Konzept nicht direkt referiert wird, sondern durch Angabe von Merkmalen spezifiziert ist. Diese Fragen beziehen sich auf die Durchführung von Aktionen, also letztendlich auf SINIX-Kommandos, und beinhalten eine natürlichsprachliche Umschreibung einer Aktion, die der Benutzer mithilfe des SINIX-Systems ausführen möchte. Ein Beispiel ist die Frage "Wie entferne ich ein Directory?". Der Problemlösungsprozeß orientiert sich in diesen Fällen zunächst am Wert des Kasus 'verb' in dem vom Parser aufgebauten Kasusrahmen, also am Repräsentanten der Verbklasse, auf den das Verb des Eingabesatzes, das die Aktion bezeichnet, abgebildet wird. In obigem Beispiel ist das Verb des Eingabesatzes 'entfernen' und der zugehörige Verbklassenrepräsentant und Wert des Kasus 'verb' wäre 'löschen'. Die Frageauswertung verfügt zu jedem Verbklassenrepräsentanten über eine Tabelle, mittels derer in Abhängigkeit von zusätzlichen Konstituenten des Eingabesatzes, wie Objekte und Attribute, das betreffende Konzept der SINIX-Wissensbasis bestimmt wird. In dem Beispiel "Wie entferne ich ein Directory?" würde in der Repräsentation des Eingabesatzes zusätzlich zum Verb 'löschen' noch das Objekt 'Dateiverzeichnis' vorkommen. Das Objekt 'Dateiverzeichnis' könnte wiederum durch Attribute näher spezifiziert sein, z. B. durch die Adjektive 'leer' oder 'voll'. Die Frageauswertung sucht zunächst nach dem passenden Verb, d.h. Einträgen für 'löschen', und versucht dann, das vorkommende Objekt und ggf. sein Attribut auf einen Objekteintrag der zum Verb 'löschen' gehörenden Konzepte abzubilden. In dem oben beschriebenen Fall würde die Frageauswertung das Konzept D a t e i v e r z e i c h n i s - l o e s c h e n liefern.

Wesentlich für den Problemlösungsprozeß sind zum einen die Vorarbeiten, die durch den Parser durchgeführt werden (Abbildung auf Repräsentanten von Wortklassen), und zum anderen die hierarchische Strukturierung der Wissensbasis, die einen Zugriff auch für abstraktere Objekte und Aktionen ermöglicht, die nicht eine direkte eins-zu-eins-Entsprechung in SINIX-Objekten und -Kommandos haben.

Das von der Frageauswertung gefundene Wissensbasis-Konzept und die Repräsentation des Eingabesatzes werden an die *Antwortformulierung* übergeben, die zuständig für die Konstruktion einer natürlichsprachlichen Antwort auf die Benutzerfrage ist.

5.3. Die Antwort Formulierung

Die Aufgabe der *Antwortformulierung* ist die Erzeugung einer an den individuellen Benutzer angepaßten Antwort auf eine von ihm gestellte Frage, d.h. einer Antwort, die seinen individuellen Kenntnisstand bezüglich des SINIX-Systems berücksichtigt.

Die Generierung natürlichsprachlicher Ausgabesätze basiert auf einem einfachen Ausfüllen vorgegebener Satzmuster mit entsprechenden Inhalten der Wissensbasis, nämlich Attributwerten des betreffenden Konzeptes, das in der Antwort erläutert werden soll. Zu diesem Zweck ist jedem Attribut, das in einer Konzeptbeschreibung vorkommen kann, ein bestimmtes Satzmuster zugeordnet. Zum Beispiel ist dem Attribut Kommandoaufbau das Satzmuster "Die Syntax ist <K o n u n a n d o a u f b a u >" zugeordnet.

Die Hauptaufgabe der Antwortformulierung liegt darin, zu bestimmen, welche Attribute des Konzeptes in der Antwort verbalisiert werden sollen. Diese Auswahl erfolgt in Abhängigkeit vom *Konzept der Wissensbasis,* das die Frageauswertung bestimmt hat, dem Fragetyp, d.h der vom Parser festgestellten Illokution des Satzes, und dem *Wissensstand* des Benutzers bezüglich des SINIX-Systems, über den die Benutzermodellierungskomponente Auskunft gibt (s. Abschnitt 7). Die Liste der ausgewählten, zu verbalisierenden Attribute wird in einem sogenannten *Antwort-Rahmen* festgehalten, der anschließend mit den entsprechenden Attributwerten des Konzeptes aus der Wissenbasis aufgefüllt wird.

Die Entscheidung über die Menge der zu verbalisierenden Attribute basiert zunächst auf einer Klassifizierung von Antworten anhand

- des Konzepttyps, der aus der Position des Konzeptes in der Hierarchie der SINIX-Wissensbasis abgelesen werden kann und anhand dessen bestimmt werden kann, welche Attribute überhaupt für das Konzept definiert sind, und
- des Fragetyps, also der vom Parser gelieferten 'Illocution' des Eingabesatzes.

Die Frage "Wie verschicke ich verschlüsselte Nachrichten?" würde zum Beispiel abgebildet auf das Konzept g e h e i m e - p o s t - v e r s c h i c k e n , die Illokution wäre 'SyntacticQuestion', also eine Frage nach einem spezifischen SINIX-Kommando. Das Konzept g e h e i m e - p o s t - v e r s c h i c k e n ist innerhalb der SINIX-Wissensbasis ein abstraktes Kommando mit genau einem zugehörigen konkreten SINIX-Kommando, durch das es realisiert wird; der Antworttyp in diesem Fall ist *abstract-command-unique-name-unique-successor*. Bei abstrakten Kommandos, die i.a. mehrere konkrete SINIX-Kommandos in einer Gruppe zusammenfassen, gibt es eine Unterscheidung zwischen solchen, die Kommandos mit verschiedenen Kommandonamen subsumieren *(abstract-command-different-names),* und solchen, zu denen mehrere Kommandos mit demselben Kommandonamen aber verschiedenen Schaltern gehören *(abstract-command-unique-names-different-options).* Außerdem werden noch Antworten auf Themenebene behandelt, d.h. Antworten auf sehr unspezifische Fragen wie "Wie kann ich mit einem anderen Benutzer kommunizieren?" oder "Wie kann ich Dateien bearbeiten?" und Anworten, die ein konkretes SINIX-Kommando behandeln. In diesem letzten Fall wird noch anhand des Fragetyps unterschieden, ob in der Frage das Kommando selbst gesucht ist (SyntacticQuestion) oder eine Erklärung eines angegebenen Kommandos gefordert wird (ObjectQuestion).

Aus dem Satz möglicher Attribute zur Verbalisierung, der durch den Antworttyp bestimmt ist, werden nun unter Berücksichtigung des Benutzermodells diejenigen Attribute ausgewählt, die in der Antwortformulierung Verwendung finden sollen. Ausschlaggebend hierbei ist in erster Linie die Zuordnung des Benutzers zu einem der vier Prototypen (1=Experte,...,4=Neuling) und in zweiter Linie das spezifische Kommandowissen des Benutzers, das im individuellen Benutzermodell festgehalten ist. Die generelle Strategie der Antwortformulierung ist, einem Neuling möglichst wenig neue Information zu geben, aber die Erklärung möglichst konkret und verständlich zu halten, indem Beispiele gegeben und Bezüge zu bereits bekannten Konzepten aufgezeigt werden, z.B. Verweis auf ein Kommando mit ähnlicher Funktion. Im Fall eines erklärende Teil der Antwort möglichst knapp und prägnant ausfallen, dafür wird jedoch versucht, seine Kenntnisse bezüglich des SINIX-Systems zu erweitern, indem er auf ihm unbekannte Kommandos, die in enger Beziehung zu dem von ihm erfragten Kommando stehen, explizit hingewiesen wird.

Um dem Benutzer eine Kontrolle darüber zu ermöglichen, ob das System seine Anfrage richtig interpretiert hat, wird vor dem eigentlichen Erklärungstext eine Verbalisierung der vom Parser erzeugten Repräsentation des Eingabesatzes durchgeführt. Ein solcher *Klärungssatz* hat die Form[4]

"Wenn ich Ihre Frage richtig interpretiere, möchten Sie < *attribut* >< *objekt* >< *verb* >."

Im nachfolgenden werden einige Beispiele gegeben, die die Antwortformulierung illustrieren sollen[5] Die Anwort auf die Frage

"Ich moechte Meldungen von anderen Benutzern am Bildschirm erlauben."

wäre im Fall eines Neulings

"Soweit ich Sie verstanden habe, moechten Sie *meldung erlauben.* Sie verwenden hierzu das Kommando mesg. Die Syntax ist mesg y. Um das Kommando richtig anwenden zu koennen, sollten Ihnen die Konzepte: meldung bekannt sein."

Zu bemerken ist hierbei, daß der Benutzer explizit auf Konzepte hingewiesen wird, die ihm bekannt sein sollten, die er aber laut Eintrag im Benutzermodell nicht kennt. Es wäre ebenfalls möglich, dem Benutzer diese Konzepte unaufgefordert zu erklären oder ihm eine Erklärung anzubieten. Ein weiteres Beispiel ist die Frage

"Wie kann ich verschluesselte Nachrichten schicken."

auf die die Antwort

"Wenn ich Sie richtig verstanden habe, moechten Sie *verschluessel nachricht schicken.* Sie verwenden hierzu das Kommando xsend. Die Syntax ist xsend <Benutzerkennung>. Z.B. xsend gast. Kommandos, die aehnlich funktionieren, sind: mail."

gegeben wird. Hier wird zusätzlich zur Nennung des erfragten Kommandos xsend zum Schicken von geheimer Post noch auf das mail-Kommando hingewiesen, daß die gleiche Funktion für Standardpost erfüllt, falls dem Benutzer laut Eintrag im Benutzermodell das mail-Kommando bekannt ist.
Ein fortgeschrittener Benutzer wird im ersten Fall die Antwort

"Soweit ich Sie verstanden habe, moechten Sie *meldung erlauben.* Sie verwenden hierzu das Kommando mesg. Die Syntax ist mesg y."

erhalten. Die Antwort auf die zweite Frage wäre:

"Wenn ich Sie richtig verstanden habe, moechten Sie *verschluesael nachricht schicken.* Sie verwenden hierzu das Kommando xsend. Die Syntax ist xsend <Benutzerkennung>. Andere Kommandos in diesem Bereich sind: xget, enroll, write."

Der Benutzer wird in der zweiten Antwort nur auf solche Kommandos hingewiesen, die ihm laut Benutzermodell vermutlich unbekannt sind.
Eine Übersichtsfrage zu einem Thema ist z.B.
"Wie kann ich mit einem anderen Benutzer kommunizieren?"

[4]Das System unterscheidet in den internen Repräsentationen keine Groß-/Kleinschreibung und verfügt nicht über eine Flektionskomponente, was öfters zu holprigen, wenn auch verständlichen Ausgabesätzen führt.
[5]Ausdrücke, die der Repräsentation des Eingabesatzes entstammen, sind kursiv gedruckt; Einträge aus der Wissensbasis sind in Typewriter-Schrift.

Die Antwort auf diese Frage ist

> "Sie moechten *benutzer kommunizieren.* Die Syntax ist <Kommandonane> {<Schalter>} <Parameterl>* <Parameter2>. Das Standardkommando in diesem Bereich ist mail. Z.B. mail gast."

im Fall eines Neulings; im Fall eines fortgeschrittenen Benutzers wird ein Überblick über alle verfügbaren Kommandos gegeben, die die Kommunikation zwischen Benutzern betreffen:

> "Moegliche Kommandos in diesem Bereich sind: mail, write, xsend, xget, enroll."

6. Das Planberatungssystem

Außer der Fähigkeit, auf natürlichsprachliche Anfragen zu antworten, ist SC in der Lage, von sich aus aktiv zu werden und dem Benutzer individuell angepaßte Hilfeinformationen zu geben (vgl. Fig. 1). Um diese aktive Fähigkeit zu realisieren, besitzt das System ein *Planberatungssub-system* (vgl. Fig. 3). Dieses Subsystem besteht aus der *Planerkennungskomponente REPLIX* und einer *Plankorrekturkomponente* (vgl. [DGH87], [Hec87] und [HKN*88]). Gibt der Benutzer die beiden Kommandos:

```
mv Brief 1 Rechnungen
mv Brief2 Rechnungen
```

ein, erkennt SC, daß es sich um eine suboptimale Befehlssequenz handelt, und schlägt einen besseren Plan vor:

```
Sie können das gleiche Ziel erreichen durch
mv Brief 1 Brief2 Rechnungen
```

Hierzu werden die vom Benutzer verwendeten SINIX-Kommandos an REPLDC weitergereicht. Ausgehend von vorgegebenen Planschemata versucht REPLIX, die eingegebenen Kommandos auf einen oder mehrere Pläne abzubilden. Wurde ein Plan erfolgreich erkannt, wird angenommen, daß der Benutzer das mit dem Plan assoziierte Ziel verfolgt. Ist zur Erreichung des Ziels ein besserer Plan vorhanden, d.h. ein Plan mit weniger oder effizienteren Kommandos, so wird dieser dem Benutzer vorgeschlagen. Die Präsentation des neuen Plans erfolgt angepaßt an den Wissensstand des jeweiligen Benutzers. Die Bestimmung des besseren Plans und die Anpassung der Hilfeinformation an das Benutzerwissen erfolgt durch die *Plankorrekturkomponente.* Sie verwendet hierzu Annahmen über das SINIX-Wissen des Benutzers, die im Benutzermodell enthalten sind, und Wissen über das SINIX-System, das von der SINIX-Wissensbasis bereitgestellt wird. Zu beachten ist, daß die Planschemata abstrakte Pläne sind und nicht nur eine eins-zu-eins Abbildung durchgeführt wird, sondern auch eingeschobene Sub-Pläne und überlappende Pläne erkannt werden, über Ignore-Kommandos nicht zu den Plänen gehörige Kommandos ignoriert werden können und zur Fokussierung der Planerkennung Interrupt-Kommandos berücksichtigt werden. Informationen über den internen Zustand des Planerkennungsprozesses können in der APS 5815-Version graphisch präsentiert werden.

Die Planerkennungskomponente ist so realisiert worden, daß sie an andere Betriebssysteme angepaßt werden kann. Dies wurde über die Definition einer Kommando-Syntax erreicht, die zwischen Kommando-Wort, Flags und Parametern unterscheidet. Darüber hinaus ist es möglich, Wildcards und spezielle Zeichen (z. B. Anfangszeichen der Flags) anzugeben. Der Aufbau einzelner Planschemata - die in verschiedene Planpakete zusammengefaßt werden können - ist wie folgt:

[(NameOfPlan (GoalString)
(CmdWord$_1$ Flags$_1$ ObjectList$_1$)
(CmdWord$_2$ Flags$_2$ ObjectList$_2$)
…
(CmdWord$_n$ Flags$_n$ ObjectList$_n$)
(ListOflgnoreCommands)
(ListOflnterruptCommands)

Jeder Plan hat einen eindeutigen Namen (NameOfPlan). Als Ziel (GoalString) kann ein beliebiger String angegeben werden. Über diesen Ziel-String findet die Anbindung von REPLLX an nachgeordnete Komponenten statt, die das Ergebnis des Planerkennungsprozesses verwenden. Jedes Kommando wird beschrieben durch:

- ein Kommandowort (z.B. m v oder m k d i r),
- Schalter (z. B. - l a oder - f) und
- eine Liste von Parametern (z. B. N a m e d e r D a t e i).

Drei verschiedene Typen von Parametern sind vorhanden:

- fixer Parameter, d.h. das zugehörige Kommando wird nur erfolgreich erkannt, wenn es mit diesem Parameter verwendet wird.
- Scnemaparameier (beginnt mit ';')> d.h. das Kommando kann mit beliebigen aktuellen Parametern verwendet werden. Über diesen Parametertyp kann festgelegt werden, daß Parameter verschiedener Kommandos innerhalb eines Plans übereinstimmen müssen.
- Mengenpasasneter (beginnt mit einem '!'), d.h. eine Liste aktueller Parameter kann spezifiziert werden.

Beispiel: Das folgende Planschema, das aus vier Kommandos besteht, verfolgt das Ziel "delete a directory together with its content":

(1) cd ;dir
(2) rm *
(3) cd ..
(4) rmdir ;dir

Durch das erste Kommando wird in das angegebene Dateiverzeichnis gewechselt. Der verwendete aktuelle Parameter wird in der Schemavariablen ';dir' abgespeichert. Alle Dateien des angegebenen Dateiverzeichnisses werden gelöscht (Kommando 2) und es wird zum übergeordneten Dateiverzeichnis zurückgewechselt (Kommando 3). Durch das letzte Kommando wird das leere Dateiverzeichnis gelöscht.

In Kommando 1 und 4 ist derselbe Schemaparameter angegeben. Der Plan wird nur dann erfolgreich erkannt, wenn in beiden Kommandos derselbe aktuelle Parameter verwendet wird. Verwendet der Benutzer folgende Kommandosequenz:

(1) c d R e c h n u n g e n
(2) r m *
(3) c d . .
(4) r m d i r

erkennt REPLIX den zugehörigen Plan und liefert die folgenden Informationen an die Plankorrekturkomponente:

(*4* (DeleteDir (delete a directory and its content))
((cd Rechnungen) (;dir (Rechnungen)))
((rm *) NIL)
((cd .. NIL)
((rmdir Rechnungen) (;dir (Rechnungen))))

Nach dem vierten Kommando wurde die Vervollständigung des Plans mit Namen 'DeleteDir' erkannt. Außer dem Plannamen und dem Ziel werden die Kommandos mit den verwendeten aktuellen Parametern an die Plankorrekturkomponente geliefert.

REPLIX ist nicht nur in der Lage, einfache Pläne einzeln oder in Folge zu erkennen, sondern kann auch Planunterbrechungen und eingeschobene Sub-Pläne erkennen. Angenommen, der Benutzer verwendet die Kommandosequenz:

```
(1)   cd Rechnungen
(2)   lpr meyer
(3)   lpr neumann
(4)   rm *
(5)   cd . .
(6)   rmdir
```

Der oben genannte Plan wird nach dem ersten Kommando unterbrochen und ein anderer Plan, bestehend aus den Kommandos lpr meyer und lpr neumann, wird eingeschoben. Mit dem vierten Kommando wird der erste Plan wieder aufgenommen. Diese Einbettung wird erkannt und wie folgt an die Plankorrekturkomponente gemeldet:

```
(*3*   (PrintFiles (print two files)) ...)
(*6*   (DeleteDir (delete a directory and its content)) ...)
(*6*   (INSERTED (PrintFiles IN DeleteDir) 1 ))
```

Der vollständige Gebrauch beider Pläne und die Interaktion zwischen den Plänen wird gemeldet. Über den *branch counter* ist es möglich, die maximale Anzahl geschachtelter Einbettungen, die noch vom Planerkennungsprozeß akzeptiert wird, einzustellen.

Neben eingeschobenen Sub-Plänen ist REPLDC auch in der Lage, *Überlappungen* zwischen Plänen zu erkennen. Endet ein verwendeter Plan mit Kommandos, die zugleich Anfang eines nachfolgenden Plans sind, so meldet der Planerkenner zusätzlich zum Erkennen der Einzelpläne die Überlappung und die daran beteiligten Kommandos.

Da nicht jedes Kommando auf einen Plan abgebildet werden kann, muß ein Mechanismus vorhanden sein, der den Abbruch des Erkennungsprozesses bei solchen Kommandos verhindert. Das Konzept der Ignore-Kommajidos realisiert dies. Hierbei kann zu jedem Plan eine Liste von Ignore-Kommandos angegeben werden. Beispiele für solche Kommandos sind date, pwd oder ps -a. Ob ein Kommando zu ignorieren ist oder nicht, hängt von der Verwendung der Planerkennung ab. Über den *Ignore-Counter* kann spezifiziert werden, wie oft die Ignorierung durchgeführt werden soll, bevor ein Abbruch der Planerkennung erfolgt.

Das Konzept der *Interrupt-Kommandos* wird zur Fokussierung der Planerkennung verwendet. Der Planerkennungsprozeß wird abgebrochen, sobald ein Kommando der Liste der Interrupt-Kommandos auftritt. So kann z. B. bei der Verwendung des Kommandos cc (Aufruf des C-Compilers) die Erkennung der Pläne abgebrochen werden, die den Bereich der Datei-Verwaltung abdecken.

Die Planerkennungskomponente ist in der Lage, beliebige Kombinationen aus Einbettung und Überlappung mit Ignore- und Interrupt-Kommandos zu verarbeiten.

Die vollständige Beschreibung der Planerkennungskomponente REPLIX findet sich in [DGH87]. Zusätzlich zur Planerkennungskomponente wurde für die APS 5815-Version ein Plan-Editor realisiert, der den Aufbau der Planbibliothek unterstützt. Eine umfassende Liste der verwendeten suboptimalen und optimalen Pläne sowie eine ausführliche Darstellung der Plankorrekturkomponente finden sich in [HKN*88].

7.Die Benutzermodellierungskomponente

Ein Benutzermodell ist derjenige Teilbereich der Wissensbasis eines Kl-Systems, in dem explizite Annahmen über alle Aspekte des Benutzers enthalten sind, die für das Dialogverhalten des Systems relevant werden können (vgl. [WK86]). Eine Benutzermodellierungskomponente hat die Aufgabe:

- inkrementell ein individuelles Benutzermodell aufzubauen,
- Einträge im Benutzermodell zu speichern, zu löschen oder zu ändern,
- die Konsistenz des Modells zu erhalten und
- andere Systemkomponenten mit Annahmen über den Benutzer zu versorgen.

Die Benutzermodellierungskomponente SC-UM (vgl. [Nes87a]) kann auf Standardannahmen über Benutzer zurückgreifen, die in Prototypen für vier Benutzerklassen zusammengefaßt sind. In bezug auf SINIX-Kenntnisse werden die Benutzer von SC grob in die Klassen Neulinge, Anfänger, Fortgeschrittene und Experten eingeteilt. Das typische Wissen von Benutzern aus diesen Klassen wurde durch eine empirische Untersuchung ermittelt (vgl. [Nes87a]). Jeder Prototyp besteht aus einer Menge von Verweisen auf Wissen über bestimmte Kommando- und Objektnamen, die den Wissenstand eines Benutzers der entsprechenden Klasse charakterisieren. Von allen nicht im Prototyp aufgeführten Wissenselementen nimmt SC-UM solange an, daß ein Benutzer dieser Klasse sie nicht kennt, bis im individuellen Benutzermodell ein Eintrag erfolgt, der dieser Annahme widerspricht. So wird z. B. von einem Fortgeschrittenen im Gegensatz zu einem Anfänger angenommen, daß er das Konzept `Schalter` und das Kommando `mkdir` kennt, während Wissen über das Kommando `xget` nur bei SINIX-Experten vermutet wird.

Während der Interaktionen mit SC baut SC-UM inkrementell ein individuelles Benutzermodell auf. Die Annahmen in diesem Modell werden aus den Fragen und Kommandoeingaben des Benutzers sowie den Ratschlägen und Antworten von SC abgeleitet. Durch neu abgeleitete Annahmen im individuellen Benutzermodell kann die Zuordnung des Benutzers zu einer Klasse revidiert werden. Durch die Verwendung der Prototypen verfügt SC bereits nach wenigen Interaktionsschritten über ein grobes Benutzermodell, das im Verlauf der Dialogsitzung korrigiert oder immer weiter verfeinert werden kann.

Neben den Annahmen, die sich direkt aus der Beobachtung des Benutzerverhaltens ergeben, leitet SC-UM mit Hilfe einer Menge von Wissensstandregeln weitere Einträge für das individuelle Benutzermodell ab. Diese Inferenzkomponente von SC-UM geht von zwei Grundprinzipien aus:

- Aus der Annahme, daß der Benutzer ein Wissenselement kennt, läßt sich mit hoher Sicherheit folgern, daß er alle als Voraussetzung für die Kenntnis dieses Elementes geltenden Wissenselemente ebenfalls kennt.
- Aus der Annahme, daß der Benutzer ein Wissenselement nicht kennt, läßt sich folgern, daß er mit hoher Sicherheit nicht alle Wissenselemente kennt, die für die Kenntnis dieses Elementes vorausgesetzt werden.

Wenn das System annimmt, daß der Benutzer das spezielle Kommando `mail-q` kennt, schließt SC-UM mithilfe einer Wissensstandregel, daß er auch das allgemeine Kommando `mail` und das Konzept `Post` kennt. Hier können auch heuristische Regeln zum Einsatz kommen, wenn beispielsweise angenommen wird, daß ein Benutzer, der das Kommando `more` zum Lesen von Dateien verwendet, das im allgemeinen verfügbare `less` nicht kennt, da die meisten Benutzer das komfortablere `less` vorziehen würden.

Die Annahmen aus dem Benutzermodell werden vom Planberatungs- und vom Dialogsystem (vgl. Fig. 1) dazu verwendet, eine vom Benutzer angeforderte oder von SC für notwendig gehaltene Hilfe in einer dem Wissensstand des Benutzers angepaßten Form anzubieten. Einem wenig erfahrenen Benutzer wird eine ausführliche Hilfestellung in Verbindung mit einem Beispiel angeboten, während sich die Ausgabe für einen erfahrenen Benutzer auf eine kurze Beschreibung der wichtigsten Fakten, wie z. B. der Syntax eines Kommandos, beschränkt. Zusätzlich wird der Benutzer auf Wissenselemente hingewiesen bzw. ihm wird eine Beschreibung der Wissenselemente vermittelt, die für eine angeforderte Hilfe als Voraussetzung für das Verstehen notwendig sind, für die aber im Benutzermodell verzeichnet ist, daß der Benutzer sie nicht kennt.

8. Implementationsstatus

Der SINIX Consultant wurde in Interlisp-D und dem darauf aufsetzenden Objekt-orientierten Programmiersystem LOOPS auf einem Siemens APS 5815 entwickelt und anschließend auf SINIX-Rechner unter FranzLisp und PMFS[6] portiert [BEG*88].

Die Interlisp-Version ist lauffähig auf dem APS 5815; die FranzLisp-Version ist in der gleichen Funktionalität verfügbar für den Siemens MX2 und den MX500 sowie DEC VAX-Rechner. Beide Versionen sind sowohl interpretiert als auch compiliert verfügbar; die Systemgröße zur Laufzeit beträgt interpretiert ca. 3 MB bei der Interlisp-Version und ca. 2.3 MB bei der FranzLisp-Version.

Es wurden Laufzeitmessungen für die Bearbeitung einzelner Anfragen und Kommandos auf den vier Implementierungsmaschinen durchgeführt. Die (handgemessenen) Antwortzeiten des SC-Systems lagen - je nach Rechner große - im Bereich von einigen Sekunden bis zu maximal ein bis zwei Minuten.[7] Diese Messungen wurden für die interpretierten Versionen durchgeführt; durch Compilation des Systems kann eine Verbesserung der Laufzeiten um den Faktor 2-3 erreicht werden. Somit bewegen sich die erzielbaren Antwortzeiten des SC durchaus im Rahmen der für den Einsatz eines intelligenten Hilfesystems akzeptablen Werte.

Die Entwicklung des SINIX Consultant wird im Rahmen folgender Diplomarbeiten weitergeführt:

1. Entwicklung eines kasusrahmenbasierten Parsers zur robusten Sprachanalyse.
2. Entwicklung eines Generators für die Konstruktion flexibler natürlichsprachlicher Antworten.
3. Entwicklung eines Plangenerators, der zu vorgegebenen Zielbeschreibungen Sequenzen von SINIX-Kommandos generiert.
4. Erweiterung des Dialogsystems um eine Komponente zur Behandlung spezieller Dialogphänomene.

Danksagung

An der erfolgreichen Durchführung des SC-Projektes waren neben den drei Autoren viele studentische Mitarbeiter beteiligt. Daher möchten wir an dieser Stelle nochmals namentlich den Diplomanden E.-J. Blum, D. Dengler, M. Gutmann, G. Hector, B. Jung, P. Neißer, E. Nessen und den Praktikanten F. Berger, J. Engelkamp, R. Gintz, B. Kipper, P. Sommer und M. Weichel für ihre Mitarbeit danken.

Außer den oben genannten Projektmitgliedern haben zum erfolgreichen Gelingen des

[6]PMFS (Poor Man's Flavor System) ist ebenfalls eine Objekt-orientierte Programmiersprache, die allerdings wesentlich weniger Sprachkonstrukte zur Verfügung stellt als LOOPS.

[7]Genauere Angaben zu den Zeitmessungen finden sich in [BEG*88j

Projektes seitens der Siemens AG Dr. Gollan und Herr Falk beigetragen. Darüberhinaus gilt den Kollegen an unserem Lehrstuhl ein besonderer Dank für ihre ständige Diskussionsbereitschaft.

Literatur

[BEG*88] F. Berger, J. Engelkamp, R. Gintz, B. Kipper, P. Sommer, and M. Weichel. *Eine SC-Version auf UNIX-Rechnern.* Memo, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1988.

[DGH87] D. Dengler, M. Gutmann, and G. Hector. *Der Planerkenner REPLIX*. Memo No. 16, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[DGS87] L. Danlos, S. Guez, and S. Sabbagh. INTERIX: An Intelligent Help System. In *(unpublished paper)*, Marcoussis, France, 1987.

[DH82] R. Douglass and St. Hegner. An Expert Consultant for the UNDC Operating System: Bridging the Gap Between the User and Command Language Semantics. In *Proceedings CSCSI/SCEIO, Conference 1982, Saskatoon, Saskatchewan, 17-19 May 1982*, pages 92-96, 1982.

[Fil68] C. J. Fillmore. The Case for Case. In E. Bach and R. T. Harms, editors, *Universals in Linguistic Theory*, pages 1-90, Holt, Rinehart and Winston, Chicago, 1968.

[FN86] W. Finkler and G. Neumann. *MORPHIX - Ein hochportables Lemmatisierungsmodul für das Deutsche.* Memo No. 8, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1986.

[Hec87] M. Hecking. How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant. In *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1-4 September, 1987*, pages 657-662, 1987.

[Hec88a] M. Hecking. The SINIX Consultant - Towards a Theoretical Treatment of Plan Recognition. In Norvig, Wahlster, and Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence*, Springer, Heidelberg, 1988. (forthcoming).

[Hec88b] M. Hecking. *Towards a Belief-Oriented Theory of Plan Recognition.* Memo, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1988. (forthcoming).

[Heg88] St. J. Hegner. Knowledge Representation in Yucca-II: Exploiting the Formal Proper-ties of Command Language Behavior. In Norvig, Wahlster, and Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence*, Springer, Heidelberg, 1988. (forthcoming).

[HH87] M. Hecking and K. Harbusch. *Plan Recognition through Attribute Grammars*. Memo No. 17, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[HKN*88] M. Hecking, C. Kemke, E. Nessen, D. Dengler, M. Gutmann, and G. Hector. *The SINIX Consultant - A Progress Report.* Memo, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1988. (forthcoming).

[HSS87] C. Houy, A. Scheller, and K. Schifferer. PORTFIX - Eine Portierung von Franz Lisp unter SINIX. In *Kompendium zum I.I.I. Forum am 4/5.11.1987 in Saarbrücken,* pages 103-107, Universität des Saarlandes und Siemens AG, 1987.

[Jer85] J. Jerrams-Smith. SUSI - a Smart User-System Interface. In P. Johnson and S. Cook, editors, *People and Computers: Designing the Interface,* Cambridge University Press, Cambridge, 1985.

[Jun87] B. Jung. *Wissenrepräsentation für ein intelligentes Hilfesystem.* Diplomarbeit, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[Kem85] C. Kemke. Entwurf eines aktiven, wissensbasierten Hilfesystems für SINIX. *LDV-Forum,* 2:43-60, 1985.

[Kem86] C. Kemke. *The SINIX Consultant - Requirements, Design, and Implementation of an Intelligent Help System for a UNIX Derivative.* Report 11, Dept. of Computer Science, University of Saarbrücken, W.Germany, 19S6.

[Kern87] C. Kemke. Representation of Domain Knowledge in an Intelligent Help System. In *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1-4 September, 1987,* pa-ges 215-220, 1987.

[Kem88a] C. Kemke. Darstellung von Aktionen in Vererbungshierarchien. In W. Hoeppner, editor, *GWAI-88. 12th German Workshop on Artificial Intelligence,* Springer, Heidelberg, 1988.

[Kem88b] C. Kemke. What Do You Know About Mail? Representation of Commands in the SINIX Consultant. In Norvig, Wahlster, and Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence,* Springer, Heidelberg, 1988. (forthcoming).

[MP88] M. M. Matthews and W. Pharr. Knowledge Acquisition for Active Assistance. In Norvig, Wahlster, and Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence,* Springer, Heidelberg, 1988. (forthcoming).

[MW88] P. McKevitt and Y. Wilks. Inference in an Operating System Consultant. In Norvig, Wahlster, and Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence,* Springer, Heidelberg, 1988. (forthcoming).

[Nes87a] E. Nessen. *Benutzermodellierung in einem intelligenten Hilfesystem.* Diplomarbeit, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[Nes87b] E. Nessen. *SC-UM: User Modeling in the SINIX-Consultant.* Memo No. 18, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[QDF86] A. Quilici, M. Dyer, and M. Flowers. AQUA: AN INTELLIGENT UNIX ADVISOR. In *Proceedings of the 7th European Conference on Artificial Intelligence,* pages 33-38, 1986.

[Wah84] W. Wahlster. Cooperative Access Systems. *Future Generations Computer Systems,* 1(2):1O3-111, 1984.

[Wah86] W. Wahlster. The Role of Natural Language in Advanced Knowledge-Based Systems. In H. Winter, editor, *Artificial Intelligence and Man-Machine Systems,* pages 62-83, Springer, Heidelberg, 1986.

[WK86] W. Wahlster and A. Kobsa. Dialog Based User Models. *IEEE Proceedings,* 74(7):948-960, 1986.

[WMA*86] R. Wilensky, J. Mayfield, A. Albert, D. Chin, C. Cox, M. Luria, J. Martin, and D. Wu. *UC - A Progress Report.* Report No. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkeley, 7 1986.