Modell eines adaptierbaren, rechnergestützten, wissenschaftlichen Arbeitsplatzes am Alfred-Wegener-Institut für Polar- und Meeresforschung

Modelling an Adaptive, Computer Based, Scientific Workplace at the Alfred-Wegener-Institute for Polar and Marine Research

---

Lutz-Peter Kurdelski

Ber. Polarforsch. 230 (1997)
ISSN 0176 - 5027

In memoriam
Ida, Alfred und Georg Hendrian
Alex Kurdelski

D17

Lutz-Peter Kurdelski

Unterhachinger Str. 49, D-81737 München, Bundesrepublik Deutschland

Diese Arbeit ist der inhaltlich unveränderte, von der Prüfungskommission des Fachbereiches Informatik der Technischen Hochschule Darmstadt genehmigte Druck der vom Autor im April 1994 vorgelegten Dissertation mit dem Titel *Entwicklung eines Resourcen–Management–Systems als Bestandteil einer adaptierbaren Benutzerschnittstelle für den wissenschaftlichen Arbeitsplatz.* Die Anhänge wurden nicht mitübernommen und sind der Originalarbeit des Autors zu entnehmen

Inhaltsverzeichnis

Zusammenfassung

Am Alfred–Wegener–Institut für Polar– und Meeresforschung wird zur Bewältigung der wissenschaftlichen Aufgaben in zunehmendem Maße eine komplexe Infrastruktur untereinander vernetzter Rechenanlagen eingesetzt. Dem Wissenschaftler steht ein Arbeitsplatzrechner sowie ein lokales Netzwerk zur Verfügung, das in mehreren Hierarchiestufen den Datenfluß kanalisiert. Zusätzlich bestehen Verbindungen zu externen Rechnern mit einer Funktionalität, die vom einfachen Datentransfer bis zur vollständigen Integration in das Netzwerk reicht. Die Rechenanlagen stellen dem Benutzer neben der dezentralen Rechenkapazität des Arbeitsplatzrechners dedizierte Dienste innerhalb des Netzwerks zur Verfügung.

Die Rechner innerhalb des Netzwerks werden als *compute server*, Datenbankserver, Graphikserver und Datenserver eingesetzt. Dabei wird der Benutzer durch verteilte Filesysteme bei der Verwaltung der Daten im Netzwerk unterstützt. Die einzelnen Rechner und Rechnerdienstleistungen werden über Zugriffsprotokolle angesprochen, die einen einfachen Datentransfer und Ausführung dedizierter Befehle bis hin zur direkten Kommunikation der Prozesse ermöglichen. Über Hochgeschwindigkeitsverbindungen einiger Rechner (>100MBit/s Glasfaserkabel) können Graphiken, die auf einem Vektorrechner berechnet wurden, als Sequenz („Film“) auf einem graphischen Terminal ausgegeben werden.

Trotz dieser leistungsfähigen Methoden ist der Benutzer dieser Anlage gefordert, tiefe Kenntnisse über die Zusammenhänge im Netzwerk zu erwerben, um in diesem heterogenen Netzwerk zu navigieren und sich die erforderlichen Ressourcen nutzbar zu machen. Daher wurden am AWI mehrere Projekte zur Unterstützung des Benutzers entwickelt.

- Die Verwaltung und Steuerung von Ressourcen wie Rechner, Datenbanken, Ausgabegeräte, Datenfiles, Programme etc., d. h. Funktionseinheiten innerhalb des Rechnernetzwerks, erfolgt über ein Steuerungssystem.

- Die graphische Repräsentation wissenschaftlicher Daten wird durch ein Programm gesteuert, das dem Benutzer bei der Auswahl von Graphiken, deren Modifikation (Skalieren, Drehen etc.) und der Auswahl des Ausgabemediums (Terminal, Tintenstrahldrucker etc.) hilft.

- Die wissenschaftlichen Datenbanken werden auf Datenbanksystemen *(database server)*, die das *client–server*–Konzept unterstützen, bereitgestellt.

In der vorliegenden Arbeit wird die Arbeit der Nutzer im Umgang mit den Ressourcen untersucht. Dabei werden Methoden zum Aufbau eines Benutzungsmodells entwickelt. Ein *ResourcenManagementSystem*, das den Benutzer bei der Navigation im Netzwerk und bei der Programmierung unterstützt, wird vorgestellt. Dabei wird auch auf Ergebnisse eines in Vorarbeiten zu dieser Arbeit entwickelten *TEX–Desktops* zurückgegriffen. Im einzelnen wurden in folgenden Bereichen neue Ergebnisse erzielt:

- ANALYSEKONZEPT
  Ein neues Konzept zur Erfassung der softwareergonomischen Anforderungen an einen wissenschaftlichen Arbeitsplatz wird vorgestellt.

- Erfassung von Wissen (Regeln, Fakten)
  Die Aktionspläne der Benutzer werden *vor* Erstellung des Systems, u. a. durch die Analyse von benutzereigenen Makrodateien, mittels des *GOMS***–Modells erfaßt.

- Eigene Beschreibungssprache
  Die Aktionen des Benutzers zur Laufzeit des Systems werden in einer eigenen Beschreibungssprache, die eng an die *GOMS***–Beschreibung angelehnt ist, notiert, um die Übersetzung von abstrakten Befehlsfolgen in Betriebssystembefehle zu realisieren.

- Beschreibung des Systemverhaltens
  Systemkomponenten und deren Verhalten, wie Benutzer, System oder Ressource, werden mittels *RFA/ATN*–Netzen und unter Verwendung der objektorientierten Analyse beschrieben.

- Abstrakte Beschreibung der Dialoghistorie
  Die Dialoghistorie, mit der Zustandsänderungen im System protokolliert werden, wird in dieser Arbeit formal als *Abstrakter Datentyp* definiert und im Hinblick auf die Implementierung von *UnDo/ReDo* betrachtet.

- Makroerstellung
  Eine Anwendung der Dialoghistorie ist die Archivierung von Handlungsfolgen als Makro, das Generieren von Makrobefehlen, die mit einem einfachen Erkennungsmechanismus aus den Aktionen des Benutzers wieder herausgefiltert werden können.

Auf dieser Basis wird ein System zur Navigation in einem Rechnernetz implementiert. Mit diesem System wird der Benutzer von der Notwendigkeit entlastet, Kenntnisse über die Zusammenhänge innerhalb des Netzwerks zu erzielen. Die Auswahl der Ressourcen erfolgt allein über deren Funktionalität.

Abstract

The scientist in his working environment at the Alfred–Wegener–Institute for Polar und Marine Research is part of a complex infrastructure. Many different computer build this environment by connecting via local area networks (LAN). The scientist uses a workstation from which he can use any service through the connection to the LAN. Several step in a hierarchie lead the flow of data. In addition there are several connections to external computer. The functionality of this environment spread over simple data transfer to complete integration into the network. Beneeth the local compute capability of the workstation this infrastructure provide dedicated services within the network.

Each computer in this network has its own capability of providing services as there are compute server, database server, graphic server and file server. Distributed filesystems help the user managing his data in the network. Each computer and each service is accessible by special protocols which include simple file transfer, execution of dedicated commands, and inter process communication. Via a high speed network (>100MBit/s fiber cable) images

which are produced on a vector computer can be transferred as a film to a graphical terminal.

In spite of this powerful possibilities the user of this environment is expected to aquire deep knowledge on the connections within the network. He has to navigate through the heterogenuous network to to use all resources. At the AWI several projects are initiated to help the user in its environment.

- The management and control of resources like computer, databases, output devices, files, program, etc., i.e. functional units, is done by a control system.
- The graphical representation of scientific data is controlled by a program which supports the users in selecting graphical images, modifying these images (e.g. scaling, rotating), and presentation of these images (e.g. terminal, inkjet printer).
- Scientific databases are offered by database servers which support the client/server concept.

This study is based on the problem of user while working with these resources. During this process methods to build a use model are developed. A ressourcen management system will be introduced which helps the user in navigation within the network and programming. Results of a parallel build user interface *TEX-Desktop* are widely used. The following list shows topics where new results were achieved.

- ANALYSIS
  A new concept for collecting the requirements for an software ergonomic scientific working place is provided.
- COLLECTION OF KNOWLEDGE (RULES, FACTS)
  With the *GOMS***–model the action plans of a user can be analysed before developing the system.
- NEW DESCRIPTION LANGUAGE
  The actions of a user while working with the system are noted by a new description language which is based on the *GOMS***–description. This description language realizes the compilation of an abstract command sequence in commands of the operating system.
- DESCRIPTION OF SYSTEM BEHAVIOUR
  RFA/ATN are used to describe the system components and their behaviour. This description is based on the object oriented model.
- ABSTRACT DESCRIPTION OF THE DIALOG HISTORIE
  Each system change is noted in the dialog history. For formal analysis an abstract data type is defined which helps at design time to check undo/redo problems.
- MACROS
  One aspect of the dialog historie is the abillity of extracting command sequences from the historie. This sequence is stored as a parameterised macro. An easy recognition mechanism is build which filters users action.

Based on this results a system for navigation in a heteregenuous network is implemented. This system reliefs of first getting deep knowledge of internal connections before using the environment. A resource is selected only by its functionality.

Vorbemerkung des Alfred-Wegener-Instituts für Polar- und Meeresforschung, Rechenzentrum

Rechner am wissenschaftlichen Arbeitsplatz sind ein unverzichtbares Werkzeug für die Forschung. Die Entwicklung der Informationstechnologie ist in den letzten Jahren durch ihren Einsatz in der Technik und in der Wirtschaft und dort insbesondere durch den Bürobereich geprägt worden. In der Wissenschaft wird sie aber vielfach anders genutzt. Analysen des Benutzerverhaltens von Wissenschaftlern sind aber bisher nur unzureichend und mit unscharfer Methodik durchgeführt worden.

Im Alfred-Wegener-Institut finden sich durch die unterschiedlichen Aufgabenbereiche der Arbeitsgruppen, wie Biologie, Chemie, Physik, numerische Modellierung usw., sehr unterschiedliche wissenschaftliche Arbeitsweisen, die sich auch in der Nutzung von Rechnern und Rechnerinfrastruktur zeigen. Die Bandbreite reicht vom PC-Arbeitsplatz mit Tabellenkalkulationsprogrammen über die Nutzung und Programmierung von UNIX Arbeitplatzrechnern, den Einsatz von Datenbankservern bis zur Modellierung auf Höchstleistungsrechnern mit Vektor- und Parallelarchitektur.

Für alle wissenschaftlichen Bereiche sollen im Alfred-Wegener-Institut je nach ihren spezifischen Anforderungen optimale Informationstechnologien nach Stand der Technik zur Verfügung gestellt werden, die in ein Gesamtkonzept eingebunden werden müssen. Aus dieser Aufgabenstellung heraus entstand die Idee zur vorliegenden Arbeit, ein Analyseverfahren für das wissenschaftliches Benutzerverhalten zu entwickeln. Ziel war es, ein Modell für den wissenschaftlichen Arbeitsplatz am Alfred-Wegener-Institut zu erstellen, um die aktuelle und zukünftige Entwicklung in der Informationstechnologie in Bezug auf ihre Anwendungen für die Wissenschaft besser beurteilen zu können.

Dr. M. Reinke

Danksagung

An dieser Stelle möchte ich mich bei all denen bedanken, die dazu beitrugen, daß diese Arbeit entstehen konnte.

Mein Dank gebührt Herrn Dr. Wolfgang Hiller, ohne dessen Engagement das in dieser Arbeit diskutierte Problem nicht bearbeitet worden wäre. Seine Idee, nicht ausschließlich Serviceleistungen für das Alfred–Wegener–Institut bereitzustellen, sondern den Benutzer durch weitergehende, wissenschaftlich fundierte Ergebnisse zu unterstützen, ließ eine Rechnerumgebung entstehen, die von den Benutzern akzeptiert wird. Herrn Dr. Manfred Reinke gebührt mein Dank für die ständige Bereitschaft zu Diskussionen über Software–Engineering und Datenbanken. Dr. Hiller und Dr. Reinke betreuten die Arbeit in allen Belangen im Alfred–Wegener–Institut.

Das Institut für Integrierte Publikations– und Informationssysteme (IPSI) der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in Darmstadt unterstützte diese Arbeit. Ich möchte Herrn Prof. Dr. Erich Neuhold für die Bereitschaft danken, diese Arbeit betreut zu haben, was auf Grund der Entfernung sicher nicht immer leicht war. Herr Dr. Heinz–Ulrich Hoppe (IPSI) übernahm die Aufgabe, die Arbeit fachlich zu betreuen, wofür ich mich hiermit bedanke. In vielen anregenden Diskussionen entstanden neue Ideen, die in dieser Arbeit verarbeitet und diskutiert wurden. Herr Lothar Rostek (IPSI) war stets bereit, Probleme mit *Smalltalk-80* und TEX zu diskutieren. Seine wertvollen Hinweise zu Implementationstechniken halfen die Realisierung zu beschleunigen.

Bei den Mitarbeitern des Rechenzentrums und den Mitarbeitern des Alfred–Wegener–Instituts möchte ich mich für die Unterstützung bei technischen Problemen und die Bereitschaft zur Mitarbeit bedanken.

Ich möchte mich bei den von mir betreuten Studenten bedanken, die im Rahmen ihrer Diplomarbeiten das TEX–System zur Einsatzreife führten. Herr Dipl.–Ing. Siegfried Makedanz führte eine Portierung durch. Herr Dipl.–Ing. Glenn Praetsch entwickelte hierfür einen Formeleditor.

Ein großer Dank geht abschließend an alle meine Freunde und Verwandten, die mich in den letzten Jahren ertragen haben.

Unsere Abhängigkeit von Computern ist weitreichend und tiefgehend, und sie wird immer intensiver. Diese Tatsache verursacht Angst, — Angst, den Computer selbst zu benutzen, Angst vor dem Umgang mit den Computern, die uns umgeben, und Angst vor der Wirkung des Computers auf unsere Gesellschaft. *John Shore*

Kapitel 1

Einleitung

Das einleitende Zitat von John Shore gibt auch heute noch Anlaß zur Beunruhigung, ob der Computer in der jetzigen Form für den Menschen das richtige Hilfsmittel darstellt. Betrachtet man dabei, daß eine intensive und individuelle Betreuung des Benutzers bei seiner Arbeit durch den Rechner auch tiefgehende Informationen über den Anwender erfordert, so wird die Angst auch in anderer Weise verständlich.

Mit den zur Zeit auf dem Markt befindlichen Systemen sind Geräte verfügbar, die sich nach kürzester Einarbeitungszeit intuitiv und leicht bedienen lassen; und es macht wieder Freude, mit den Geräten umzugehen. So wie sich niemand Gedanken darüber macht, wie ein Radio, ein Fernsehapparat oder eine Waschmaschine funktionieren, macht man sich immer weniger Gedanken darüber, wie ein Computer wirklich arbeitet. Nicht mehr der Apparat steht im Vordergrund sondern die Anwendung. Dies wird insbesondere an der Verwendung des Rechners als Terminkalender, mit wesentlich mehr Leistungsfähigkeit als ein herkömmlicher, papiergebundener Kalender, oder als Schreibgerät deutlich. Der Traum vom *Kollegen Computer* ist noch nicht zu Ende geträumt[1]. Und die Rechner, die zur Zeit an den Arbeitsplätzen, auch in der Wissenschaft, vorhanden sind, helfen dem Benutzer, doch sind individuelle Arbeitsplatzgestaltung und Arbeitsabläufe noch immer nicht in dem Maße realisiert, wie wir es vom herkömmlichen Arbeitsplatz her kennen.

Die in der vorliegenden Arbeit vorgestellten Konzepte und Analysen sollen bei der Individualisierung des Arbeitsplatzes helfen. Dazu bedarf es zunächst eines kurzen Überblicks über die Verfahren zur Benutzerverhaltensanalyse, einiger Bemer-

[1]Daß sich dahinter auch Gefahren verbergen, darf nicht übersehen werden. Nicht alles, was machbar ist, sollte auch getan werden [Dre72, Wei90].

kungen über den Arbeitsplatz[2] und eines historischen Überblicks.

1.1 Entwicklung benutzerfreundlicher Systeme

Das mit der Zeit ständig wachsende Angebot an, auch für den wissenschaftlichen Nutzer, interessanten Programmen wie Tabellenkalkulation, Textverarbeitung, Datenbanken und graphische Präsentation führte auch zu einer Vielzahl verschiedener Möglichkeiten, ein Programm zu bedienen. Auf einigen wenigen Rechnern wurde recht früh versucht, den Benutzer vom Betriebssystem zu lösen und eine einheitliche Benutzerführung zu bieten (Apple LISA; [SIK+83]). Diese anwendungsorientierten Rechner waren oft nur für festgesetzte Anwendungen vorgesehen (Bildverarbeitung, Büroautomation, [SIK+83]).

Um dem Benutzer den Umgang mit dem Betriebssystem zu erleichtern, wurden dialogorientierte Schnittstellen für die Programme entwickelt (Oberflächen)[3]. Wenn der Anwender nicht innerhalb der Produktfamilie eines Herstellers blieb, was aufgrund unterschiedlicher Leistungsdaten der Programme nicht sinnvoll erschien, mußte er mit stark differierenden Schnittstellen arbeiten[4] (fehlende Konsistenz).

Weitere von den Endbenutzern kritisierte Aspekte sind mangelnde Korrekturmöglichkeiten und fehlende Hinweise beim Auftreten eines Bedienungsfehlers oder eines allgemeinen Fehlers sowie Hilfestellungen und Erklärungen zu Befehlen und Funktionen. Ein Handbuch mußte ständig bereitgehalten werden. Im Laufe der Entwicklung haben die Hersteller mehrere Wege eingeschlagen, um das Handbuch zu vermeiden. Sie entwickelten sorgfältig erstellte Fehlertexte, Hilfestellungen und auch Tutorien [Dig86b, Dig88, Mic87], um den Benutzer, so weit wie möglich, *online*, d. h., während er an seinem Rechner arbeitet, zu unterstützen.

Bei der Entwicklung einer Benutzungsoberfläche für Systeme, die bereits existierende Applikationen mit vorgegebenen Schnittstellen besitzen, ist die Forderung nach Konsistenz nur unter großen Schwierigkeiten einzuhalten.

Jedes System ist in gewissen Grenzen adaptierbar[5]. Adaptionen durch den Benutzer können beispielsweise auf dem Apple Macintosh mit dem *Macro Maker*[6] vorgenommen werden. Um ein adaptierbares oder adaptives System aufzubauen, bedarf es einer umfassenden Erfassung von Wissen und Regeln über die Arbeit des Benutzers mit dem Rechner.

Auf dem bereitgestellten Wissen basieren auch die Hilfe–Systeme. Wichtige Vorgaben für Hilfe–Systeme wie Dynamik, Aktivität und Individualität sind von Bauer

[2]Hier wurde der wissenschaftliche Arbeitsplatz am Alfred–Wegener–Institut näher untersucht.

[3]Menüführung des Benutzers: z. B. Textverarbeitung, Datenbanken, Tabellenkalkulation

[4]z. B. Tabellenkalkulation mit *Lotus 1-2-3*, Textverarbeitung mit Microsoft *Word*

[5]Dazu zählen die Anpassung von Systemparametern, kleine, ablauffähige Befehlsdateien, die festgelegte Aktionssequenzen enthalten, sowie Symboldefinitionen.

[6]Der *Macro Maker* ist ein kleines Programm, das auf Anforderung die Aktionen des Benutzers mitprotokolliert. Diese Folge ist dann unter einer Tastenkombination verfügbar. Der *Macro Maker* ist im neuen System 7 nicht mehr enthalten

und Schwab dargestellt [BS87]. Dem System wird zum aktuellen Zeitpunkt eine Dokumentation mitgegeben, die den Benutzer über stichwortartige Verweise zu den gewünschten Informationen leitet. Viele Programme nutzen den Kontext (z. B. ein ausgewähltes Feld in einer Maske), um auf die Seite in einem Manual zu verweisen.

1.2 Methoden zur Benutzerunterstützung

Adaptierbare und adaptive Benutzungsoberflächen sind gegenwärtig ein Gebiet intensiver Forschung. Die Anpassung des Systems kann vom Benutzer selbstständig durchgeführt werden *(adaptierbare Benutzerschnittstelle)*, oder das System verfolgt den Dialog mit dem Benutzer, um sich gegebenenfalls an veränderte Situationen anzupassen *(adaptive Benutzerschnittstelle)*. Der Benutzer wird durch

- Visualisierung von Objekten, Daten und Zuständen,
- Modellierung des Benutzers und Planerkennung und
- *UnDo/ReDo*–Fähigkeit und Hilfe–Funktionen

bei seiner Arbeit mit dem Rechner unterstützt.

Das Benutzungsmodell ist die Zusammenfassung von Aktionsregeln und Handlungsplänen des Benutzers. In diesem Sinne ist die enge Verknüpfung der ersten beiden Punkte offensichtlich. Idealerweise verhält sich ein Anwendungssystem wie ein Gesprächspartner[7]. Die Planerkennung dient im Idealfall der Erweiterung der Planwissensbasis und damit der Individualisierung des Benutzungsmodells. In diesem Sinne ist das Benutzungsmodell das Bild, das sich der Gegenpart des Benutzers, hier der Computer, vom Benutzer macht.

Aus einer Mitschrift der Benutzeraktionen können Aktionssequenzen extrahiert werden, die häufig angewandt werden. Diese Pläne unterliegen in dem jeweiligen System einer besonderen Behandlung, insbesondere wird mit geeigneten Verfahren nach Übereinstimmungen einer aktuellen Aktionssequenz mit gespeicherten Sequenzen gesucht. Der Benutzer muß gegebenenfalls mit einer Hilfsfunktion auf Aktionen, fortzusetzende Pläne und Unterstützung zur aktuellen Arbeit hingewiesen werden. Im folgenden werden einige wichtige Begriffe erläutert.

Benutzungsmodelle

Mit den verschiedenen Methoden, mit denen die Interaktion zwischen Mensch und Computer analysiert wird, entstanden unterschiedliche Definitionen des Begriffes *Benutzungsmodell*. Dabei kann zwischen *operativen, stereotypen* und *mentalen Benutzungsmodellen* unterschieden werden. (Siehe Abbildung 1.1.)

Die Bildung eines Modells über einen Benutzer erfolgt meist für eine spezielle Anwendung. So modelliert Kobsa die Intentionen, Annahmen und Pläne der Benutzer

[7]Kritiken zu dieser eingeschränkten Sicht werden von Dreyfus beschrieben [Dre72].

HACKER	
hacking skill	high
math skill	med
work hours	night

THEORETICAN	
hacking skill	low
math skill	high
work hours	day

Abbildung 1.1: Zwei Stereotype, die der Benutzer annehmen kann, werden durch die Attribute *hacking skill, math skill, work hours* beschrieben. Die unbestimmten Werte *high, med, low* werden auch durch numerische Werte angegeben (aus [Ric89]).

bei natürlich–sprachlicher Eingabe [Kob85]. Im Unterschied zur Analyse der natürlichen Sprache werden bei mentalen Modellen geringere Anforderungen bezüglich der Redundanz gestellt, da Aktionen des Benutzers im Gegensatz zu einem natürlich–sprachlichen Satz, der gegebenenfalls nur aus dem Kontext heraus eindeutig gemacht werden kann, eindeutig sind [Bö87, S. 146].

Ein Ansatz, den Benutzer *allgemein* zu modellieren, wird von Kass und Finin mit dem GUMS–Modell[8] verfolgt [KF88a, KF88b]. Unterschiedliche Informationsdarstellungen, die auf Annahmen der Benutzer basieren, werden von Gargan, Sullivan und Tyler beschrieben [GJST88].

Einige Systeme, die auf der Modellierung des Benutzers basieren und zusätzlich Planerkennungsverfahren einsetzen sowie aktive Hilfe anbieten, sind u. a. bei Fischer und Gunzenhäuser zu finden [FG86].

Planerkennung

Die Modellbildung des Benutzers schließt eine Erkennung des Aktionsplanes ein. Der erkannte Plan kann fortgesetzt werden, ebenso, wie ein menschlicher Dialogpartner aus wenigen Anfangsbedingungen auf das Ziel schließen kann. Wichtig ist hierbei die Selektion aus verschiedenen Handlungsplänen. Eine Übersicht über Methoden zur Planerkennung wird von Carberry gegeben [Car88, Car89].

Nach Card, Moran und Newell sind Pläne Folgen von Benutzeraktionen *(OPERATIONS)*, mit denen ein Ziel *(GOAL)* erreicht werden soll. Mit den Aktionen des Benutzers wird eine Menge von möglichen Plänen selektiert, und der nach heuristischen Prinzipien ausgewählte Plan wird fortgesetzt [Car89].

Die Analyse der Benutzeraktionen zur Verbesserung der Handlungsfähigkeit des Benutzers wird von Hoppe beschrieben [HP89, Hop89]. Das System ist in der Lage, suboptimale Pläne zu erkennen und dem Benutzer Hinweise zur Verbesserung der Effektivität seiner Aktionen zu geben[9].

Wie Hoppe [Hop88a] verwenden auch Chi [Chi85], Fleischhack und Weber

[8] GUMS: **G**eneral **U**ser **M**odelling **S**hell

[9] Ein suboptimaler Plan ist eine zur Lösung führende Aktionssequenz, die durch Einsatz neuer Befehle verkürzt oder verändert werden kann, z. B. :

[FW89], Vouros und Spyropoulos [VS89] und Browne, Sharratt und Norman [BSN86] strenge formale Methoden zur Beschreibung von Benutzeraktionen. Hoppe verwendet eine *Task Action Grammar* zur Darstellung der Benutzeraktion. Fleischhack und Weber benutzen den Formalismus der Petri–Netze, Vouros und Spyropoulos attributierte Grammatiken zur Wissensrepräsentation. Browne, Sharratt und Norman verwenden die von Card und Moran [Mor81] entwickelte *command language grammar* zur Spezifikation einer adaptiven Benutzerschnittstelle.

UnDo/ReDo–Methoden

UnDo/ReDo geben dem Benutzer die Möglichkeit, getätigte Aktionen zu widerrufen oder erneut auszuführen. Aus den Aktionsfolgen können neue Makros erstellt werden. Diese Makros sind wiederum neue Ziele, die den Handlungsplänen hinzugefügt werden können.

Der Unterschied zwischen Dialogverfolgung mit *UnDo/ReDo* und Handlungsplanerkennung liegt in der Zielsetzung. Die Handlungsplanerkennung soll das System in die Lage versetzen, den Benutzer zu *verstehen*. Die Dialogverfolgung gibt dem Benutzer Aufschluß über durchgeführte Aktionen und deren Ergebnis.

Tutorielle Systeme und Hilfs–Funktionen

Die gelegentlichen Fragen des Benutzers an das System nach Unterstützung oder Hilfe werden durch Hilfe–Systeme beantwortet. Je nach Einsatz und Ausformung der Hilfe erhält der Benutzer umfassende oder nur sehr kurze Antworten. Die kontextabhängige Hilfe wird gefordert, um dem Benutzer zu dem Problem im Umfeld seines aktuellen Systemzustandes Hinweise zu geben.

Die Erweiterung dieser Hilfe–Systeme zu tutoriellen Systemen, die dem Benutzer nicht nur Hinweise geben, sondern auch Simulationen durchführen können, wird an vielen Stellen nützlich sein, wenn z. B. ein Benutzer die Auswirkungen eines Befehls nicht abschätzen kann. Tutorielle Systeme dienen in erster Linie der *intelligenten* Einarbeitung und Führung des Benutzers. Das System ergänzt den menschlichen Tutor.

Ein Hilfe–System, das dem Benutzer hilft, mit den Techniken des Systems vertraut zu werden, wird von Tuck und Olsen beschrieben [TO90]. Herrmann diskutiert die Selbsterklärungsfähigkeit des Systems, um daraus ein Konzept zur Gestaltung der Benutzungsoberfläche abzuleiten [Her86b, Her87]. Dieses Konzept steht den streng regelorientierten Ansätzen entgegen.

Suboptimaler Plan	optimierter Plan
`rename a b`	`rename a b`
`rename c d`	`rename c a`
`rename b c`	`rename b c`
`rename d a`	

Ergonomie

Ergonomie ist ein Schlagwort, das in der am Anfang vorgestellten Auflistung neuer Techniken nicht auftritt. Ergonomie stellt sich als ein Sammelbegriff dieser neuen Ansätze dar. Jedoch gibt es allgemeine Punkte zu beachten, die aus den bisher vorgestellten Überlegungen nicht unmittelbar deutlich werden. Diese betreffen insbesondere das Dialogverhalten und die Selbsterklärungsfähigkeit des Systems. Für den Benutzer ist ein Dialogverhalten, das die gewohnte Arbeitsumgebung wiederspiegelt, Akzeptanz fördernd [Shn85][10]. In der DIN 66234 Teil 5: *Bildschirmarbeitsplätze: Codierung von Information* wird dafür der Begriff des *verläßlichen Dialogverhaltens* eingeführt [iD84a]. Nach Stevlovsky kann dies auch recht kurz zusammengefaßt werden [Ste84]:

Different things must be done differently
und
similar things must be done similarly.

Herczeg unterscheidet weiter zwischen *innerer* und *äußerer Konsistenz*, die sich auf das Verhalten des Systems an sich *(innere Konsistenz)* und das Verhalten des Systems im Zusammenspiel mit verschiedenen Applikationen beziehen *(äußere Konsistenz)* [Her86a].

1.3 Eine reale Arbeitsplatzsituation

Die Aufgabe, eine adaptierbare/adaptive Benutzerschnittstelle am Alfred–Wegener-Institut für Polar– und Meeresforschung, im folgenden kurz AWI genannt, zu entwickeln, ist aus den Anforderungen der Benutzer an die Datenverwaltung und Datenaufbereitung entstanden. Wesentliche Probleme treten bei der Einarbeitung der Benutzer in die bestehende Hard– und Software und später durch die nur gelegentliche Nutzung auf. Da es unmöglich ist, alle Systeme neu zu erstellen, auch wenn dies der sinnvollere Weg zu sein scheint [BS87], sind intensive Arbeiten erforderlich, um dem Benutzer ein möglichst einfaches Bild der zur Verfügung stehenden Hilfsmittel zu geben. Ein kurzer Überblick über die Verfahren, Möglichkeiten und die bisher am Institut getätigten Arbeiten soll die vorliegende Arbeit in das Gesamtkonzept des Institutes einordnen.

Die Benutzerbetreuung kann im AWI in drei Komplexe aufgeteilt werden:

1. Bereitstellung von Datenbanken
2. graphische Datenverarbeitung
3. Bereitstellung von *Resourcen* im Netzwerk

[10] Die *Bleistift– und Papier*–Methode versagt bei der Anwendung auf Tabellenkalkulation *(spreadsheet)* [AW86]. Dies steht nicht im Widerspruch zu Shneiderman [Shn85], da die Tabellenkalkulation eine neue Technik ist, für die eine Arbeitsmetapher erst entwickelt werden muß.

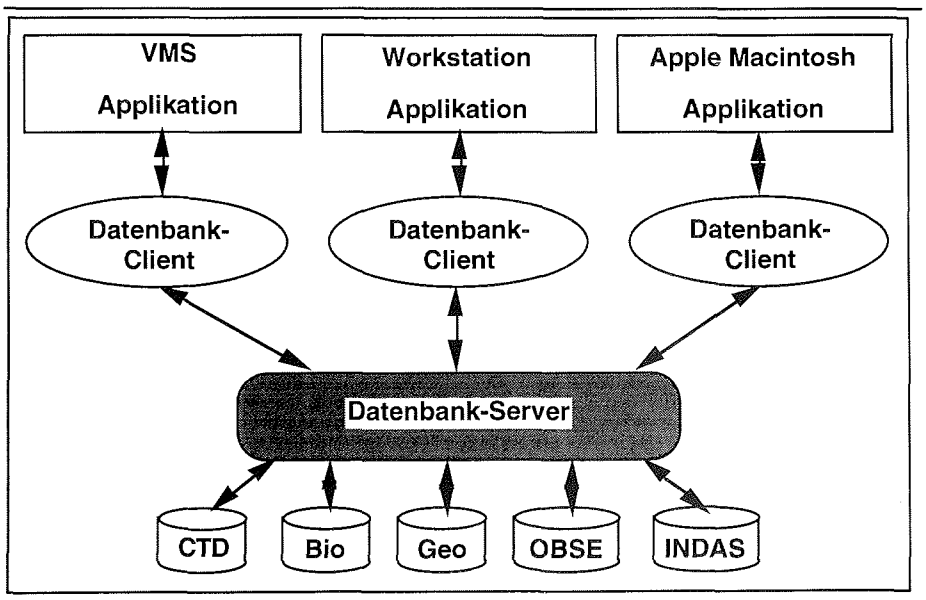


Abbildung 1.2: Die Faktendatenbank im AWI

Die während der Forschungsreisen der *FS Polarstern* gesammelten Daten werden den wissenschaftlichen Nutzern zur Verfügung gestellt. Da die Datenmengen nicht mehr ohne Hilfsmittel zu bearbeiten sind, wurde der Aufbau einer interdisziplinären Faktendatenbank für das AWI im Rahmen des Projektes *Informationssysteme am AWI* beschlossen (Abbildung 1.2). In diese Informationssysteme können die Wissenschaftler arbeitsteilig neue Daten einfügen, und sie können schnell und komfortabel auf die Daten in der Datenbank zugreifen.

Das Konzept zum Aufbau der Informationssysteme und die Definition der Benutzerschnittstelle mit den Problemen der rechnerunerfahrenen Benutzer waren Thema zweier Diplomarbeiten am AWI [Wes87, Wei87]. Während Westerwick Probleme bei der Einbindung von Datenbanken in übergeordnete Programme untersuchte, zeigte Weiß in seiner Arbeit, daß bei der Erstellung einer interaktiven Schnittstelle viele Fakten über den Benutzer mit einbezogen werden müssen. Ein erstes Konzept einer auf diesen Erkenntnissen beruhenden menügeführten Oberfläche auf Textterminals wurde von ihm erarbeitet (Abbildung 1.3).

Die Handhabung der Informationssysteme ist für den Anwender nicht ohne Einarbeitung möglich. Für den rechnerunerfahrenen Benutzer waren die Abfragesprachen, z. B. SQL, zu komplex. Er lehnte es ab, diese Sprachen zu erlernen. Diese Weigerung ist auch in der nur gelegentlichen Nutzung der Datenbank durch den

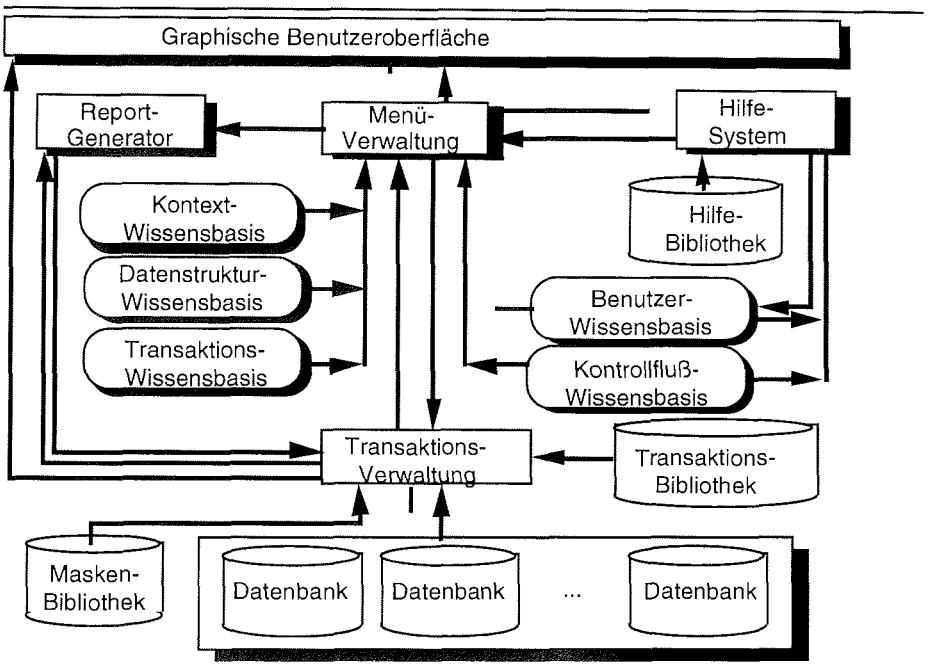


Abbildung 1.3: Ein Konzept einer graphischen, interaktiven und wissensbasierten Benutzerschnittstelle für ein Datenbanksystem (Aus [Wei87])

einzelnen Benutzer zu sehen, da die Einarbeitung und Wiederholung mit oft nicht vertretbarem Zeitaufwand erfolgen müssen.

Ein wesentliches Ergebnis der Arbeit von Weiß war die Forderung, ein Benutzungsmodell des wissenschaftlichen Arbeitsplatzes am AWI zu erstellen [Wei87]. Dieses Benutzungsmodell soll die Anforderungen und Kenntnisse des Benutzers in bezug auf die bereits vorhandenen Betriebsmittel der Benutzerschnittstelle zur Verfügung stellen, damit eine optimale Hilfe gewährleistet werden kann.

Ein Teil der Faktendatenbank, die Stationsdaten[11] der „FS Polarstern", wurden zum Anlaß genommen, im Rahmen einer Diplomarbeit am AWI eine graphische Oberfläche zur Bedienung der Datenbank zu erstellen [Opi89].

Die Probleme im graphischen Bereich sind durch die komplexe, für den rechnerunerfahrenen Benutzer ungeeignete Software und durch die große Anzahl verschiedener numerischer und graphischer Bibliotheken und Ausgabegeräte gegeben. Eine

[11]Eine Station ist ein Zeitpunkt und geographischer Ort, an dem Experimente durchgeführt werden, z. B. Auswerfen eines Netzes, Fieren von Sonden, Radiosondenaufstieg etc.; Stationsdaten enthalten die Position des Schiffes, die Uhrzeit und Aktivitäten auf einer Station.

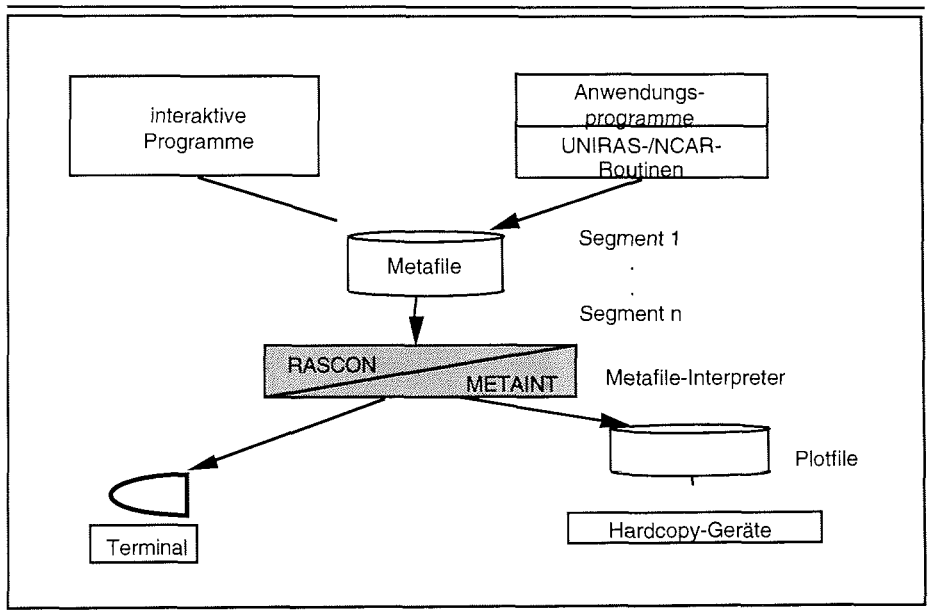


Abbildung 1.4: Schematische Darstellung der Arbeitsumgebung *Graphische Datenverarbeitung* im AWI. Graphiken werden in einem Metaformat *(Metafile)* beschrieben und von einem (interaktiven) Steuerprogramm *(Metaint)* auf die Ausgabegeräte verteilt.

Einschränkung auf wenige standardisierte Produkte wird derzeit angestrebt, dennoch ist der Benutzer mit den Systemen überfordert. Zentrales Problem ist die Behandlung der aus den Rohdaten erzeugten Bilddaten. Diese werden in Zwischenformaten, *Metafiles*, abgelegt (Abbildung 1.4). Ein spezieller *MetaFile*–Handler kann diese Dateien verwalten und gibt dem Benutzer die Möglichkeit, interaktiv die Daten seinen Erfordernissen entsprechend zu visualisieren, *ohne* daß er Kenntnisse über die Vorgänge im Rechensystem benötigt. In zwei Diplomarbeiten der Hochschule Bremerhaven wurde eine pseudo–graphische Benutzeroberfläche für den *MetaFile*–Handler entwickelt [PS89], die auch auf Textterminals lauffähig ist[12].

Bedingt durch die räumliche Trennung von Terminal, Rechenanlagen und Ausgabegeräten wird der Benutzer bei der Ausgabe der Daten vor die Frage gestellt, wo das Gerät, das das gewünschte Resultat liefert, steht und wie es anzusprechen ist. Die Möglichkeit, die Daten beliebig im hauseigenen Netzwerk zu verschieben und die einzelnen Geräte netzweit anzusprechen, sind Ursache mancher Benutzeranfragen (Abbildung 1.5). Die Verwaltung der zur Verfügung stehenden Betriebsmittel

[12]Die Software verwendet keine Graphikbefehle, um Masken darzustellen, sondern benutzt einfache Textausgabe und benötigt einen frei positionierbaren *Cursor*. Das Produkt wird zur Zeit auf eine neue Oberfläche unter Verwendung von *X–Windows* umgestellt.

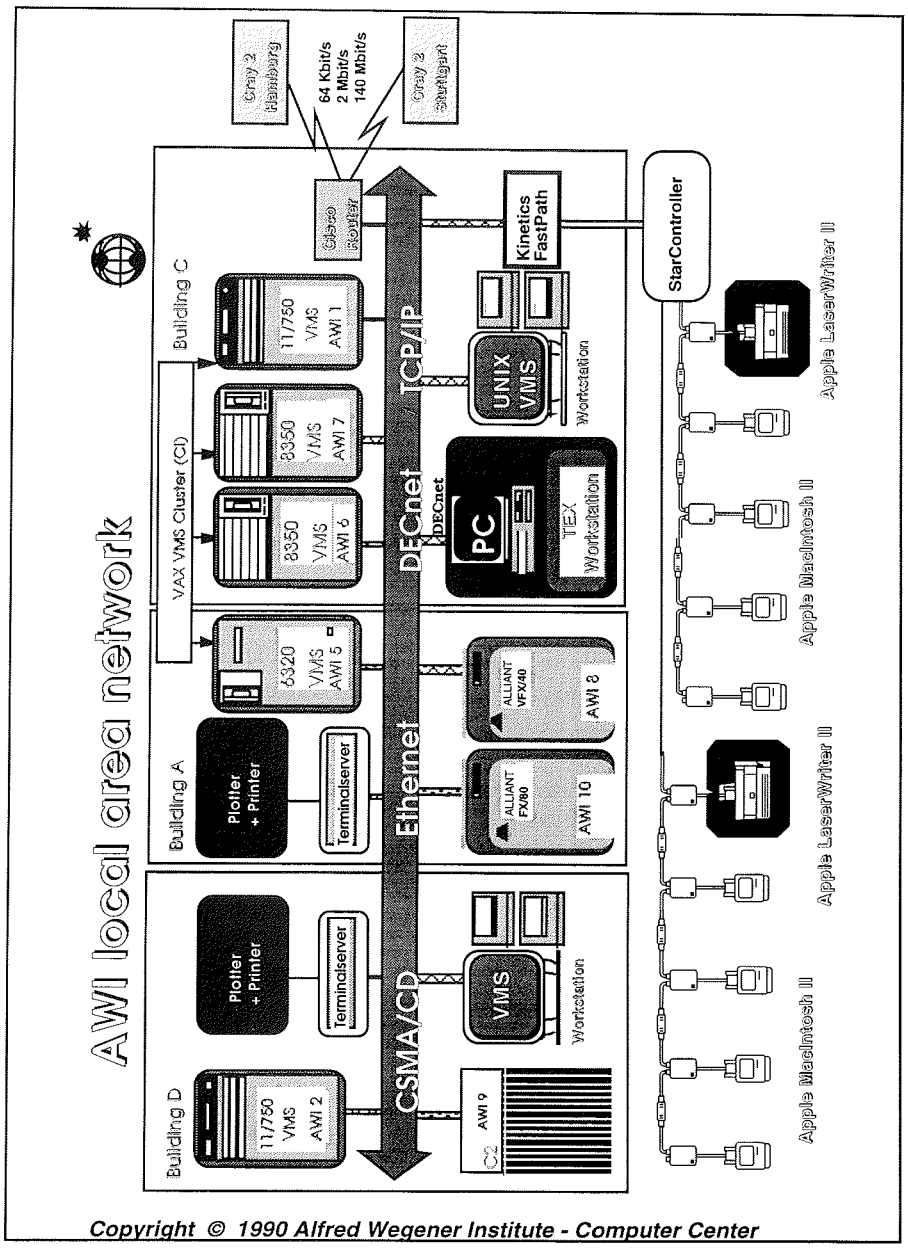


Abbildung 1.5: Rechnerkonfiguration des AWI (Stand 1988).

und eine umfassende Untersuchung der Benutzerwünsche, Benutzeranforderungen, Benutzerkenntnisse und Benutzererfahrungen sind das Ziel der vorliegenden Arbeit. An diesem *ResourcenManagementSystem* wurden die in dieser Arbeit vorgestellten Konzepte verifiziert.

1.4 Ziele und Ergebnisse der Arbeit

Die vorliegende Arbeit geht von der speziellen Situation der Wissenschaftler im AWI aus. Die bei der Benutzerbetreuung gemachten Erfahrungen werden zur Entwicklung einer benutzerorientierten Rechnerschnittstelle ausgewertet. Dabei werden Analysemethoden untersucht und Methoden zur Implementierung verglichen. Die Arbeit behandelt zwei wesentliche Aspekte. Auf der einen Seite stehen die Voranalysen, deren Ergebnisse aus dem Pflichtenheft, einem Regelwerk des Benutzerverhaltens und Parametern, die dieses Verhalten steuern, bestehen. Das Konzept zur Erfassung dieser Ergebnisse wird hier entwickelt. Auf der anderen Seite steht die Realisierung. Die Analyse und die Realisierung werden am Beispiel eines *ResourcenManagement-Systems* verifiziert. Dabei spielt insbesondere die Protokollierung der Benutzeraktionen eine wichtige Rolle. Auch für die Implementierung wird ein Konzept vorgestellt, so daß insgesamt eine Methode zur Generierung einer adaptierbaren Benutzungsoberfläche vorgestellt werden kann. Im folgenden wird der Weg zu diesem Konzept kurz beschrieben.

Die Erfassung und Strukturierung der benutzerrelevanten Daten erfordert ein Modell, ein Profil des Benutzers und Methoden zur Erfassung benutzerrelevanter Daten bezogen auf den Arbeitsablauf, d. h. genaue Kenntnisse über die Arbeitsweise der Benutzer. Hierzu zählt vor allem, welche Arbeitsmittel eingesetzt werden und wie eine Aufgabe bearbeitet wird.

In Kapitel 2 wird zunächst der Zustand der Rechenanlagen im AWI und die Situation der Benutzer vor Beginn dieser Untersuchung beschrieben. Die Darstellung ist gleichzeitig ein historischer Rückblick. Die Veränderungen, die durch Einführung neuer Hilfsmittel, wie Arbeitsplatzrechner, erfolgten, sind hier bewußt nicht berücksichtigt. In dieser Darstellung werden die Wünsche der Benutzer zusammengetragen, die diese von Hardware und Software erwarten. Die allgemeinen Anforderungen an einen ergonomischen Arbeitsplatz werden gezeigt. Außerdem wird an zwei Beispielen gezeigt, wie mit herkömmlichen Mitteln der wissenschaftliche Arbeitsplatz auf dem Rechner gestaltet werden kann. Eine Beispielanwendung, der *TEX-DeskTop*, diente auch zur Verifikation von in dieser Arbeit entwickelten Konzepten.

Um diesen Anforderungen gerecht zu werden, wird ein konzeptioneller Ansatz zur Analyse der Arbeit mit dem Rechner am wissenschaftlichen Arbeitsplatz entwickelt. In Kapitel 3 wird ein Überblick über Ansätze zur Analyse und Modellierung des Benutzerverhaltens gegeben. Neu in dieser Arbeit ist die Kombination von Verfahren des Softwaredesigns mit Verfahren zur Benutzermodellierung in einem Gesamtkonzept (Kapitel 4). Das System wird mit *RFA/ATN*-Netzen beschrieben, das Benutzerverhalten mit einem modifizierten *GOMS**-Modell analysiert. Dieses

neue *GOMS***–Modell verfügt über eine Methode, Wissen direkt in der Spezifikation eines Objektes zu plazieren. Die Bewertung und der Test des zu entwickelnden Systems erfolgen über die Auswahl zweier Benutzergruppen. Eine Gruppe, die Zielgruppe, liefert relevante Daten des Benutzerverhaltens und eine Aussage über die Anwendbarkeit des Konzeptes. Die zweite Gruppe, die Kontrollgruppe, dient zur Verifikation des Konzeptes.

Zunächst wird ein Verfahren entwickelt, um das Benutzungsmodell aufzustellen. Auf der Basis von Stereotypen oder Handlungsplänen kann das Verhalten des Benutzers im System beschrieben werden. Weiterhin werden Objekte, die zur Unterstützung des Systemverhaltens benötigt werden, definiert. Dieses Verfahren erleichtert, gemeinsam mit objektorientierten Implementierungsmethoden, die Beschreibung des Verhaltens und der Struktur und die Implementierung.

Zur Unterstützung des Benutzers bei der Bearbeitung von Aktionsplänen wird eine Mitschrift seiner Dialoghistorie bereitgehalten. Die Dialoghistorie wird in dieser Arbeit in Kapitel 5 formal als abstrakter Datentyp betrachtet. Neu ist die kombinierte formale Betrachtung von technischen Datenstrukturen und deren Nebeneffekten im System. Ebenso wird die Generierung von Makros aus der Dialoghistorie beschrieben. Systembefehle werden in Metabefehle überführt und analysiert. Dazu zählt die Beseitigung redundanter Befehle, Ersatz von Symbolen in Parametern und Erkennung von Parametern. Ein Zusammenhang zwischen der hier definierten Sprache zur Beschreibung von Systembefehlen und dem *GOMS***–Ansatz wird hergestellt.

Auf der Basis der vorliegenden Ergebnisse wird ein geeignetes System zur Implementierung ausgewählt. Im Vorfeld der Arbeit gab es bereits deutliche Hinweise, daß ein System, das dem Benutzer bei der Navigation in einem Rechnernetz behilflich ist, von den Benutzern bevorzugt wird und das als *ResourcenManagementSystem* bezeichnet wird. In Kapitel 6 wird die Auswahl dieses Systems, die Festlegung der Zielgruppe und der Kontrollgruppe beschrieben. Die Auswahl und der Vergleich von Werkzeugen kann erst hier nach Erstellung des Gesamtkonzeptes erfolgen. Die Methode der Analyse wird am Fallbeispiel erläutert und verifiziert. Die Implementation der Dialoghistorie und der Planerkennungsmethode wird an Beispielen erläutert.

Ein Ausblick über die Zukunft des Systems und die Entwicklung im AWI vervollständigt die Arbeit. Im Anhang sind einzelne Beispiele der Analysen, Implementierungsdarstellungen und ein weiteres im Rahmen dieser Arbeit entwickeltes System zu finden, das Anwender des Textsatzprogramms TEX bei der Dokumentbearbeitung unterstützt *(TEX–DeskTop)*.

Insgesamt konnte gezeigt werden, daß die Verbindung von geeigneten Analysemethoden (*RFA*–Netze, *GOMS***–Analyse, objektorientierte Analyse) in der Lage ist, eine umfassende Aufnahme der Benutzersituation am wissenschaftlichen Arbeitsplatz durchzuführen und ein Verfahren zur Implementierung eines Systems bereitzustellen. Die Verbindung dieser Analysen mit der Dialoghistorie sind durch die implizite Möglichkeit, Verfahren zur automatischen Erfassung der Benutzersituation (Adaptivität) zu integrieren (objektorientierter Ansatz), zukunftsweisend. Die Dia-

loghistorie selbst kann, durch Anpassung der Voraussetzungen, sowohl das Modell von Herczeg als auch die Modelle von Archer, Conway und Schneider sowie von Vitter ersetzen.

```
Bitte geben Sie eine Zahl
zwischen 1 und 10 ein: 2
Danke.
Und jetzt kommt der Trick...
FATAL ERROR...
REGISTER OVERFLOW AT AF45
712 547 234 232
777 234 342 455
209 487 439 332
>
```

John Shore

Kapitel 2

Der wissenschaftliche Arbeitsplatz am AWI

Die Gruppe der wissenschaftlichen Mitarbeiter des AWI setzt sich aus den Disziplinen Biologie, Physik, Geologie/Geophysik, Meteorologie, Ozeanographie und Chemie zusammen. Die Wünsche, Aufgabenstellungen und zu verarbeitende Daten dieser Gruppen sind heterogen. Die Situation eines zentralen Dienstes, im Umfeld dieser Untersuchungen der *Zentralbereich Datenverarbeitung* des AWI (Rechenzentrum des AWI/ZBDV), wird mit zunehmender Benutzerzahl in immer stärkerem Maße von der Unerfahrenheit der Benutzer im Umgang mit dem Rechner gekennzeichnet [Wei87]. Waren die Rechner einst überwiegend zur Lösung numerischer Probleme eingesetzt, so steht heute die Verwaltung von Daten im Vordergrund.

Im historischen Rahmen wurde durch die Dezentralisierung der Rechenleistung auf dem Apple II und dem IBM PC mit der Unterstützung durch interaktive Hilfsmittel (z. B. BASIC Interpreter) die elektronische Datenverarbeitung für größere Anwenderkreise interessant. Dabei müssen sich die neuen, rechnerunerfahrenen Anwender mit alten Techniken der Datenverarbeitung vertraut machen (z. B. Befehlszeilen unter MSDOS). Eine problemfreie Bearbeitung der Aufgaben wird durch die schwer zu erlernenden Befehlssätze und Eingaberegeln erschwert. Die Rechenzentren sind gefordert, den Benutzer bei seinem täglichen Umgang mit dem neuen Arbeitsmittel zu unterstützen. Eine progressive Unterstützung des Benutzers, d. h., Auswahl neuer Betriebsmittel und Entwicklung oder Konzeption von Applikationen, ist kaum möglich. Die Entwicklung einer Datenbank für das AWI legte diese Pro-

bleme offen dar [Wei87]. Um eine zufriedenstellende Betreuung des Benutzers durch seinen *Kollegen* Computer bereitzustellen, sind Grundlagen für einen effektiven Aufbau einer Benutzerschnittstelle zu schaffen. Eine speziell angepaßte Schnittstelle zur Unterstützung der Benutzer im wissenschaftlichen Bereich des Hahn–Meitner–Institutes wird von Fromme beschrieben [Fro87].

Bei den vorliegenden Arbeiten fehlt jedoch immer die Einbeziehung des Benutzers in das System in Form eines Benutzungsmodells, das der Schnittstelle Informationen über Arbeitsabläufe und Zusammenhänge geben kann. Als Ergebnis der vorliegenden Analyse wird ein Benutzerprofil erstellt und daraus eine adaptive (mindestens adaptierbare) Benutzerschnittstelle abgeleitet. Daher muß geprüft werden, inwieweit sich die Ergebnisse der bisherigen Forschungsarbeiten auf den wissenschaftlichen Arbeitsplatz übertragen lassen.

Die Darstellung geschlossener Modelle als Grundlage einer Benutzerschnittstelle wurde für die Benutzer im Verwaltungssektor (Banken, kaufmännischer Bereich etc.) durchgeführt [TFK87, KHTF87a]; die Software für diesen Anwendungsfall befindet sich im Prototypenstadium. Im wissenschaftlichen Sektor jedoch fehlt diese Betrachtung.

Aus strukturellen Gründen können im AWI keine großflächigen psychologisch und soziologisch orientierten Untersuchungen durchgeführt werden. Es wird statt dessen nötig sein, die in der Literatur verbreiteten Ergebnisse stichprobenartig zu verifizieren. Fragestellungen wie „Was will der Benutzer machen?“, „Wie will der Benutzer seine Daten verarbeiten?“ und „Welche Hilfsmittel benötigt er?“ werden die Arbeit beherrschen.

2.1 Funktionsbezogene Forderungen

Die in diesem Abschnitt dargestellten Forderungen an ein Rechnersystem im AWI sind aus Diskussionen und Befragungen der Benutzer hervorgegangen. Sie stellen für die einzelnen Fachbereiche die Basisanforderungen dar, die erfüllt werden müssen, damit Unterstützung überhaupt erst einmal gewährleistet werden kann. Dabei wird zwischen den Anforderungen der Benutzer im allgemeinen und sich daraus ergebenden speziellen Anforderungen an Hard– und Software unterschieden.

2.1.1 Anforderungen der Benutzer

Die Anforderungen der Benutzer an den Rechner werden zunehmend komplexer. Eine einfache Schnittstelle zum Rechner ist nicht mehr ausreichend. Das Ausgabemedium muß Anforderungen wie Graphik, Auswahlmenüs, Ein– und Ausgabemasken etc. genügen. Sind diese Anforderungen bei einem Einzelrechner, z. B. einem PC, noch recht gut zu erfüllen, so entstehen bei Mehrplatzsystemen Probleme. Diese die Interaktion des Benutzers mit dem Rechner betreffenden Probleme lassen sich durch Betrachtung der Anforderungen an Hard– und Software eines Rechners oder Rechnerverbundes analysieren.

Die Hardware eines Rechners nimmt durch die hier aufgeführten möglichen Bedingungen auf die Interaktionsmöglichkeiten des Benutzers entscheidenden Einfluß[1] (entnommen aus [H+87]):

- Rechenleistung vor Ort, d. h. Dezentralisierung und offener Benutzerzugang
- Terminals
  - einfaches Textterminal (z. B. VT220)
  - langsames Graphikterminal (z. B. Tek4107)
  - langsamer Anschluß an den Rechner (z. B. RS232 mit 9600 Baud)
  - Graphik–Terminal mit *X11*–Protokoll
  - Workstation mit LAN–Anbindung (Übertragungsrate >10MBit)
- Mehrere Rechner in einem Netz
  - Protokollvielfalt z. B. TCP/IP, DECNet
  - unterschiedliche Betriebssysteme
  - Anwendungsschicht auf höheren Ebenen z. B. ARPA–Dienste, telnet, ftp
  - Zugriffe innerhalb des Netzes (LAN, WAN)
- *client/server*–Prinzip
  - Database–Server
  - Compute–Server
  - *Disk*– und *Tape*–Server
  - Graphik–Server
  - Backup–Server
- Ausgabegeräte an mehreren Rechnern
- Räumliche Trennung von Rechner, Terminal und Ausgabegerät
- Verschiedene Kommunikationswege, z. B. Terminalleitung, Netzwerkverbund
- Rechnen auf nicht im Haus vorhandenen Anlagen

Ist die Hardware an die Bedürfnisse des Benutzers angepaßt, so kann immer noch ein Engpaß durch die Software entstehen, wenn

[1] In diesem Kapitel wird der Zustand zu Beginn der Arbeiten beschrieben. Zwischenzeitlich sind nahezu alle Arbeitsplätze mit Arbeitsplatzrechnern, i. e. DECStation, VAXStation, Apple MacIntosh II, SUN etc., ausgestattet, die nahezu die geforderte, hier aufgeführte Funktionalität unterstützen.

- die Software nur Textterminals unterstützt,
- die Software lange Antwortzeiten hat,
- die Software nicht alle Eingabemöglichkeiten ausschöpft (z. B. Graphiktablett),
- die Software zwar Graphik unterstützt, aber durch veraltete Konzepte keine graphische Oberfläche bietet,
- verschiedene Formate zur Datenspeicherung eingesetzt werden,
- unterschiedliche mathematisch/statistische Bibliotheken Verwendung finden,
- unterschiedliche Graphikstandards berücksichtigt werden müssen.

Standardsoftware, wie sie vom Benutzer im AWI gewünscht wird, ist im folgenden aufgeführt:

- mathematisch/statistische Bibliotheken, die übliche Standards unterstützen, z. B. NAG, IMSL, CERN
- Graphikstandards, z. B. GKS, PHIGS, CGM
- Standardprogrammiersprachen, z. B. FORTRAN, C
- leistungsfähiger Editor, z. B. LSE von Digital Equipment
- standardisierte Graphiksoftware, z. B. UNIRAS
- einfache Bedienung der Ausgabe, z. B. Queue–Management
- einfache Bedienung der Eingabe, z. B. Datenbank
- einfaches Kopieren von Dateien zwischen Rechnern, z. B. FTP
- verteilte Datenhaltung, z. B. NFS
- einfaches Bearbeiten von Graphikdateien, z. B. MetaFile–Handling
- vom Betriebssystem unabhängiges Agieren im System
- transparentes Arbeiten im Netz, z. B. *ResourcenManagementSystem*

Die Tabelle 2.1 (Seite 19) gibt, geordnet nach Fachbereichen, Auskunft über die Anforderungen der Benutzer an einen Arbeitsplatz im AWI.

In Tabelle 2.2 (Seite 20) und Tabelle 2.3 (Seite 22) werden im Gegensatz zur Forderungssammlung in Tabelle 2.1 die Anforderungen in ein Rechnersystemkonzept umgesetzt. Die Komplexität der Abbildung 1.5 (Seite 10) wird so etwas deutlicher.

Aber gerade diese Komplexität, verbunden mit der für Rechnerneulinge schwer erlernbaren Syntax der Software und des Gesamtsystems, kann durch diese Analyse erfaßt werden.

In den folgenden Abschnitten sollen die Anforderungen an Hard– und Software dargestellt werden, die aus den von den Benutzern vorgetragenen Wünschen entstanden sind.

2.1.2 Funktionsbezogene Forderungen: Hardware

Die funktionsbezogenen Anforderungen an die Hardware (Tabelle 2.2) sind durch die notwendigen Arbeiten des Benutzers zur Auswertung gewonnener Daten gegeben. Im Grundsatz ist ein ständig wachsendes Datenvolumen zu betrachten. Im Einklang damit steht die Forderung nach immer schnelleren Rechnern und mehr Speicherkapazität, um diese Datenvolumina auch zu handhaben bzw. auf ein sinnvolles, verwaltbares Volumen zu reduzieren.

Die theoretischen Modelle zur Erklärung und Vorhersage von Eigenschaften, Entwicklungen und Phänomenen in den Ozeanen und in der Atmosphäre werden zu hochauflösenden Modellen der Vorgänge (z. B. wirbelauflösende Modelle) weiter entwickelt. Speziell in diesem Bereich ist, begründet durch eine hohe Anzahl von Iterationen gleicher Rechenschritte über weite Darstellungszeiträume, ein Vektorrechner und/oder Parallelrechner erforderlich.

Um Rohdaten und bearbeitete Daten zu verwalten, ist der Einsatz einer schnellen Datenbank notwendig geworden. Um eine genügende Unterstützung der Datenbank von Seiten der Rechenleistung zu gewährleisten, ist auch hier der Einsatz schnellerer Prozessoren und großer Datenspeicher sinnvoll.

Der Wissenschaftler an seinem Arbeitsplatz wird durch den Einsatz von Farbgraphik–Terminals bzw. Workstations unterstützt. Über diese Terminals kann er sich auf verschiedenen Rechnern des Instituts bewegen, auf ihnen rechnen und Daten abfragen. Auf dem Arbeitsplatzrechner können kleinere Aufgaben wie Datenerfassung, Textverarbeitung und Verwaltung kleiner Datenmengen durchgeführt werden.

Aus einer kleinen Rechenanlage ist inzwischen ein heterogenes Netz aus verschiedenen Arbeitsplatzrechnern, Mainframes und Vektorrechnern entstanden, die sich teilweise im Hause, teilweise außer Haus befinden und zu denen schnelle Datenleitungen geschaltet sind. Durch den Einsatz von standardisierten Netzwerken (Ethernet) und den Einsatz von standardisierten Protokollen, die von allen Maschinen eingehalten werden müssen (TCP/IP, DECNet), wird versucht, den Zugriff zu homogenisieren. Bei den Problemen des Benutzers bei der Bewältigung seiner Aufgaben konnte durch Spezial–Software auf Mini– und Mikrorechnern geholfen werden. Der hohe Vernetzungsgrad mit den damit verbundenen Möglichkeiten erzeugt jedoch immer neue Probleme.

Die Betrachtung der Hardware liefert Erkenntnisse über die Einsatzmöglichkeit von Dialogtechniken. So ist z. B. im interaktiven Betrieb des Hochleistungsrechners *Cray* keine Fensteroberfläche möglich, da zum einen *Cray* dieses noch nicht un-

terstützt[2], zum anderen durch den hohen Datenfluß der Betrieb durch die Kosten der Datenübertragung zur Zeit unerschwinglich wird.

Tabelle 2.1: Anforderungskatalog eines Wissenschaftlers im AWI an einen rechnergestützten Arbeitsplatz

Forderung	Anwendergruppe					
	BIO	PHY	BAT	CHE	GLA	OZE
Textverarbeitung	x	x	x	x	x	x
Datenbank						
– geringe Datenmengen	x			x		
– große Datenmengen		x	x		x	x
– wenige Relationen		x	x	x	x	x
– viele Relationen	x					
Statistik	x	x	x	x	x	x
– deskriptive Analysen	x			x		
– Cluster–Analysen	x					
Graphik	x	x	x	x	x	x
– Tabellen	x			x		
– Isolinien		x	x		x	x
– 2–D Darstellung	x	x		x	x	x
– 3–D Darstellung			x			
Rechenleistung						
– einfache Rechner	x			x		
– Datenverwaltung	x	x		x	x	x
– Numerik		x	x		x	x
– Vektorrechner		x				x
Mailing	x	x	x	x	x	x
Rechnen außer Haus		x				x
Vernetzung						x
– Rechnernutzung		x	x			x
– Mailing	x	x	x	x	x	x

	Erläuterungen zur Tabelle
BIO	Biologie
PHY	Meeresphysik/Geophysik
BAT	Bathymetrie
CHE	Chemie
GLA	Glaziologie
OZE	experimentelle/theoretische Ozeanographie

[2]Für die aktuelle Version von UNICOS, dem Betriebssystem der *Cray*, existieren inzwischen fensterorientierte Werkzeuge, die jedoch nicht den in der vorliegende Arbeit diskutierten Kriterien genügen.

Tabelle 2.2: Anforderungen an ein Rechnersystem zur Unterstützung der Benutzerforderungen im Bereich Hardware

Forderung	Anwendergruppen						
	BIO	PHY	BAT	CHE	GLA	OZE	ZBDV
Hochleistungrechner		x	x			x	
– Vektorrechner		x				x	
– Parallelrechner		x				x	
schnelle Vorrechner		x	x		x	x	
Workstation		x	x		x	x	x
PC	x			x			x
Datenbankprozessor							x
Plattenkapazität							
– große Datenmengen	x	x	x		x	x	x
Vernetzung der Rechner							
– untereinander		x	x		x	x	x
– Datex–P		x	x		x	x	x
– 64 KBit		x				x	x
– > 2 MBit		x				x	x
Graphik	x	x	x	x	x	x	
– Liniengraphik	x	x	x	x	x	x	x
– Farbgraphik	x	x	x	x	x	x	x
– großformatig		x	x		x	x	x
– hochauflösend			x				x
– Diaproduktion		x	x		x	x	x
– Filme		x				x	x
Graphikterminal	x	x	x	x	x	x	x
– hochauflösend		x	x				x
– schneller Bildaufbau		x	x		x	x	x
– Animation		x				x	x

	Erläuterungen zur Tabelle
BIO	Biologie
PHY	Meeresphysik/Geophysik
BAT	Bathymetrie
CHE	Chemie
GLA	Glaziologie
OZE	experimentelle/theoretische Ozeanographie
ZBDV	Zentralbereich Datenverarbeitung

2.1.3 Funktionsbezogene Forderungen: Software

Die Anforderungen für Software (Tabelle 2.3) werden ebenfalls durch die Arbeitsmethoden und die daraus erwachsenden Forderungen bestimmt. In allen Bereichen ist die Erkenntnis vorherrschend, daß ein Bild mehr sagt als tausend Worte. Die ausgedehnten und verschiedenartigen Graphikanforderungen verdeutlichen dies.

Die Datenbank wird im Rahmen des Datenbankprojektes im AWI erstellt, dessen Grundlagen die Arbeiten von Weiß und Westerwick sind [Wei87, Wes87]. Die unterschiedlichen Datenvolumina und die verschiedenen Relationen und Abfragen waren mit Standard Datenbankanwendungssystemen, die insbesondere auf einem PC lauffähig sein sollten, nicht möglich. Es wurde daher nach einem schnellen Datenbanksystem gesucht, das auf nahezu allen Rechnern verfügbar ist und das eine einheitliche Oberfläche bietet.

Statistiken experimentell gewonnener Resultate werden teilweise auf PC's erstellt, teils auf Großrechnern ermittelt. Die unterschiedliche Behandlung ist zum einen an den zu bearbeitenden Datenmengen, zum anderen durch die Verfügbarkeit leistungsfähiger und leicht zu bedienender Programme begründet. Die Unterstützung bei der Erstellung von Graphiken durch die meisten PC–Programme favorisieren diese Klasse von Rechnern.

Die Programmentwicklung erfolgt, abhängig von Standards und Funktionalität, mittels älterer, unhandlicher Programmiersprachen wie FORTRAN oder einfach zu lernender Sprachen wie BASIC. Viele Entscheidungen für eine Programmiersprache fallen auch aus strukturellen und finanziellen Gründen. FORTRAN ist vielfach verbessert worden, und viele gute Bibliotheken für den Wissenschaftler wurden in dieser Sprache erstellt.

Die Textverarbeitung ist im wesentlichen auf das Schreiben von Veröffentlichungen beschränkt. Die Vorgaben in diesem Bereich werden meist von den Verlagen gegeben, so daß häufig mit einfachen Textverarbeitungsprogrammen das Ziel erreichbar ist. Jedoch setzen sich hier mehr und mehr die DTP–Programme *(desktop publishing)* durch, d. h., es erfolgt rückwirkend eine Voraussetzung für die Auswahl der Rechnerhardware.

Bei der Auswahl von Software ist insbesondere darauf zu achten, daß die Programme auf den verschiedenen Rechnern ablauffähig sind, bzw. daß zumindest ein Datenformat zum Datenaustausch existiert. Die bisher verfügbaren Softwarepakete unterstützten den Benutzer zwar mit großer Leistung, ließen aber beim interaktiven Einsatz viele Fragen offen, d. h., sie stellten Einzellösungen mit schlechter Integrationsmöglichkeit dar.

Tabelle 2.3: Anforderungen an ein Rechnersystem zur Unterstützung der Benutzerforderungen im Bereich Software

Forderung	Anwendergruppen						
	BIO	PHY	BAT	CHE	GLA	OZE	ZBDV
Graphik	x	x	x	x	x	x	
– interaktiv	x			x			
– CAD Ansprüche		x	x			x	
– Projektionen			x				
– Balkendiagramme	x			x			
– Isoflächen			x				
– Isolinien		x				x	
– Seismik		x					
– Liniengraphik					x		
– Statistiken	x			x			
Numerik		x	x		x	x	x
– mathematische Lib's		x				x	
– spezielle Routinen		x	x		x	x	
Statiktik	x			x			
– deskriptive Statistik	x			x			
– Clusteranalyse	x						
– interaktive Programme	x			x			
Datenbank	x	x	x	x	x	x	
– viele Relationen	x						
– einfache Relationen		x	x	x	x	x	
– kleine Datenmengen	x			x			
– große Datenmengen	x	x	x		x	x	
– einfache Abfrage	x	x	x	x	x	x	
– SQL o. ä.		x				x	
Textverarbeitung	x	x	x	x	x	x	
– Textedieren	x	x	x	x	x	x	
– *desktop publishing*	x	x	x	x	x	x	

	Erläuterungen zur Tabelle
BIO	Biologie
PHY	Meeresphysik/Geophysik
BAT	Bathymetrie
CHE	Chemie
GLA	Glaziologie
OZE	experimentelle/theoretische Ozeanographie
ZBDV	Zentralbereich Datenverarbeitung

2.2 Allgemeine Zielsetzungen

Die Forderungen, Programme, Programmiersysteme und Arbeitsplätze ergonomisch, d. h. am Benutzer orientiert, zu gestalten, sind aus den Erfahrungen entstanden, die während des langen Nutzungszeitraumes von Rechenanlagen gesammelt wurden. Schon früh (Anfang bis Mitte der sechziger Jahre) wurde über die Interaktion des Operateurs mit dem Rechner nachgedacht. Während zunächst nur die technische Entwicklung der Rechenanlagen vorangetrieben wurde, entstand bei ständig wachsender Nutzung der Anlagen eine Unzufriedenheit der Benutzer, die auch mit dem Begriff *Softwarekrise* umschrieben wird. Heute steht fest, daß die Betriebsmittel und Leistungsgrenzen einiger Maschinen von Applikationen nur unvollständig oder überhaupt nicht ausgenutzt werden (z. B. Entwicklung des Betriebssystems OS/2 unter Berücksichtigung der Leistungsfähigkeit des eingesetzten Prozessors). Problemorientierte Programmiersprachen (FORTRAN, ALGOL, COBOL, PASCAL, SIMULA) sowie die Einführung neuer Dialoggeräte (Videoterminals) öffneten dem Entwickler und dem Benutzer den Weg zu neuen Konzepten des Arbeitsablaufs und der Arbeitsplatzgestaltung. Die Entwicklung von Arbeitsplatzrechnern und Personalcomputern beschleunigte diesen Prozeß.

Die wissenschaftliche Diskussion über die ergonomische Gestaltung der Software und des Arbeitsplatzes wurde auf Fachkonferenzen (IBM Scientific Computing Symposium 1965) und in Gremien (CHI; INTERACT) geführt. Kennzeichnend für diese Konferenzen und das Forschungsgebiet der *Softwareergonomie* ist die interdisziplinäre Betrachtung der Problematik. Dazu zählt auch der wachsende Bedarf an Informationen über mentale Vorgänge der Benutzer (Psychologie) und soziales Verhalten der Benutzer (Soziologie) am computerunterstützten Arbeitsplatz. Diese Zusammenhänge werden in dem Versuch, das Benutzerverhalten quantitativ zu erfassen [CMN83], und bei der Beschreibung der Benutzerkomplexität deutlich [KP85]. Eine ausführliche Darstellung ist bei Streitz zu finden [Str89].

Zur Einbeziehung des Benutzers in neue Programmierkonzepte ist es notwendig, Wissen über seine Arbeitsumgebung zu sammeln. Die Kenntnis, wie der Benutzer beim Lösen der Aufgaben agiert und welche Objekte der Benutzer bearbeitet, ist zwar immer notwendig gewesen, aber erst durch neue Techniken wird eine vernünftige Verwaltung und Nutzung dieses Wissens, z. B. durch Wissensbasen (Wissensdatenbanken), möglich. Der Aufbau und die Verwaltung von Wissensbasen erfolgt mit Methoden der *künstlichen Intelligenz (KI)*[3]. Programmiersprachen wie PROLOG und *Smalltalk-80* sowie Erweiterungen von LISP (Flavours, LOOPS, ObjTalk) sind leistungsfähige Werkzeuge zur Erstellung von Expertensystemen für Benutzerschnittstellen.

[3] *Künstliche Intelligenz* ist die unzureichende Übersetzung des angloamerikanischen *artificial Intelligence*, die sich aber eingebürgert hat. Schmalhofer definiert *Künstliche Intelligenz* als Suche „nach Verfahren, die zu einem intelligenten Verhalten eines Computers führt“ [SW88].

2.2.1 Allgemeine Anforderungen an die Hardware

Mit zunehmender Rechenleistung am Arbeitsplatz werden mehr und neue Einsatzmöglichkeiten geschaffen und genutzt, so z. B. die graphische Darstellung in Verbindung mit Menüs oder Fenstertechniken. Neue Einsatzmöglichkeiten schaffen aber auch wieder neue Forderungen, da sich der Benutzer schnell an die neu geschaffene Leistung gewöhnt und diese als Anforderungen bei der Erstellung eines neuen Systems formuliert. Die Entwicklungen auf dem Sektor der Benutzerführung (Software) und der Rechnersysteme (Hardware) sind nicht unabhängig voneinander.

Die Hardware eines Computers ist die direkte Schnittstelle zum Computer. Auch hier müssen bestimmte Kriterien erfüllt werden. Ergänzend zu den vorhergehenden Darstellungen werden hier Anforderungen beschrieben, die den Dialog des Benutzers *mit* dem Rechner betreffen und nicht Forderungen, die *aus* der Arbeit des einzelnen Benutzers entstehen.

Zunächst wird erfaßt, was der Benutzer heute tatsächlich fordert und wie sich die Forschung auf dem Gebiete der Ergonomie dazu stellt. Die heute verfügbare Hardware läßt leicht eine überschwengliche Meinung der Nutzbarkeit der neuen Hilfsmittel entstehen *(allgemeine Softwarekrise[4])*.

Eine gute Zusammenstellung der Forderungen ist bei Shneiderman aufgeführt [Shn85]. Danach sind folgende Punkte zu berücksichtigen:

1. *Tastaturdesign:* Die Eingabetastatur sollte den anatomischen Gegebenheiten des Menschen entsprechen.[5]

2. *Bildschirmanzeige:* Die Anzeige sollte hochauflösend sein mit einer hohen Bildwiederholrate, um den Benutzer nicht zu überanstrengen.[6]

3. *Antwortzeiten:* Die Antwortzeit des Systems im Dialogbetrieb sollte den Reaktionszeiten des Benutzers angemessen sein, d.h., recht klein sein.[7]

4. *Zeigeinstrumente (Erkennen und Zeigen):* Die textuelle Darstellung wird durch verständliche bildhafte Darstellung ersetzt[8].

5. *Spracheingabe und Sprachausgabe:* Dem Benutzer wird eine gewohnte Handlungsweise zur Verfügung gestellt.

[4]Die allgemeine Softwarekrise ist durch die Diskrepanz zwischen ständig wachsender Rechnerleistung und dem Unvermögen, diese Leistung mit entsprechenden Programmen wirklich zu nutzen, gekennzeichnet.

[5]Wichtig hierbei ist die bisher übliche unnatürliche Haltung der Hände bei Standardtastaturen.

[6]Die Wiederholrate sollte oberhalb von 60 Hz liegen. Texte müssen ohne Anstrengung lesbar sein.

[7]Es werden maximal 2 Sekunden zwischen Absetzen des Befehls und Antwort des Systems als optimal angesehen, wenn nicht der Benutzer durch die Aufgabenstellung eine längere Zeit erwartet. Beim heutigen Stand der Technik (z. B. das Zeichnen von Linien beim Erstellen von Graphiken) kann jedoch diese Zeit (2 s) bereits zu lang sein.

[8]Die Maus, der *TrackBall* und das Graphiktablett sind nicht das Optimum für die Dateneingabe. Hier müssen zusätzliche Möglichkeiten, die das Zeigen und Erkennen unterstützen, vorhanden sein, z. B. direktes Ansprechen durch Eingabe einer Befehlssequenz.

6. *Leise Geräte*

2.2.2 Allgemeine Anforderungen an die Software

Im Gegensatz zur leicht zu spezifizierenden Hardware gibt es bei der Software wesentlich mehr Details zu berücksichtigen. Hier sind die Modelle der Mensch–Maschine–Schnittstelle zu berücksichtigen, mit denen versucht wird, den Menschen und die Maschine als ein globales System zu betrachten [CMN83] oder das Verhalten des Benutzers zu modellieren [Shn85, CMN83, SIK+83]. Bei der Beschreibung dieser Spezifikationen muß zwischen den direkt erfüllbaren Forderungen, wie Einsatz von Bibliotheken oder der Basisfunktionalität und den Modellansätzen, die in Form von Wissensbasen, Planerkennern und lernfähigen Systemen realisiert werden, unterschieden werden.

Die Kriterien werden in folgende Kategorien eingeteilt:

- Software–Verhalten
- Menschliche Faktoren *(Human Factors)*
- Konsistenz und Hilfe

In der Kategorie Softwareverhalten sind die Modalitäten bei Fehlbedienungen, bei reproduzierbaren Ereignissen, bei Antwortzeiten etc. zu berücksichtigen. Shneiderman hat für den Designer sehr detaillierte Anweisungen niedergelegt, die, wenn sie berücksichtigt werden, eine zuverlässige Benutzungsoberfläche ergeben [Shn85]. Im einzelnen sind die folgenden Punkte zu berücksichtigen:

- *Verläßlichkeit:* Steht das Resultat in direktem Zusammenhang mit den erwarteten Ergebnissen?
- *Schutz vor Fehlbedienung:* Werden Fehlbedienungen von der Software erkannt und ist eine Fehlerbehebung durch den Benutzer möglich bzw. assistiert das System dem Benutzer bei der Behebung des Fehlers?
- *Ausführungzeit:* Ist die Antwortzeit des Programms angemessen und zumutbar?
- *Komplexität der Bedienung:* Gibt es bei der Bedienung der Software viele zu erlernende Einzelheiten, oder sind nur wenige Grundkenntnisse erforderlich, um einen ersten Erfolg zu garantieren?[9]

[9]Die Diskrepanz zwischen Leistungsfähigkeit eines Systems und leichter Erlernbarkeit führt zur Unterscheidung zwischen Experten– und Novizenmodus. Einige Systeme versuchen einen Kompromiß einzugehen, so daß der Lernaufwand überschaubar wird, die Leistung jedoch akzeptabel bleibt (vergleiche die Untersuchung von Ackermann [Ack84]).

Die Unterschiede bei einzelnen Personen und Personengruppen in Arbeitsverhalten, Gewöhnung und individuellen Auffassungen begründen den großen Umfang modellhafter Forderungen. In der angloamerikanischen Literatur wird vom *Human Factor* gesprochen (zweite Kategorie). Der Umfang der Kategorie des *Human Factors* macht die Schwierigkeit aus, ein Programmsystem zu schreiben, das jedem Benutzer gerecht wird. Die menschlichen Faktoren sind bis auf wenige quantitativ erfaßbar. Meistens muß eine subjektive Reaktion bewertet werden. Hier soll zunächst nicht zwischen qualitativen und quantitativen Merkmalen unterschieden werden. Die folgende Liste gibt die einzelnen *Human Factors* wieder:

- Einarbeitungszeit
- Antwortzeiten
- Fehlerrate des Benutzers
- Zufriedenheit des Benutzers *(satisfaction)*
- Zeitkonstante des Vergessens *(retention time)*
- Novize/normal Benutzer/Experten Modus

Außer den *Human Factors* wird das Verhalten der Software bei der Eingabe von Befehlen und beim Wechseln in einen anderen Arbeitskontext betrachtet. Dieser Wechsel ist bei den heutigen Arbeitsplatzrechnern eines der Hauptprobleme der Softwarebedienung. Herczeg definiert hier die Begriffe der *inneren* und *äußeren Konsistenz*, die hier als dritte Kategorie aufgeführt werden [Her86a]:

- *innere Konsistenz:* Verläßlichkeit des Systems, gleiches Verhalten bei gleichen Aktionszielen
- *äußere Konsistenz:* Verläßlichkeit der Benutzerschnittstelle über mehrere Anwendungssysteme hinweg

Konsistenz kann vom Benutzer in beschränktem Umfang durch Adaption des Systems an die Bedürfnisse erzielt werden. Die Adaption der Softwareumgebung durch den Benutzer oder durch den Systemmanager wird heute bereits durch leistungsfähige Hilfsmittel ermöglicht, z. B. *awk*-Scripte unter UNIX. Dieses kann von einfachen Einstellungen, wie Vergabe von Symbolen als Abkürzung komplexer Befehle bis zu umfangreichen Befehlsdateien führen. Systeme dieser Art heißen *adaptierbar*. Die im Gegensatz dazu automatisch einstellenden Systeme *(adaptive Systeme)* sind Gegenstand weitergehender Forschung [KHTF87b, TKH86, Ste84].

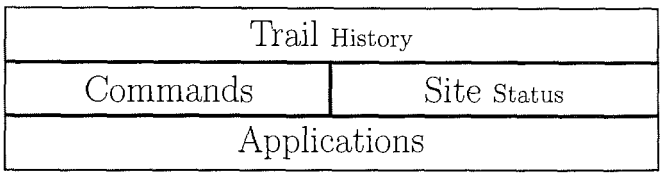


Abbildung 2.1: Die grundsätzliche Aufteilung eines Fensters des XS-2 Systems nach [Ste84]

2.2.3 Unterstützung, Hilfe und Protokolle

Wichtiger als die Beeinflussung des Systems ist die Fähigkeit eines Programms, in jeder Situation deutliche und angemessene Hilfe zu leisten. Hier muß zwischen automatischer (aktiver) und angeforderter (passiver) Hilfe unterschieden werden. Hilfe ist zu jeder Zeit, in jeder Situation kontextabhängig darzustellen. Aktive Hilfe kann störend wirken, wenn der Platz auf dem Bildschirm nicht ausgenutzt, (z. B. DATAEASE Datenbank mit automatischer Hilfe [Dat88]) oder falsch plaziert wird. Eine Ausnahme stellt das Beheben eines aufgetretenen Fehlers dar, sofern er *nur* durch einen Eingriff des Benutzers behebbar ist (→ Regel in einer Wissensbasis).

Ebenso sind Protokolle der Sitzung zu führen, die dem Benutzer einen Überblick über durchgeführte Aktionen bieten. Sie werden als *History* oder *Trace* bezeichnet. Diese Protokolle sind auch Grundlage für umfangreiche *UnDo/ReDo*–Aktionen, die dazu dienen, Bearbeitungsschritte, die zu nicht gewünschten Resultaten führten, rückgängig zu machen bzw. die Wiederholung einer Sequenz von Befehlen erlauben. Eine Diskussion der *UnDo/ReDo*–Aktionen ist von Herczeg erfolgt [Her86a]. Danach ist keine allgemeine *UnDo/ReDo*–Funktion möglich, sondern sie muß für den jeweiligen Anwendungsfall neu erstellt werden. Jedoch kann durch Einsatz von Vererbungsmechanismen ein Großteil der Basis–Funktionalität in einer übergeordneten Funktionsklasse implementiert werden *(objektorientiertes Design)*.

Die *UnDo/ReDo*–Funktionalität wird beispielsweise bei *Smalltalk-80* mittels eines Protokolls der Änderung des Quelltextes realisiert *(change–File)*[10]. Mit Hilfe dieser Datei kann im Falle eines Fehlers und folgendem Systemzusammenbruch der letzte gültige Zustand vor Auftreten des Fehlers wiederhergestellt werden [Dig86a, Gol84]. Stevlovsky legte bei seiner Implementierung *XS–2* (Abbildung 2.1) sehr viel Wert auf Protokolle von Aktionen [Ste84]. Sie dienen der Rückverfolgung der ausgeführten Aktionen, Speicherung als Macro und Wiederausführung. *UnDo* ist durch Rücknahme des ausgeführten Befehls implementiert. (Siehe auch [SSNB84].)

[10] Ähnliche Verfahren werden bei den Editoren auf DEC Rechnern *(journal file)* realisiert. Hier wird jeder Tastendruck abgespeichert und wird bei einer Wiederherstellung automatisch erneut ausgeführt.

UnDo/ReDo wird ausführlich in Kapitel 5 behandelt.

In beschränktem Unfang sollten auch Systemmeldungen mit angezeigt werden, wenn sie zur Verdeutlichung von Vorgängen dienen, z. B. wenn eine UNIX–Oberfläche das Betriebssystem versteckt, der versierte Benutzer aber die Systemmeldungen benötigt, um zu lernen oder um die Vorgänge zu verfolgen.

2.2.4 Modellierung

Die genannten Punkte berücksichtigen jedoch noch nicht das Benutzerverhalten am Rechner. Die Modellierung dieses Verhaltens ist Gegenstand intensiver Forschung und findet sich auch im Anlaß dieser Arbeit wieder. Ein sehr anschauliches und interessantes Modell ist bei Card, Moran und Newell nachzulesen [CMN83]. Hier wird der Mensch als Teil eines Gesamtsystems, des *Model–Human–Processors*, interpretiert, dessen Hardware (Teile des Gehirnes, Augen, Ohren, Arme, Hände) und Software (Ausführung der Reaktionen auf die Reize des ihn umgebenden Systems) ebenfalls mit Antwortzeiten, Fehlerraten etc. belegt werden können. Dieses einfache Modell macht die Interaktion Mensch–Maschine deutlicher als jedes andere Modell[11]. Um jedoch den Menschen innerhalb dieses Systems detaillierter beschreiben zu können, bedarf es weiterer Methoden, die ihn durch seine Arbeitsweise und sein Verhalten am Arbeitsplatz im Rahmen seiner Tätigkeit mit dem Werkzeug Computer beschreiben.

In der Literatur sind mehrere Ansätze erkennbar, den Benutzer und seinen Dialog mit dem Rechner zu erfassen. Die aufgelisteten Modelle sollen im folgenden näher erläutert und auf ihre Tauglichkeit zur Beschreibung des wissenschaftlichen Arbeitsplatzes geprüft werden. Die Modelle sind nach dem Erscheinungsdatum geordnet.

- Produktions–Systeme [KP85, New72]
- Syntaktisches/semantisches Modell [SM77]
- KeyStroke–Level Modell [CM80]
- Ebenenmodell [FvD82]
- Model–Human–Processor [CMN83]
- GOMS: Goals–Operators–Methods–Selectors [CMN83]
- User–Conceptual–Model [MaA85]
- Simulationsmodell [S$^+$86]
- Model von Balzert [Bal88]

[11] Die *Vermenschlichung* des Computers bzw. die maschinenhafte Darstellung des Menschen wird von mehreren Autoren kritisiert. Weizenbaum [Wei90] und Dreyfus [Dre72] kritisieren den Anspruch der *KI*, je einen Computer konstruieren zu können, der menschliches Verhalten zeigt. Dreyfus anerkennt jedoch die Leistung der *KI* in bestimmten Teilgebieten.

Tabelle 2.4: Charakteristika des Büroarbeitsplatzes

Arbeitsmittel	**Aktionen**	**Implementierung**
Schreibtisch	zentrale Verteilstelle	*Desktop*
Schreibmaschine/Textautomat	Texte bearbeiten	*DTP*
Aktenordner	Ordnen von Vorgängen	Ordner, Directory
Ablage	bearbeitete Vorgänge sichern	Datei, File
Brief	Kommunikation	Datei, Post, Mail
Kartei	Datenabfrage	Datenbank
Journale	Relationen darstellen	Graphik, *spreadsheet*
kaufmännisches Rechnen	Buchhaltung, Angebote	Spezielle Komplettlösungen
Kommunikation	Informationsaustausch	elektronische Post

- *GOMS** [Are89a]

Die verschiedenen Modelle entsprechen den Ansätzen der Forschung, ob Daten quantitativ erfaßt werden sollen (z. B. *KeyStroke–Level Model*) oder ob nach einer Beschreibung des Benutzerverhaltens gesucht wird (z. B. *GOMS*). Der Designer sollte sich bewußt sein, daß ein quantitatives Modell wie das *KeyStroke–Level Model*, das die Prozeßzeit des Benutzers am Rechner erfaßt, nicht bei der Entwicklung der Benutzungsoberfläche hilft, sondern Daten liefert, wie gut (oder schlecht) diese in der jeweiligen Bedienungsumwelt bedient werden kann. Ein Überblick über einige Modelle mit Beispielanwendungen wird von Hoppe, Tauber und Ziegler gegeben [HTZ86]. Wegen ihrer Relevanz werden die genannten Modelle in Kapitel 3 ausführlich erläutert.

2.3 Der wissenschaftliche Arbeitsplatz am AWI

Die Ergebnisse der bisherigen Forschung erlaubten Ansätze zu adaptiven Benutzerschnittstellen wie z. B. *X–Aid* [KHTF87b, HKST87][12]. Die Realisierung dieser Benutzerschnittstellen erfolgte jedoch meist für den Bürobereich *(office automation)*, der mit seinen festgelegten Handlungsabläufen gut zu analysieren und zu beschreiben ist. Gunzenhäuser et al. stellen eine anwendungsneutrale Benutzerschnittstelle für den Bürobereich vor [BBG+89]. Eine allgemein anerkannte Standardisierung der Arbeitsabläufe ist im Bürobereich erkennbar. Sie wird nach den speziellen Bedürfnissen eines Institutes oder einer Firma im Einzelfall angepaßt (z. B. [iD84b]). Insbesondere sind die Arbeitsmittel weitgehend festgelegt. (Siehe Tabelle 2.4.)

Auch Arbeitsvorgänge, z. B. Buchführung, Bilanzierung, sind festgelegt. Dabei werden die Daten dem Benutzer häufig mittels Bildschirmmasken präsentiert. Die Daten können verändert und dann gespeichert werden. Masken sind für Arbeitsabläufe mit sich ständig wiederholenden Transaktionen sinnvoll.

[12]Zur Zeit konzentriert sich ein Großteil der Forschung auf maschinelles Lernen.

Tabelle 2.5: Charakteristika des wissenschaftlichen Arbeitplatzes am AWI

Arbeitsmittel	**Aktionen**	**Implementierung**
Schreibtisch	zentrale Verteilstelle	Desktop
Textautomat	Texte bearbeiten	*DTP*
Graphiken	Graphiken erstellen	spezielle Graphikprogramme
Aktenordner	Ordnen von Vorgängen	Ordner, Directory
Ablage	Daten sichern	Datenbank, Datei
Kartei	Datenbankabfrage	Datenbank
Kommunikation	Datenaustausch	*LAN, WAN, Mail*
eigene Programme	Programmentwicklung	Compiler, Library
wissenschaftliches Rechnen	Statisik, allg. Numerik	spezielle Einzellösungen

Nach Hertweck und Söhr sind am wissenschaftlichen Arbeitsplatz, im folgenden *WAP* genannt, basierend auf einer Umfrage, folgende Basisdienstbereiche erforderlich [HS88]:

- Textverarbeitung
- Text/Graphikverarbeitung
- Kommunikation
- Datenverwaltung
- Integration der Daten
- wissenschaftliches Rechnen

Eine Zusammenfassung der Anforderungen an einen Arbeitsplatzrechner für Wissenschaftler ist in den *DFG-Kriterien zum Arbeitsplatzrechner für Wissenschaftler* zu finden [DFG89]. Eine qualitative Übereinstimmung zwischen dem Büroarbeitsplatz und dem *WAP* ist erkennbar. (Siehe Tabelle 2.5.)

In vielen Bereichen, z. B. Text-/Graphikverarbeitung, sind Parallelen erkennbar. Das Anfertigen eines Dokumentes, einer Veröffentlichung, die Erstellung einer Graphik und die Integration von Graphiken und Text erfolgt wie bei der entsprechenden Aufgabe im Bürobereich. Es existieren jedoch spezielle Anforderungen wie z. B. mathematischer Formelsatz, Isoliniengraphik usw.

Die Ablage von Daten erfolgt in einer äquivalenten Form wie im Büro. Der Inhalt einer Kartei ist unterschiedlich, die Handhabung über eine Datenbankschnittstelle dagegen gleich.

Die Arbeit im Bürobereich ist vom Einsatz kommerzieller Programme geprägt. Selten werden Programme durch den Anwender entwickelt. Der wissenschaftliche Anwender hingegen erstellt programmtechnische, spezielle Lösungen für die besonderen, eigenen Aufgaben.

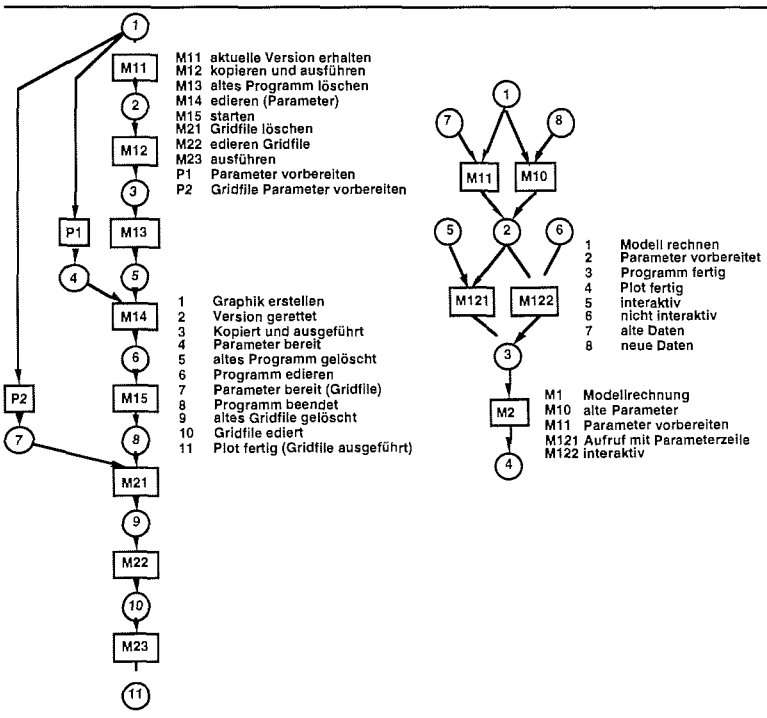


Abbildung 2.2: Zwei Benutzer modellieren Strömungsvorgänge im Ozean. Die Programmiertechniken und speziellen Modelle ergeben unterschiedliche Arbeitsabläufe, die hier abgebildet sind.

Die weitere Übereinstimmung wird durch die grundverschiedenen Arbeitsabläufe gestört. Bedingt durch die heterogene Zusammensetzung einer Arbeitsgruppe und die Individualität des Wissenschaftlers bilden sich nicht einmal innerhalb von Arbeitsgruppen gleichartige Arbeitsmethoden heraus. Die Arbeit ist von Arbeitsabläufen geprägt, die für eine begrenzte Zeit durchgeführt, z. B. Entwicklung eines Programms zur Berechnung einer speziellen Graphik, oder nur ein einziges Mal ausgeführt werden. In Abbildung 2.2 ist der Arbeitsablauf zweier Anwender einer Modellbildung (numerische Simulation) von Vorgängen im Ozean dargestellt. Während im Bürobereich die einzelnen Arbeitsabläufe standardisiert werden können, ist im wissenschaftlichen Bereich *a priori* keine derartige Standardisierung möglich.

Da nicht *jedes* Arbeitsmittel bereitgestellt werden kann, wird der Wissenschaftler gezwungen, sich auf eine verfügbare Menge zu beschränken. Diese Arbeitsmittel müssen dem Benutzer transparent gemacht werden, d. h., der Benutzer kennt Parameter des Systems, z. B. die Leistungsdaten der Maschinen, einige Besonderheiten, z. B. Beachtung des Vektorisierungskonzeptes des Vektorrechners, und die Netzwerk-

struktur. Bei genügender Transparenz benötigt der Benutzer keine spezialisierten Handbücher, um sein Ergebnis zu erhalten.

Bestimmte Methoden zur Behandlung von Daten können standardisiert werden. Dazu zählen Datenbankabfragen, Datenbankzugriffe, Graphikausgabe auf einem bestimmten Gerät sowie Kommunikation.

Eine dedizierte, durch Anwendung von Masken vorgeschriebene Arbeitsweise läßt sich aus den genannten Gründen nicht realisieren. Die Komplexität der Arbeitsbereiche und die Arbeitsabläufe der Benutzer werden durch diese Verfahren eingeschränkt.

Im folgenden wird nun versucht, den wissenschaftlichen Arbeitsplatz im AWI nach softwareergonomischen Gesichtspunkten neu zu gestalten. Ein System, das, ähnlich einem *integrierten Softwarepaket*, die Arbeitsmittel unter einer Oberfläche bereitstellt, ohne den Benutzer in seinem Arbeitsablauf zu behindern, wird entwickelt. Eine detaillierte Analyse der Benutzeranforderungen in einem ausgewählten Arbeitsbereich liegt zwei bereits eingesetzten Systemen, dem *MetaFile*–Handler [PS89] und dem *TEX–DeskTop* [KMP93], zugrunde. Zum besseren Verständnis werden diese Arbeiten zunächst kurz vorgestellt.

2.3.1 *WAP*–Beispiel 1: Der *MetaFile*–Handler des AWIs

Dem *MetaFile*–Handler liegen standardisierte Entwurfsmethoden der Programmentwicklung zugrunde [ANS84, WB84]. Das Pflichtenheft schreibt nachvollziehbar und verifizierbar die Applikation, Meilensteine in der Entwicklungsphase und Verhalten der Applikation fest. Thematisch wurden insbesondere die spätere Akzeptanz und Benutzerfreundlichkeit berücksichtigt.

Intensive Analysen der Erfahrungen der Benutzer im Umgang mit den existierenden Produkten am AWI beeinflußten die Auswahl der Entwickungswerkzeuge, das sichtbare Design und die Entwicklungsumgebung. Die Adaption an Benutzerpräferenzen konnte überzeugend eingearbeitet werden. Neben der Benutzerfreundlichkeit wurde insbesondere eine umfassende Funktionalität und die versteckte Nutzung bestehender Programme gefordert. Das System ist nicht dynamisch oder protokollierend.

Der *MetaFile*–Handler ist eine erste Realisierung einer menüorientierten, ergonomischen Benutzerschnittstelle, die auf den Rechnern im AWI eingesetzt werden kann. Benutzerwünsche gehen in Form von Funktionen ein. Einstellungen, die der Benutzer mittels Menüauswahl in mehreren Schritten bestimmt, werden durch Einträge in einer *setup*–Datei festgehalten.

Dieses Produkt ist von Erfahrungen beeinflußt, die mit dem Einsatz des Apple Macintosh II im AWI gemacht wurden. Die Analyse der Anforderungen erfolgte gemäß ANSI–Vorgaben [ANS84]. Die vollständige Beschreibung und das Pflichtenheft des *MetaFile*–Handlers sind der Arbeit von Plaschke und Schnars zu entnehmen [PS89]. Hier folgen einige wichtige Punkte aus dem Pflichtenheft, die im wesentlichen die am Benutzer orientierten Anforderungen berücksichtigen:

Mußkriterien

- Die Benutzerschnittstelle ist menüorientiert.
- Sämtliche Einstellungen in den Menüs können in einer speziellen *setup*–Datei des Benutzers abgelegt werden, so daß der Benutzer die Möglichkeit hat, sich z. B. mehrere Dateien mit unterschiedlichen Druckereinstellungen anzulegen.
- Beim Programmstart werden alle Einträge mit den Werten aus einer geladenen *setup*–Datei belegt. (Als *setup*–Datei wird die vom Benutzer zuletzt geladene Datei genommen; ist keine benutzereigene *setup*–Datei zu finden, wird die Grundeinstellung geladen.)
- Für die Ausgabe der Segmente eines graphischen Metafiles können verschiedene Optionen eingegeben/verändert werden, die das Layout betreffen *(viewport, window, fast options etc.)*.
- Für die Ausgabe kann der Benutzer aus den verfügbaren Graphikausgabegeräten auswählen.
- Für jeden Menüpunkt steht dem Benutzer eine passive, statische *Hilfe*–Funktion zur Verfügung.
- Die Benutzungsoberfläche wird so gestaltet, daß sie sowohl über Tastatur als auch über Maus bedient werden kann.

Wunschkriterien

- Es werden einige Betriebssystemfunktionen für die Dateiverwaltung zur Verfügung gestellt (z. B. *delete, rename*).
- Der Hilfetext erscheint wahlweise in Englisch oder Deutsch.
- Auf Wunsch kann der Bildschirm zweigeteilt werden *(splitting)*, um mehrere Segmente gleichzeitig nebeneinander darstellen und vergleichen zu können.

Abgrenzungskriterien

- Auswahl eines Metafiles
- Auswahl der Segmente durch Erstellung einer Segmentliste
- Druck eines Plotfiles
- Auswahl eines Druckers

- Verwaltung von benutzereigenen Setups (Laden, Sichern)
- *Hilfe*–Funktion
- Ausgabe von Informationen zum System
- Programmstart im Batch

Die zitierten Punkte umfassen die benutzerabhängigen Aufgaben *(user dependent tasks)*, die sich in der Schnittstelle repräsentieren lassen.

Dieses Pflichtenheft wurde aufgrund der Erfahrungen im Umgang mit den Benutzern bei der Handhabung der graphischen Ausgabegeräte, den beobachteten Problemen der Benutzer im Umgang mit der Graphiksoftware und den besonderen Wünschen der Benutzer erstellt. Nach diesem Pflichtenheft sind Grundforderungen an eine ergonomische Benutzerschnittstelle zu erfüllen: Menütechnik, Hilfemöglichkeit, halbgraphische Oberfläche. Die individuelle Arbeitsweise des Benutzers wird durch *setup*–Dateien berücksichtigt. Selektionsregeln oder Handlungspläne haben hier noch einen geringen Stellenwert.

Die erweiterte Version des *MetaFile*–Handlers basiert auf einem Datenbanksystem. In einer Datenbank werden die *setups* der Benutzer abgelegt. Druckaufträge des Benutzers werden in der Datenbank gesammelt und von einem zentralen Steuerprogramm *(jobserver)* abgearbeitet. Durch das *client/server*–Konzept des Datenbanksystems ist der Eintrag eines Auftrags von jedem Rechner im Netzwerk möglich.

Zusätzlich werden für jedes Ausgabegerät Privilegien vergeben, die einer Gruppe oder einem einzelnen Wissenschaftler zugeordnet werden. Ein Benutzer kann nur die Ausgabegeräte benutzen, für die er auch privilegiert ist.

In einer weiteren Entwicklungsstufe wird die Interaktion über *X–Windows* erfolgen.

2.3.2 *WAP*–Beispiel 2: Der *TEX–DeskTop* des AWIs

Ein weiteres Aufgabengebiet innerhalb des wissenschaftlichen Arbeitsplatzes ist die Dokumenterstellung. Im mathematisch–orientierten Bereich ist das Formatierprogramm TEX aufgrund seiner Leistungsfähigkeit beim Formelsatz ein häufig anzutreffendes Hilfsmittel. Eine PC–Version von TEX wird im AWI eingesetzt[13].

Der Wunsch nach einer einfach zu bedienenden Benutzungsoberfläche des gesamten TEX–Systems hat seine Wurzeln in den Erfahrungen, die bei der Textbearbeitung gemacht wurden. Der unübersichtliche Quelltext und der benutzerunfreundliche Aufbau der Handbücher waren ebenfalls Gründe für die Entwickung des *TeXEditors*. Das Edieren eines Textes und die Verwaltung der Dateien im Dateisystem des Rechners waren Grundanforderungen. Wunschkriterien sind Hilfsmittel für spezielle Eigenschaften von TEX, z. B. Formelsatz.

[13]Während der Ausführung der Arbeiten sind Versionen für den MacIntosh und SUN–Rechner hinzugekommen, die den Benutzer unterschiedlich unterstützen.

Daher wurde ein Steuerprogramm *TEX-DeskTop*[14] entwickelt, das die Aufgabe der korrekten Behandlung von TEX-Dateien und des Übersetzungsvorgangs übernehmen sollte. Dieses System diente gleichzeitig als Vorstudie zu der vorliegenden Arbeit. Die Anforderungsanalyse ergab folgende Forderungen:

- Integrierte Gesamtlösung basierend auf den verfügbaren Produkten,
- Einsatz eines frei wählbaren TEX-Compilers,
- Einsatz eines frei wählbaren *Preview*-Programms,
- Einsatz mehrerer Drucker und Druckertreiber,
- Integration des PC's in das lokale Netzwerk des AWI,
- netzweites Drucken,
- Unterstützung unterschiedlicher TEX-Formate und Makropakete,
- auf Formate und Makropakete angepaßter Editor.

Eine erste Realisation erfolgte in Smalltalk/V, die sich als leistungsfähig erwies und weiterentwickelt wird.

Der *TEX-DeskTop* wurde parallel zu dem in dieser Arbeit beschriebenen *ResourcenManager* entwickelt. Konzepte und Ideen wurden zwischen den Systemen übertragen. Ein erstes positives Ergebnis ist die Konsistenz in der Bedienung der beiden Systeme.

Beim *TEX-DeskTop* wurde im wesentlichen eine sinnvolle Unterstützung des Benutzers bei der Verwaltung der Dateien und beim Edieren des Dokuments realisiert (Abbildung 2.3). Das Betriebssystem wird vollständig vor dem Benutzer verborgen. Die Interaktion erfolgt über Menüs, Eigenschaftsfelder *(property sheets)* und Dialogboxen. Hilfe für den Benutzer wird z. Z. zu Menüauswahlpunkten angeboten. Ein *Online*-Handbuch ist in Vorbereitung.

Das System ist an verschiedene TEX-Makropakete adaptierbar. Spezielle Menüs und Prozeduren unterstützen bei der Eingabe von Befehlen oder Strukturen. Dazu zählten der Aufbau einer Tabelle, die strukturierte Eingabe von Listen und das Konzept der virtuellen Tastatur für die Eingabe von speziellen Zeichen und Symbolen. Der aktuelle Zustand des Systems wird benutzerbezogen gespeichert. Es werden *keine* Smalltalk-*images* gerettet.

Das System ist im Einsatz und wurde von den Benutzern mit einer guten Akzeptanz angenommen, da viele Funktionen, die sonst nur unter Konsultation des TEX/LATEX-Handbuchs ausführbar sind, in das System integriert wurden.

In mehreren Diplomarbeiten am AWI wurde dieses System erweitert. Eine Portierung auf den Apple Macintosh wurde von Makedanz durchgeführt [Mak90]. Zur Unterstützung des mathematischen Formelsatzes wurde ein *Formel-Editor* entwickelt [Pra90]. Weitere Angaben zum *TEX-DeskTop* sind im Anhang A.7 zu finden.

[14]Die Namensgebung entstammt der Arbeit von Makedanz [Mak90].

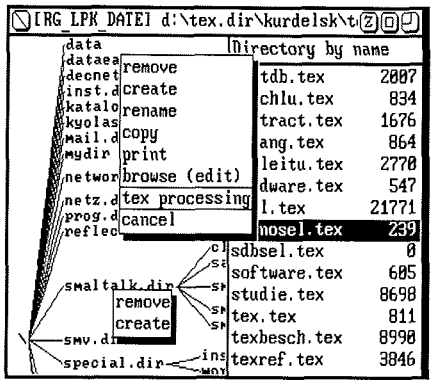


Abbildung 2.3: Das Menü des TEX–Zyklusses eines *TeXDiskBrowsers*. Bei jeder erreichbaren Funktion wird nach speziellen Parametern unter der Verwendung von Menüs, Dialogboxen und *Property–Sheets* gefragt.

Subjektive Reaktionen basieren ebenfalls auf der Art, in der Dinge interagieren. Der Unterschied besteht darin, daß wir uns in ihnen nicht mit Objekten der Außenwelt befassen, sondern mit Prozessen in unseren Gehirnen.

Marvin Minsky

Kapitel 3

Benutzerschnittstellen, User–Interface–Management–Systeme und Modellierungsmethoden

Jeder Anwender stellt aus seiner Arbeitssituation heraus unterschiedliche Anforderungen an ein Rechnersystem. Im AWI erzeugte insbesondere der Umgang mit Arbeitsplatzrechnern (IBM AT; Apple Macintosh II)[1] ein von Ablehnung über mangelnde Nutzung bis zur spontanen Akzeptanz reichendes Benutzerverhalten. Erfahrungen aus Diskussionen mit den Benutzern förderten neue Aspekte in der Benutzerbetreuung zutage, die in Ansätzen bereits in Angriff genommen wurden (standardisiertes Softwareangebot mit Benutzerbetreuung, standardisierte Hardware, Ausbildung der Benutzer, vorgegebene Symbole und Definitionen). Dabei darf man den wichtigsten Dialog, den zwischen Rechner und Benutzer, nicht vergessen. Dieses Kapitel ist eine Zusammenfassung der, in der Literatur bekannten, Ergebnisse der auf dem Gebiet der Benutzerunterstützung geleisteten Arbeit. Die dabei auftretenden Probleme werden verdeutlicht.

3.1 Die Modellansätze

Für das Verständnis der Handlungsabläufe des Benutzers sind mehrere Modelle entwickelt worden, die aus verschiedenen Blickwinkeln die Aktivitäten des Benutzers beschreiben. Im folgenden werden einige der wichtigsten Ansätze dargestellt, die auch die vorliegende Arbeit beeinflußt haben. Die Modelle sind in bezug auf ihren thematischen Zusammenhang nach folgenden Kriterien geordnet:

[1] seit November 1990 auch SUN und DEC Workstations

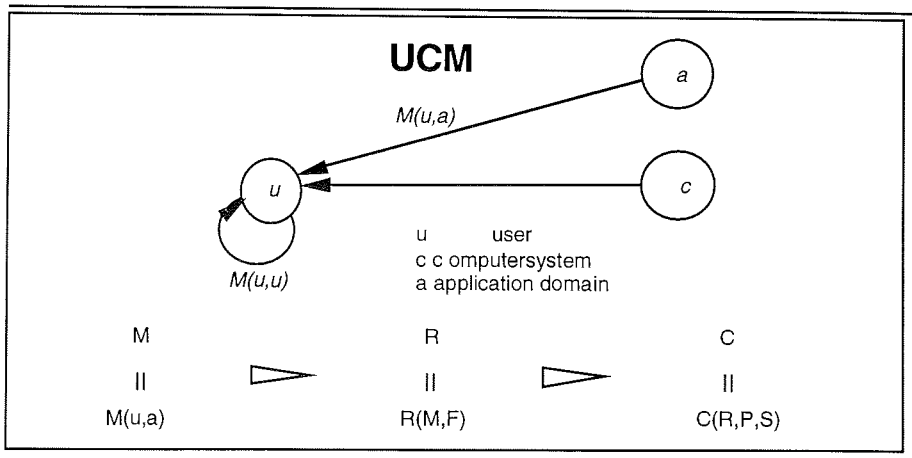


Abbildung 3.1: Ein *User–Conceptual–Model (UCM)* nach [MaA85]

M(u,a)	*UCM* des Benutzers u von der Entität a
R(M,F)	Repräsentation von M in dem formalen System F
C(R,P, S)	Encodierung der Repräsentation R mit der Programmiersprache P und der Semantik S

- konzeptuelle Modelle: UCM, Ebenenmodell, Simulationmodell, Modell von Balzert und Shneiderman
- quantitative Modelle: *Model–Human–Processor*, *KeyStroke–Level–Model*
- Regelbasierte Modelle: *GOMS*, *GOMS**

3.1.1 Das User–Conceptual–Model

Das *User–Conceptual–Model*, das die Konzepte und Beziehungen des Benutzers bezüglich einer Menge von Entitäten darstellt, wird von Mac an Airchinnigh formal spezifiziert [MaA85]. (Siehe Abbildung 3.1.)

> *A UCM is the set of all concepts and conceptual relations possessed by a (human) user with the respect to some set of entities. To each UCM there corresponds a universe of discourse that is the expression of those concepts and conceptual relations.*

Ein *User–Conceptual–Model* ist also die Vorstellung des Benutzers von der zu lösenden Aufgabe oder Einheit, dem Rechner und der Applikation. Gerade die konzeptuellen Modelle der Benutzer stellen das Bindeglied zwischen den an der Entwicklung einer Benutzerschnittstelle beteiligten Disziplinen dar (Informatik, Psychologie, Linguistik, Philosophie).

Das Modell ist geeignet, die Bildung von Vorstellungen und deren Abhängigkeiten zu erklären. Für die Realisierung eines Benutzungsmodells kann es jedoch nur Hintergrundinformationen liefern.

3.1.2 Ebenenmodell

Im Ebenenmodell von Foley und van Dam wird ein Ansatz in vier Beschreibungsebenen benutzt [FvD82]. Die Ebenen unterscheiden Konzepte, Semantik, Syntax und lexikalische Abhängigkeit.

- **Konzept:** Mentales oder konzeptionelles Modell des Benutzers vom System
- **Semantik:** Bedeutung der Benutzer–Eingabe bzw. Computer–Ausgabe
- **Syntax:** Regeln zur Formung von Instruktionen an den Computer
- **Lexikalische Abhängigkeit:** Geräteabhängige Spezifikation der Instruktionseingabe

Beispiele für die einzelnen Ebenen sind leicht zu finden. Der zeilenorientierte und der bildschirmorientierte Editor sind zwei konzeptionelle Modelle der Textbearbeitung. Die semantische und die syntaktische Ebene sind auch ohne Beispiel verständlich. Auf der lexikalischen Ebene kann z. B. zwischen der Befehlseingabe mittels Tastatur und Maus unterschieden werden.

3.1.3 Simulationsmodell

Das **G**eorg **W**ashington **U**ser **I**nterface **M**anagement **S**ystem *(GWUIMS)* benutzt einen objektorientierten Ansatz, um die Beziehungen zwischen Applikation, Interaktion, Repräsentation und Benutzer abzubilden [S+86]. In Abbildung 3.2 ist ein Überblick über das Objektnetz des *GWUIMS* dargestellt.

Die Kommunikation innerhalb der Benutzungsschnittstelle erfolgt auf drei Ebenen. Die Applikationsobjekte *(A_Objekt)* der semantischen Ebene stehen für die Applikation. Die Darstellung mittels Objekten ermöglicht zunächst eine Simulation der Applikation. Die zur Simulation verwendeten Applikationsobjekte werden später auf die reale Applikation hin erweitert.

In der lexikalischen Ebene wird bestimmt, wie Informationen dem Benutzer zugänglich gemacht werden *(R_Objekt)*. Dabei wird eine einfache syntaktische Überprüfung der Eingabe vorgenommen.

An der Schnittstelle zwischen Syntax und Semantik arbeiten die Interaktionsobjekte *(I_Objekt)*. In diesen Objekten wird allgemeines Wissen über die Applikation gehalten.

Dieses Modell ist eine konzeptionelle Darstellung der Kommunikation in einer komplexen Benutzungsoberfläche. Das Modell besitzt Eigenschaften des Ebenenmodells. Besonders hervorzuheben sind die Implementierungsnähe und die objektorientierte Struktur.

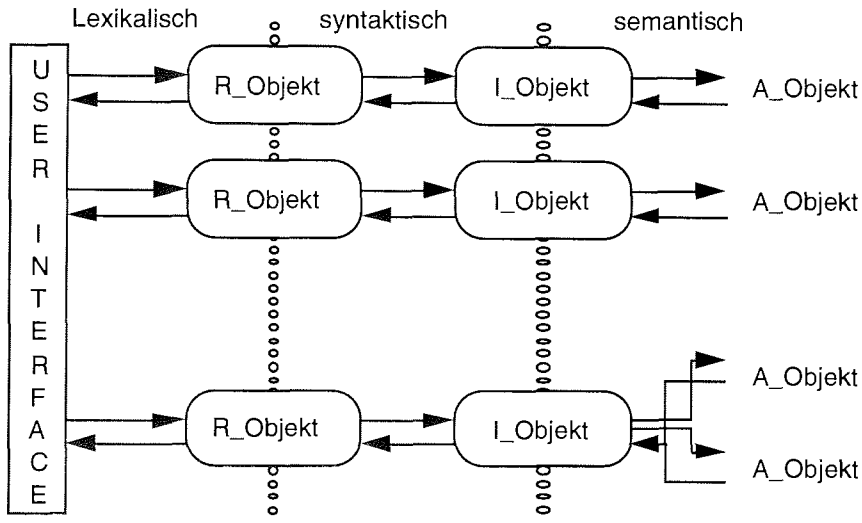


Abbildung 3.2: Überblick über das GWUIMS. (Erläuterungen siehe Text)

R_Objekt	Repräsentationsobjekt
I_Objekt	Interaktionsobjekt
A_Objekt	Applikationsobjekt

3.1.4 Modell nach Balzert

Ein System, das den Benutzer optimal unterstützt, muß Wissen über diesen Benutzer bereithalten. Dieses Benutzungsmodell beinhaltet Regeln und Fakten über den Benutzer, die individuell für jeden Benutzer erfaßt werden. Nach Balzert sollte ein Benutzungsmodell folgende Bereiche abdecken [Bal88, 345ff]:

Benutzerkonvention: Gestaltungsmöglichkeiten der Mensch–Computer–Schnittstelle und Anpassung der Anwendungssysteme auf die individuellen Bedürfnisse des Benutzers z. B. *login–files, profiles, autoexec–files*

Benutzerkompetenz: Erfahrungen, Wissensstand, Kenntnisse des Benutzers im Umgang mit Rechenanlagen, den geforderten Abläufen und den Grundlagen des zu erstellenden Systems. Beispielparameter sind

- Systemerfahrung des Benutzers,
- Zeitspanne seit der letzten Systembenutzung,
- Benutzungshäufigkeit von Systemfunktionen,
- Fehlerhäufigkeit als Indikator für ein Hilfe–System oder ein tutorielles System,

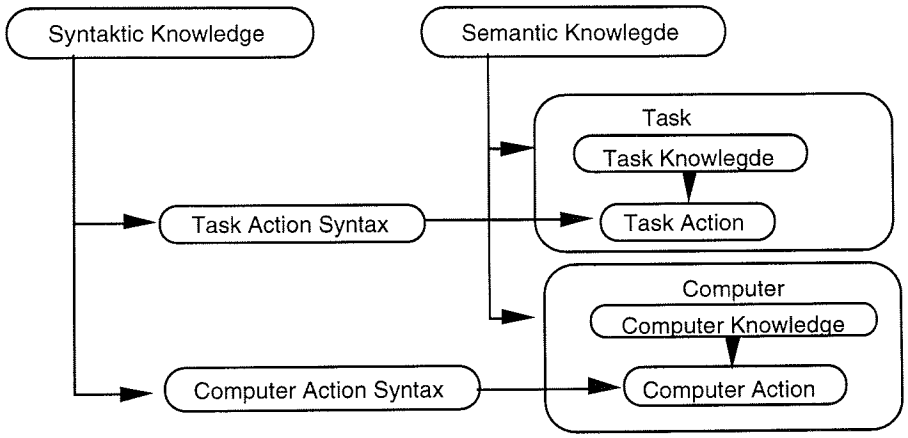


Abbildung 3.3: Das syntaktisch/semantische Modell nach Shneiderman

- Benutzungshäufigkeit von Funktionen, die in verschiedenen Anwendungen genutzt werden.

Benutzerintentionen: Erkennen der Absichten des Benutzers für mittel– bis langfristige Zeiträume bzw. Handlungszeiträume.

Die Anforderungen an ein System werden zunächst in einem Pflichtenheft beschrieben. Nach Balzert fallen die Spezifikationen des Pflichtenhefts in den Bereich der Konventionen einer Benutzer*gruppe*. Individuelle Konventionen sind mit anderen Methoden erfaßbar, z. B. *GOMS*–Ansatz und Netzdarstellungen zur Beschreibung von Benutzerhandlungsplänen.

3.1.5 Syntaktisches/semantisches Modell

Das syntaktische/semantische Modell unterscheidet zwischen der Syntax und der Semantik einer Aufgabe. Es unterstellt, daß der Benutzer syntaktisches Wissen über die geräteabhängigen Details und semantisches Wissen über die Konzepte der beteiligten Objekte besitzt (Abbildung 3.3).

Syntaktisches Wissen umfaßt die *low–level*–Details der Computerbedienung, wie „Wie bewege ich den Cursor?“, „Welchen Befehl benötige ich, um den Directory–Inhalt zu sehen ?“, „Nach welchen Regeln werden Abkürzungen erkannt?“. Syntaktisches Wissen ist vom jeweiligen System abhängig.

Semantisches Wissen kann in zwei Teilbereiche aufgeteilt werden, die *Computer–Konzepte (computer concepts)* und *Aufgaben–Konzepte (task concepts)*. In beiden Bereichen wird zwischen den Objekten und den Aktionen unterschieden. Die Konzepte sind hierarchisch aufgebaut, d. h., eine Aufgabe, die mit einem globalen Task

beginnt, wird bei der Analyse in mehrere Teilaufgaben unterteilt. Zu jedem dieser Teilkonzepte gehören wieder *task*– und *computer*–Konzepte.

Zur Unterscheidung und Erklärung der Begriffe sei ein Beispiel von Shneiderman nachempfunden [Shn85]. Um einen Geschäftsbrief zu schreiben, muß der Benutzer wissen, wie ein Geschäftsbrief *(task object)* geschrieben wird *(task action)*. Der Brief wird auf dem Rechner als File *(computer object)* gespeichert. Der Benutzer muß wissen, wie der Text abgespeichert werden kann *(computer action und syntactic knowledge)*. Zum Schreiben gehört außerdem noch das Wissen, wie ein Satz *(task object)* aufgebaut wird *(task action)*. Schließlich muß der Benutzer die Details des Buchstabierens eines Wortes *(task)* kennen, wie der Cursor auf dem Bildschirm bewegt wird *(computer concept)* und welche Taste für jeden Buchstaben gedrückt werden muß *(syntactic knowledge)*.

Das syntaktisch/semantische Modell wurde generiert, um das Programmieren zu beschreiben. Es wurde auch auf Datenbankmanipulationen und Systeme, die das Verfahren der *direkten Manipulation*[2] einsetzen, angewandt. Es ist besonders geeignet, um einen ersten Überblick über die Aufgaben des Benutzers zu gewinnen und im weiteren zur Analyse dieser Aufgaben beizutragen.

3.1.6 Der Model–Human–Processor

Mit dem Modell des *Model–Human–Processors* können die Zusammenhänge zwischen den kognitiven und motorischen Arbeitsphasen des Menschen verdeutlicht werden. Das Modell beschreibt die kognitiven Leistungen des Menschen mit Begriffen aus der Computertechnik. Diese Beschreibung hilft bei der Anwendung quantitativer Untersuchungsmethoden.

Der menschliche Funktionsapparat wird in die Bereiche Speicher *(memory)* und ausführende Einheiten *(processor)* unterteilt. Im Speicherbereich sind das Kurz– und Langzeitgedächtnis *(working memory, long term memory)* und die komplexen Speichereinheiten der visuellen *(visual image storage)* und auditiven *(auditory image storage)* Verarbeitung *(Sehen und Hören)* zu finden. Die Speicher besitzen Kapazitäten, Zugriffszeiten und Zugriffsverfahren. (Siehe Abbildung 3.4.)

Der Ausführungsteil enthält die drei zur Beschreibung des Funktionsapparates notwendigen Einheiten der sinnlichen Wahrnehmung *(perceptual processor)*, der Verarbeitung von Wissen und des Bildens von Folgerungen und Aktionen *(cognitive processor)* und der motorischen Steuerung *(motor processor)*. Diese Prozessoren greifen auf die Speicher zu und sind durch die Zugriffszeiten der Speicher und deren Kapazitätsbeschränkungen zusätzlichen Grenzen unterworfen[3].

Das Gesamtsystem kann durch eine Zykluszeit gekennzeichnet werden, die die Laufzeit vom empfangenen Reiz bis zur ausgeführten Aktion bestimmt. Kennzeichnend für die menschliche Funktionalität ist die Möglichkeit, die Prozessoren parallel

[2]Siehe hierzu [Shn83].

[3]Diese einfache Sicht wird insbesondere von Dreyfus und Weizenbaum kritisiert. Minsky hingegen unterstützt und erweitert dieses Konzept.

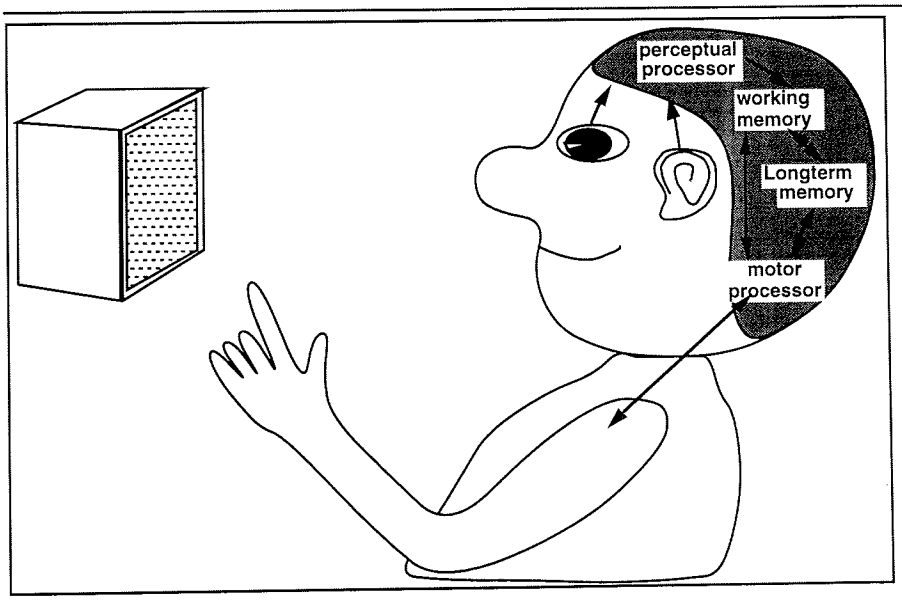


Abbildung 3.4: Der *Model–Human–Processor* (nach [CMN83])

und seriell zu betreiben. Das Modell wird ausführlich von Card, Moran und Newell beschrieben [CMN83].

Aus den Untersuchungen auf der Grundlage eines derartigen Modells lassen sich Richtlinien zur Gestaltung einer Benutzungsoberfläche entwickeln, wie z. B. der Einsatz von Maus oder Tablett anstatt oder gemeinsam mit der Tastatur, die syntaktischen Regeln bei Abkürzungen und Funktionstastenbelegung, die Wiederholrate des Anzeigeinstrumentes (*displayrate:* Wahrnehmungsteil), der Gestaltung der Tastatur (*Anordnung der Tasten:* motorischer Teil) oder der Berücksichtigung der Reaktionszeiten des Menschen durch angepaßte Antwortzeiten. Eine Aussage über das Lernverhalten des Benutzers ist ebenfalls möglich, da u. a. Befehlsauswahl– und Entscheidungszeiten meßbar sind.

3.1.7 Das KeyStroke–Level–Model

Das *KeyStroke–Level–Model* (Abbildung 3.5) ist eine Ergänzung des *Model–Human–Processors* und des *GOMS*–Modells. Es enthält Methoden, die auf der Ebene der Tastatureingabe die Aktionen des Benutzers erfassen und quantitativ bewerten, insbesondere Messung der Ausführungszeiten. Das Modell wurde zum Vergleich mehrerer Texteditoren angewandt, kann aber auf jede andere Aufgabe übertragen werden. Mit diesem Modell können Aussagen über die Ausführungszeiten einzelner Programme oder Programmteile bei Anwendung durch einen fehlerfrei arbeitenden *expert user*

Task T1: *Replace one 5–letter word with another (one line from previous task).*

Method for Task T1–Bravo:

Reach for Mouse	$\mathbf{H}_{[mouse]}$	*Home hands*
Point to word	$\mathbf{P}_{[word]}$	*Point with mouse*
Select word	$\mathbf{K}_{[yellow]}$	*Press key*
Home on keyboard	$\mathbf{H}_{[keyboard]}$	
Issue Replace command	$\mathbf{M}\ \mathbf{K}_{[R]}$	*Response by system*
Type new word	$5\ \mathbf{K}_{[word]}$	
Terminate type–in	$\mathbf{M}\ \mathbf{K}_{[ESC]}$	*Mentally prepare*
Wait for completion	$\mathbf{R}(0)$	

Abbildung 3.5: Auszug aus einer Analyse mit dem *KeyStroke–Level–Model* (nach [CMN83, S. 299])

gemacht werden.

Im Ausführungsteil der Aufgabe werden die Operatoren Tastendruck **K** *(keystroke)*, Zeigen **P** *(pointing)*, Zurückziehen der Hand zum Ursprungsort **H** *(homing)*, Zeichnen **D** *(drawing)* im physikalisch–motorischen Bereich, der mentale Operator **M** *(mental)* und die Antwortzeit des Systems **R** *(responsetime)* bestimmt. Nur der mentale Operator **M** muß mit heuristischen Methoden bestimmt werden. Die anderen Operatoren werden während des Experiments quantitativ erfaßt.

Bei diesem Modell werden die Unterschiede der einzelnen untersuchten Systeme in Hinblick auf den Kenntnisstand des Benutzers im Umgang mit Computern nicht beachtet. Ein Experte kann, z. B. mit dem Texteditor *TECO*, in vielen Fällen bessere Resultate bei der Textbearbeitung erzielen als ein normaler Benutzer (*intermittend user*) mit einem graphischen Textverarbeitungssystem. Die subjektive Einstellung des Benutzers zum System kann mit diesem Modell nicht erfaßt werden.

Für eine Analyse des Benutzerverhaltens ist dieses Modell nur beschränkt nutzbar, da im wesentlichen nur physikalische Parameter erfaßt werden

3.1.8 GOMS

Mit dem *GOMS*–Modell ist die Beschreibung einer Aufgabe, die der Benutzer zu absolvieren hat, und deren Analyse durchführbar. Zur Beschreibung einer Aufgabe sind Ziele *(**g**oals)* zu definieren, die nach feststehenden Methoden *(**m**ethods)* abgearbeitet werden. Die Auswahl der Methoden im einzelnen erfolgt durch Selektoren *(**s**electors)*. Mit Operatoren *(**o**perators)* werden explizite Anweisungen dargestellt.

In Abbildung 3.6 ist ein Beispiel einer Aufgabenanalyse gezeigt. Es wird das Verfahren gezeigt, mit dem der Benutzer ein Wort in einem Manuscript ediert.

```
GOAL: EDIT-MANUSCRIPT repeat until no more unit tasks
. GOAL: EDIT-UNIT-TAST
. . GOAL: ACQUIRE-UNIT-TASK
. . . GET-NEXT-PAGE   if at end of manuscript page
. . . GET-NEXT-TASK
. . GOAL: EXECUTE-UNIT-TASK
. . . GOAL: LOCATE-LINE
. . . . [select: USE-QS-METHOD
                 USE-M-COMMAND]
. . . . VERIFY-EDIT.
```

Abbildung 3.6: Beispiel einer Aufgabenbeschreibung mit *GOMS* (nach [CMN83])

Die Struktur des Modells erleichtert durch ihren Aufbau die Analyse der Arbeit des Benutzers. Das Modell ist hierarchisch gegliedert. Ziele *(Goals)* werden mit Methoden realisiert, die ihrerseits wieder auf Teilziele zurückgreifen.

Das *GOMS*–Modell eignet sich gut, um einen ersten detaillierten Überblick über die Verfahren und Handlungsweisen des Benutzers zu erhalten. Sein besonderer Wert liegt in der direkten Transformation in ein Regelsystem[4]. Im gemeinsamen Einsatz mit dem *KeyStoke–Level Model* haben Card, Moran und Newell die Bedeutung bei der zeitlichen Analyse des Systemverhaltens gezeigt [CMN83].

3.1.9 Das GOMS*–Modell

Das von Card, Moran und Newell entwickelte *GOMS*–Modell basiert auf der Beschreibung des fehlerfrei arbeitenden Experten [CMN83]. Mit diesem Modell kann aber nicht der Anfänger oder der Lernende beschrieben werden [Are89a]. Ebenso fehlen Informationen, wie Kontrollstrukturen notiert werden und welche Informationen unter einem GOAL oder einer Operation erlaubt sind.

Arend entwickelte eine Erweiterung, um den erwähnten Lücken zu begegnen [Are89a]. Das neue *GOMS**–Modell wird um Kontrollflußstrukturen erweitert. Mentale Ziele und physikalische Operatoren werden über die zugrundeliegenden kognitiven, wahrnehmenden und motorischen Prozesse unterschieden.

Die implizite prozedurale Beschreibung des *GOMS*–Modells wird im *GOMS**–Modell mit syntaktischen Hilfsmitteln aus prozeduralen Programmiersprachen als Sprachelement festgelegt. Prozeduren fassen eine Sequenz von *GOMS**–Instruktionen zu einer Einheit zusammen. Andere Kontrollstrukturen erheben die kommentarähnlichen Bemerkungen zu neuen syntaktischen Schlüsselwörtern (z. B. *re-*

[4]Kritiken zu Regelsystemen siehe wiederum [Dre72].

Tabelle 3.1: Prozeduren und prozedurales Wissen in *GOMS**: Darstellung einer Prozedur mit dem zur Ausführung der *UNIT*s notwendigen Wissen in Form eines *SCHEMAS* (aus [Are89a])

procedure insert_course	*SCHEMA* insert_course
.*UNIT:* **CHOOSE**(room)	.*UNIT:* **TYPE**(menu_selection)
.*UNIT:* **CHOOSE**(course)	.*UNIT:* **TYPE**(course_selection)
.*UNIT:* **CHOOSE**(n_hours,free_hours)	.*UNIT:* **TYPE**(mark_fields)
endprocedure	*ENDSCHEMA*

peat_until — *endrepeat*). Parallel zu den Prozedurdefinitionen wird eine Wissensrepräsentation eingeführt, die der Prozedur notwendige *Wie–führe–ich–es–aus*–Kenntnisse über die Aktionen bereitstellt. Das Wissen ist in einem der *procedure* entsprechenden Schema dargestellt. Ein Beispiel wird in der Tabelle 3.1 gezeigt.

3.2 Beschreibungsmittel

Die vorgestellten Modellansätze erfassen Informationen über den Benutzer in strukturierter, überschaubarer und implementierbarer Form. Das Wissen über einen Benutzer bzw. die Handlungsarten des Benutzers werden am anschaulichsten durch eine Graphik dargestellt. Diese Repräsentation ist gleichzeitig als formale Beschreibung einsetzbar, wenn die graphischen Elemente formalen Anforderungen genügen. Die Anforderung der Implementierbarkeit ist z. B. beim *GOMS**–Modell durch den prozeduralen Charakter der verwendeten Operatoren gegeben [Are89a][5]. Wichtige Vertreter dieser formalen graphischen Beschreibungen sind Transitionsnetze. Ein besonderer Vertreter ist die Darstellung als Petri–Netz. Im folgenden werden drei für den Einsatz im Bereich Benutzungsmodellierung verwendete Darstellungen erläutert.

3.2.1 Platz–Transitionsnetze

Transitionsnetze sind als Beschreibungen von Systemverhalten durch Zustände (Stellen) und Transitionen bekannt. Durch die Verbindungen der Zustände mit Transitionen und umgekehrt und das Einhalten formaler Aspekte bei der Darstellung des Netzes lassen sich Regeln aus dem Netz ablesen, nach denen ein Zustand erreicht wird.

Ein Transitionsnetz (speziell Petri–Netz) besteht aus Ereignissen und Bedingungen (allgemeiner Transitionen und Stellen). Transitionen werden *gezündet*, wenn *alle*

[5]Arend zeigte, wie mit wenig Aufwand die Spezifikationen mittels einer *GOMS**–Analyse direkt in eine implementierbare Form gebracht werden können[Are89a].

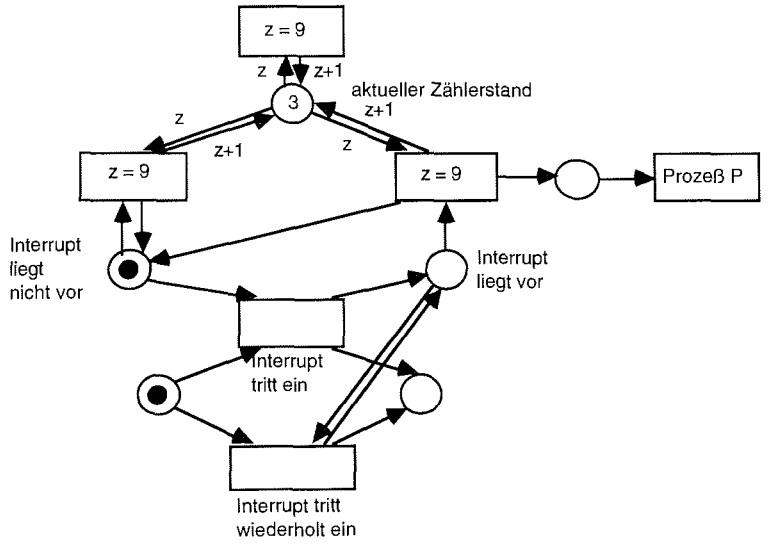


Abbildung 3.7: Beispiel eines Transitionsnetzes: *Eine einfache Interrupt–Steuerung* (aus [Rei85, S. 61])

zugehörigen Bedingungen erfüllt sind. Die Folgezustände werden daraufhin gesetzt. Ein Beispiel ist in Abbildung 3.7 gezeigt.

Formal ist ein Platz–Transitionsnetz ein Tupel $(\mathcal{P}, \mathcal{T}, \mathcal{R})$. Dabei sind $\mathcal{P}$ und $\mathcal{T}$ die Plätze bzw. die Transitionen. $\mathcal{R}$ ist eine Relation, die jeweils einen Platz mit einer Transition verbindet und umgekehrt. Ordnet man den Verbindungen zwischen Plätzen und Knoten natürliche Zahlen, die Vielfachheit V zu, und setzt im Netz durch Marken, die belegte Plätze repräsentieren, eine Markierung m_0, so erhält man ein Petri–Netz $(\mathcal{P}, \mathcal{T}, \mathcal{R}, V, m_0)$. Wenn die Vielfachheit $V = 1$ für alle Bögen aus $\mathcal{R}$ ist, wird das Netz als gewöhnliches Petri–Netz bezeichnet.

Probleme treten bei der Darstellung des gesamten Netzes auf, da dieses sehr umfangreich werden kann. Netze können daher verfeinert und zusammengefaßt werden. Dabei ist darauf zu achten, daß die Reduktion der Netze die Eigenschaften des ursprünglichen Netzes invariant läßt.

Für die Beschreibung realer Vorgänge wird die Beschreibung mit Petri–Netzen aufwendig und unübersichtlich. Daher werden Modifikationen an Plätzen, Transitionen und Bögen durchgeführt, die das Problem leichter überschaubar machen. Die Modifikationen bergen aber die Gefahr, daß u. U. die theoretisch fundierte Grundlage verlorengeht. Die entstandenen Netztypen werden durch Festlegung der Art der Marken (z. B.: unterscheidbar), der Art der Beschriftung der Netzelemente (z. B.: Schaltausdrücke für Transitionen), die Art der Bewertung von Netzelementen (z. B.:

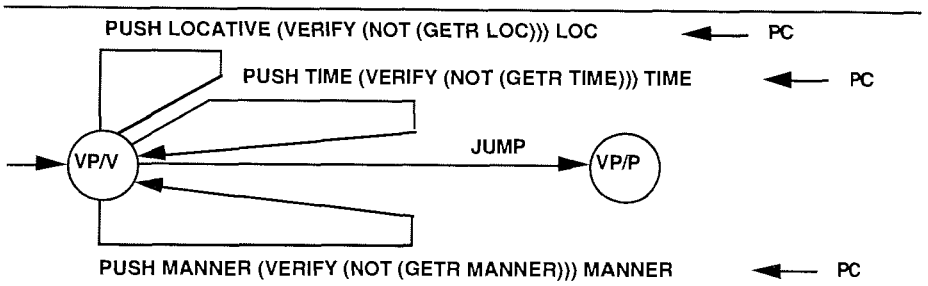


Abbildung 3.8: Beispiel eines ATN-Netzes: [Sha87]. Die Bögen erlauben Adverbialphrasen des Ortes (**LOC**), der Zeit (**TIME**) und der Art und Weise (**MANNER**) nach einer Verbalphrase (**VP**) in beliebiger Reihenfolge. (**V** Verb, **PC** erkannter Bestandteil *parsed constituent*).

Kapazitäten für Plätze, Prioritäten für Transitionen), die Schaltregel und die Schaltstrategie unterschieden.

3.2.2 ATN/GTN–Netze

Die Übertragung natürlicher Sprache in eine Form, die vom Computer verarbeitet werden kann, wird mit verschiedenen Techniken durchgeführt. Eine wichtige Technik dabei ist der Einsatz von Grammatiken. Grammatiken können graphisch in Netzform dargestellt werden. (Vergleiche Syntax–Diagramme [Wir75].) Eine wichtige Form der graphischen Repräsentation einer Grammatik ist das *ATN (augmented transition network)*. Eine Eingabe des Benutzers, z. B. einzelne Zeichenketten, Zeichenkettenfolgen, wird an Hand dieser Grammatik analysiert und in eine computergerechte Form überführt.

Das Netzwerk besteht aus Zuständen *(states)* und Transitionen *(arc)*. Jeder Bogen ist durch den Namen des Konstituenten gekennzeichnet, der die Transition ermöglicht. Eine Transition kann wiederum auf ein Netzwerk bezogen sein, das die Transition näher beschreibt. In diesem Fall liegt ein *RTN (recursiv transition network)* vor. Besitzt ein Netz Register, die Zwischenzustände speichern können, d. h. einzelne Abschnitte der Analyse, dann spricht man von einem *ATN*. Formal kann ein *ATN* als Automat betrachtet werden, dessen momentane Belegung die aktuelle Position der Eingabe, den Namen des Zustands, die Menge der Registerinhalte und den Stack kennzeichnet (nach [Sha87], Seiten 323–333). Ein Beispiel befindet sich in Abbildung 3.8.

Eine erweiterte Form der *AT*–Netze wurde von Kieras und Polson zur Betrachtung der kognitiven Benutzerkomplexität *(cognitive user complexity)* eingeführt: GT-Netze *(generalized transition networks GTN)* [KP85]. Diese Netze unterscheiden sich in der graphischen Darstellung von einfachen Transitionsnetzen. Die Zustände

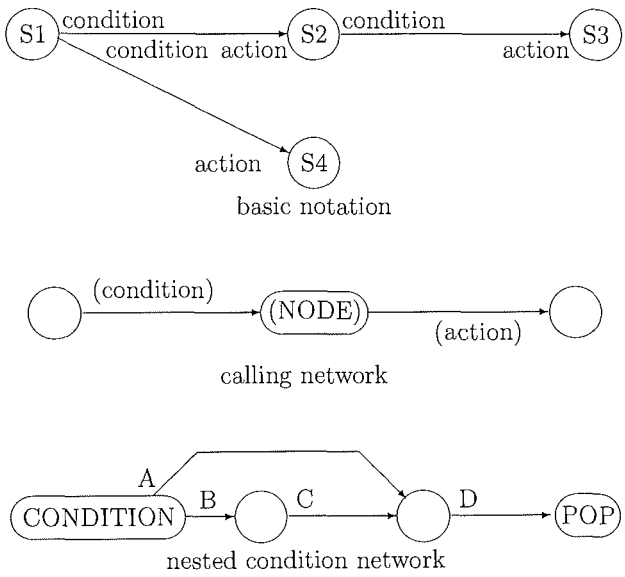


Abbildung 3.9: Beispiel eines GTN-Netzes: [KP85]

werden als kreisförmiges Symbol abgebildet. Die Transitionen werden textuell an die Verbindungspfeile zwischen den Zuständen geschrieben. Bedingungen, die für eine Transition notwendig sind, werden ebenfalls an den Verbindungspfeil geschrieben.

Die Darstellung in Form von *GTN* erlaubt eine einfache Repräsentation hierarchisch geordneter Abläufe in einem System. In Abbildung 3.9 ist ein Beispiel gezeigt.

Die *GTN* erlauben die Schachtelung eines anderen Netzes in der Art eines Unterprogramms in einer Programmiersprache. Rekursionen sind ebenfalls möglich.

Der Aufruf eines Netzes kann über Bedingungen *(condition nesting)* und/oder Aktionen *(action nesting)* bzw. durch den Aufruf eines Zustands *(state nesting)* erfolgen.

3.2.3 RFA–Netze

Während *GTN*–Netze für die psychologisch orientierte Analyse des Benutzerverhaltens bestimmt sind, werden mit *RFA*–Netzen Aspekte der Systemanalyse und des Software–Engineerings unterstützt. Bedingt durch die konsequente Betrachtung des Benutzers in seiner Rolle im Gesamtsystem können *RFA*–Netze zur Beschreibung des Benutzerverhaltens herangezogen werden.

Bei der Lösung eines Organisationsproblems wird eine Bedarfsanalyse durchgeführt, die Aussagen über die auszuführenden Tätigkeiten gibt. Aus dieser Bedarf-

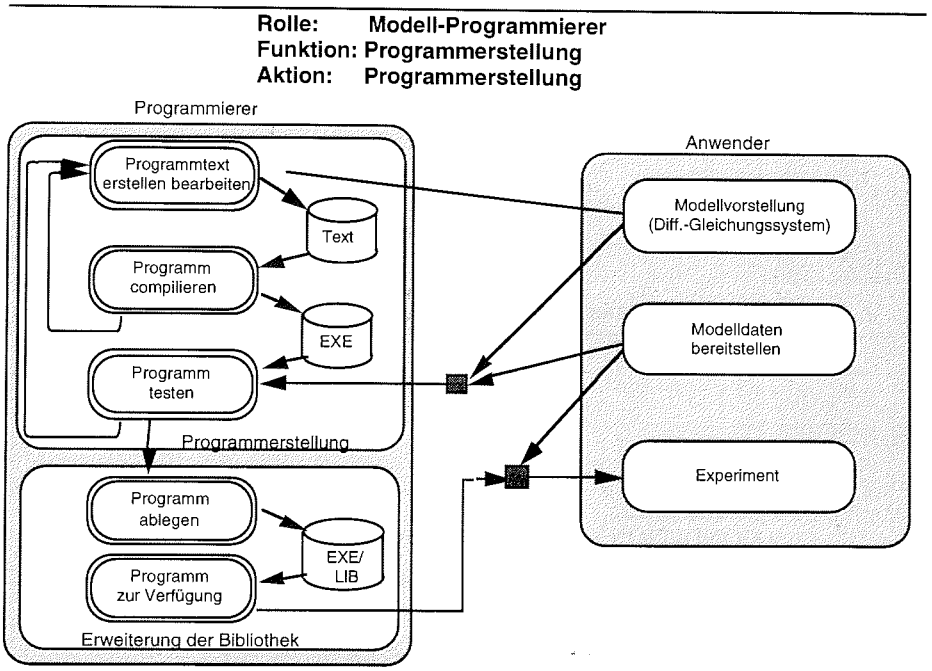


Abbildung 3.10: Der Benutzer als Programmierer: Beispiel eines *RFA*–Netzes, wie es in der vorliegenden Arbeit angewandt wurde

sanalyse wird ein Entwurf erarbeitet und dieser dokumentiert. Der Benutzer wird bei der Entwurfsgestaltung nicht mit einbezogen:

> *Die Entwickler gehen mit einem aus dem Umgang mit Computern entstandenen und durch vielfältige Abstraktionen gekennzeichneten Grundverständnis an die Aufgabe heran, eine Organisation zu verändern. Sie sind auf Grund ihrer speziellen Perspektive nicht in der Lage, praktisch relevante Phänomene und Bedürfnisse bei der Modellbildung (Systemanalyse, Entwurf, Dokumentation) zu erfassen.* [Obe88]

Um den Benutzer frühzeitig in die Entwicklung mit einzubeziehen und eine benutzerorientierte Beschreibungssprache für das System Mensch–Maschine zu entwickeln, führt Oberquelle RFA–Netze[6] ein [Obe88]:

Rolle: Eine Rolle ist die zielgerichtete Aufgabe einer Person mit dem zur Erfüllung notwendigen Wissen und allen Hilfsmitteln.

[6] **R**olle/**F**unktion/**A**ktion

Funktion: Eine Funktion ist eine Zusammenfassung der zur Erfüllung der Rolle notwendigen Tätigkeiten zusammen mit dem Träger (eine Person) und der benötigten Mittel.

Aktion: Eine Aktion ist eine Zusammenfassung der für eine Tätigkeit notwendigen Handlungen und des dynamischen Verhaltens.

RFA–Netze: Ein RFA–Netz ist eine Menge von beschrifteten Netzen, die statische Strukturen von Rollen und Funktionen und die Dynamik auf der Aktionsebene beschreiben.

RFA–Netze sind eine Weiterentwicklung der Transitionsnetze. Die Neuerung der *RFA*–Netze liegt in der Bereitstellung einer Beschreibungssprache aus der Sicht des Benutzers, der als Rollenträger eingeführt wird. Die abstrakten Rollenträger können in ihrer Gesamtheit und mit allen Abhängigkeiten dargestellt werden.

In (Abbildung 3.10) ist ein Beispiel eines *RFA*–Netzes gezeigt. Sie zeigt den Benutzer in seiner Rolle als Programmierer aus der Analyse des Autors.

3.2.4 Produktions–Systeme *(production systems)*

In einem Produktionssystem wird aus Prämissen eine Konklusion gebildet. Eine beispielhafte Regel ist: „Wenn ich den Text in der dritten Zeile verändern will, dann muß ich den Cursor zunächst in die dritte Zeile bewegen". Produktions–Systeme bestehen aus Regeln mit Bedingungen und Aktionen nach dem Schema

WENN *(Bedingung)* **DANN** *(Aktion).*

Dieser Ansatz entspricht dem Verfahren, Handlungen durch eine Sammlung von Regeln in einem Rechner nachzubilden. Die Einführung von Expertensystemen ermöglicht die Simulation des Benutzerverhaltens durch Implementation der Regeln in einer Wissensdatenbank. Das System kann auf dieses Wissen zurückgreifen und Aktionen auswählen, verwerfen oder Hilfe anbieten.

Newell verwendete diesen Ansatz als Werkzeug, um formale Modelle psychologischer Prozesse zu bilden [New72]. Diese Modelle dienten der Entwicklung problemlösender Prozesse, der Entwicklung des Lernens und anderer kognitiver Prozesse *(GPS*[7]*)*. Das Wissen der Benutzer wird von Kieras und Polson mittels des *GOMS*–Modells erfaßt und dann in die Produktionsdarstellung überführt [KP85]. In dieser letzten Form kann nun die Benutzeraktion verfolgt und unterstützt werden. Kieras und Polson benutzten ähnlich wie Shneiderman [Shn85] *task*– und *device*–abhängige Unterscheidungen des Benutzerwissens in Form von *task representation* und *device representation.*

Die Repräsentation des Wissens des Benutzers in Produktionsregeln läßt sich mit einer Sprache wie PROLOG oder LISP oder mit einem auf Regeln basierenden

[7] *general problem solver*

Expertensystem realisieren. Das System von Kieras und Polson ist auch ein Beispiel einer Implementierung.

Dieses Modell kann zur Bestimmung der Einlernzeiten und Ausführungszeiten eines interaktiven Programms eingesetzt werden. Die Anzahl und Komplexität der Regeln ist ein Maß hierfür.

3.2.5 Weitere Beschreibungsmittel

In der Literatur finden sich weitere Methoden, die auf Netzen basieren. Die meisten dieser Darstellungen werden bei der Anforderungsermittlung für ein System eingesetzt.

Aufgabennetze: Keil–Slawik nutzt *Aufgabennetze* als gemeinsame Sprache innerhalb einer Projektgruppe [KS89]. Aufgabennetze sind weiterentwickelte Petri–Netze. Nach Keil–Slawik sind sie verwandt mit *RFA*–Netzen.

Interaktionsmanagementnetze: Stary geht von globalen aufgabenorientierten Aktivitäten des Benutzers aus, die problem– als auch interaktionsspezifisch analysiert werden [Sta89]. Es erfolgt eine Transformation der Aktionen zu Systemfunktionen und Interaktionsprozeduren. Das benötigte Wissen wird in Form von konzeptuellen Einheiten und deren Abhängigkeiten repräsentiert. Das Interaktionsmanagementnetz wird algebraisch formuliert.

Beide vorgestellte Verfahren dienen der Spezifikation der Aktionen und Rollen des Benutzers bei der Aufgabenanalyse. Es ist nicht vorgesehen, diese Netze zu implementieren und während des Betriebes anzupassen. Die Informationen, die bei der Analyse gewonnen werden, können als Grundlage für ein Benutzungsmodell dienen.

3.3 Diskussion der Einsatzmöglichkeiten

Die gezeigten Modelle wurden von ihren Autoren entwickelt und/oder eingesetzt, um

- Daten über die Leistung eines Systems zu erlangen [CMN83],
- das Benutzerverhalten zu simulieren [KP85],
- Benutzerschnittstellen zu testen [S$^+$86].

Jedes dieser Modelle bietet Ansätze, das Benutzerverhalten zu analysieren und formal darzustellen. Keines dieser Modelle ist bisher in einem realen System angewandt worden, insbesondere wird Wissen in Wissensbasen erst in wenigen Systemen implementiert. Mit der Benutzerschnittstelle *X–Aid* der GMD Birlinghoven ist ein erstes langfristiges Projekt entstanden, ein System mit *intelligenten* Eigenschaften

zu erstellen. Meist sind Basisfunktionen implementiert wie Menüs, Fenstertechnik, graphische Oberfläche etc.

Lernfähige Systeme sind bisher immer noch Gegenstand der Forschung. Die Lernfähigkeit des Benutzers bezüglich des Dialogs mit dem Rechner wird bei Bösser behandelt [Bö84]. Die Systemunterstützung wird der Lernphase des Benutzers angepaßt. Für den Verlauf des Lernfortschritts ist der situative Kontext maßgeblich. Ein Verfahren, das auf einer *Task–Action–Grammar* basiert, erkennt aus den Benutzeraktionen Handlungspläne, lernt diese und kann gegebenenfalls Makros generieren [Hop88a, Hop89]. Ein beispielhafter Einsatzfall ist ein *UNIX–Coach* zur Unterstützung des Benutzers bei der Anwendung von UNIX [HP89]. Der Dialog zwischen Mensch und Maschine mittels natürlichsprachlicher Ein–/Ausgabe und die Probleme bei der Analyse der Semantik werden bei Kass und Finin diskutiert [KF88b].

Die Auswertung der auf dem Markt verfügbaren Systeme zeigt, daß mittlerweile mehrere Hersteller mit den Konzepten der ergonomischen Benutzerführung vertraut sind und versuchen, den Benutzer, so weit es möglich ist, vom Betriebssystem und den Eigenheiten des Rechners fernzuhalten. Bei Herstellern, die auf das Betriebssystem UNIX gesetzt haben, ist dies durch die Berücksichtigung des Betriebssystems naturgemäß schwerer (z. B. SUN, Apollo) als bei einem Betriebssystem, das die Eigenschaften einer graphischen Benutzeroberfläche bereits integriert hat (z. B. Apple).

Die Kommerzialisierung einer Methode oder eines Konzepts ist mit langen Entwicklungszeiten und derzeit noch hohen Kosten verbunden. Daher ist es durchaus verständlich, daß noch kein Hersteller derartige Benutzungsoberflächen anbietet.

Nach dem Erfolg der graphischen Oberflächen ist der Einsatz dieser Hilfsmittel auch auf Rechnern zu beobachten, deren Hersteller bisher auf diese Hilfsmittel verzichteten. In diesem Zusammenhang seien nur die Oberfläche *WINDOWS/386* (respektive IBM Presentation Manager), das neue DOS 5.0 und *WINDOWS 3.1* genannt.

Auch einzelne Programme wie *Excel* (Tabellenkalkulation), spezielle Programme für Wirtschaftsunternehmen wie Ausschreibungsbearbeitung werden mit diesen und weiteren Techniken ausgestattet. Die Ansätze beruhen auf speziellen Erfahrungen mit den Anwendern und sind meist reine technische Lösungen. Die besonderen Erfahrungen im wissenschaftlichen Bereich werden in den folgenden Kapiteln analysiert.

Human engineering, which was seen as the point put on at the end of a project, is now understood to be the steal frame on which the structure is built.

Ben Shneiderman

Kapitel 4

Analyseverfahren, Benutzungsmodell, Dialog

Im Vordergrund der weiteren Betrachtungen steht ein Verfahren zur Analyse des wissenschaftlichen Arbeitsplatzes am Alfred–Wegener–Institut. Die im vorhergehenden Kapitel vorgestellten Modelle werden auf ihre Anwendbarkeit überprüft. Dabei wird besonders die im Vorfeld stehende Analyse der Arbeitsvorgänge und die Bedarfsanalyse diskutiert. Ein neuer konzeptioneller Ansatz, mit dem das Interaktionsverhalten des Benutzers und seine Aufgaben erfaßt werden können, wird vorgestellt.

Im Gegensatz zu Viereck wird hier keine Analyse des Designprozesses vorgenommen [Vie93] sondern ein Verfahren entwickelt, wie Informationen über den Benutzer sinnvoll erfaßt und geeignet dargestellt werden können. Ein Hilfsmittel wie EXPOSE [FGV93] ist ein statisches Werkzeug, mit dessen Hilfe eine feste, unveränderbare Benutzungsoberfläche entwickelt werden kann. Als Expertensystem im Gesamtsystem integriert könnte EXPOSE gute Dienste leisten.

Ein weiterer Überblick über Analyse und Entwurfsmethoden ist bei Ziegler zu finden [Zie88]. Außerdem werden die Grundlagen für ein Benutzermodell analysiert, und es wird die Dialoghistorie in Rahmen des *UnDo/ReDo* eingeführt.

4.1 Beschreibungskonzept des wissenschaftlichen Arbeitsplatzes am AWI

Bei der Entwicklung eines neuen Systems wird zuerst ein Anforderungskatalog der Benutzer aufgenommen [WB84]. Dieser Anforderungskatalog stellt die Basisforderungen an das System zusammen (Pflichtenheft). Ein Pflichtenheft beschreibt einen Ist– und einen Sollzustand. Diese Zustände berücksichtigen insbesondere keine dynamischen Komponenten, die das *Aktionsverhalten* des Benutzers beschreiben

oder unterstützen. Systeme und Modelle, die nur eingeschränkt auf diese besonderen Wünsche eingehen, werden durch Vernachlässigung wichtiger Informationen bei einer Simulation des Benutzerverhaltens sicher versagen.

In Kapitel 2 wurden bereits die Basisanforderungen der Benutzer des AWI an die Rechenanlage dargestellt. Die Anforderungen konnten nur durch den Einsatz unterschiedlicher Software und Hardware von verschiedenen Herstellern befriedigt werden. Eine Basisfunktionalität wird gewährleistet. Diese Basisfunktionen umfassen z. B. Edieren, Compilieren, Textausgabe, Graphikausgabe auf einem Plotter. Eine wirkungsvolle Unterstützung des Benutzers kann jedoch erst nach der Analyse seiner Arbeitsschritte und Arbeitsmethoden erzielt werden.

Ein umfassendes System, das mehrere der in Kapitel 2.3 beschriebenen Aufgaben bereitstellt, muß mit benutzerorientierten Analysemethoden betrachtet werden. In der Literatur sind unterschiedliche Ansätze hierzu bekannt.

Benda betrachtet das Sachproblem, das Interaktionsproblem und die Rolle des Benutzers [vB84]. Hier wird festgestellt, wer als Nutzer in Frage kommt, wie das System genutzt wird und wie sich der Benutzer das System nutzbar machen kann. Ein auf diesem Konzept basierendes Beispiel ist bei Dirlich et al. zu finden [DvBF+84].

Ackermann vergleicht verschiedene Spezifikationsansätze für den Entwurf einer Dialogschnittstelle [Ack88]. Die Ansätze unterscheiden Richtlinien, Normen und Leitfäden. Der Vergleich zeigt, daß Richtlinien wertvoll sind und ein iterativer Entwurf mit Benutzerbeteiligung angestrebt werden sollte.

Carrol verlangt zum Softwaredesign Spezifikationen der Funktionen *(functional specifications)*, der Nutzbarkeit *(usability specifications)* und Spezifikationen über die Fähigkeiten der Benutzer *(subskill specifications)* [CR85]. Da sich die Fähigkeiten des Benutzers während der Nutzungsdauer des Systems verändern, kann hier die Notwendigkeit zur Anpassung der *subskill specifications* abgeleitet werden.

Für die weitere Analyse ist ein Pflichtenheft, das die Basisfunktionalität des Systems beschreibt, sinnvoll. Mit einem Anforderungskatalog in Form eines Pflichtenheftes wird ein Zustand beschrieben, der zum Zeitpunkt der Analyse gültig ist. Dies gilt insbesondere für die Beschreibung der vom Benutzer geforderten möglichen Aktionen. Ein iteratives Design, bei dem der Benutzer bereits frühzeitig in die Entwicklung mit einbezogen wird, ergibt eine bessere Kontrolle im Entwicklungszyklus.

Wenn eine Individualisierung stattgefunden hat, so besteht ohne die Benutzerintegration aufgrund fehlender Flexibilität meist keine Möglichkeit, das Systemverhalten schnell anzupassen. Daher sind weitere Untersuchungen notwendig, um das Benutzerverhalten zu analysieren und zu erfassen. Die Methodik orientiert sich an den in Kapitel 3 dargestellten Modellen.

4.1.1 Das Beschreibungskonzept

Die grundsätzliche Überlegung bei diesem neuen Beschreibungskonzept ist, den Benutzer weitgehend ohne großflächige Interviewaktion zu befragen und gleichzeitig eine frühe Strukturierung der Daten zu ermöglichen. Interviews mußten im vorlie-

genden Fall vermieden werden, um den wissenschaftlichen Betrieb im AWI nicht durch große Aktionen zu unterbrechen.

Auswahl der Beschreibungmethoden

Die Beschäftigung mit den verschiedenen Modellansätzen ergab, daß jedes Modell Vorteile bietet, jedoch unhandlich wird, wenn mit einem gewählten Modell das gesamte System beschrieben werden soll. Daher verzichten wir auf die Auswahl eines einzigen Modells und werden statt dessen die Vorteile der verschiedenen Modelle ausnutzen und die gewonnenen Daten an geeigneten Schnittstellen in ein anderes Modell überführen.

Im Hinblick auf die Implementierung und Erkennung von Makros aus den Aktionen des Benutzers fiel die Entscheidung zunächst auf das *GOMS**–Modell[1]. Der *GOMS**–Ansatz eignet sich gut als strukturierte Darstellung von Aktionsfolgen. Er eignet sich besonders, um aus vom Benutzer erstellten Scriptdateien einen regelbasierten Handlungsplan zu erstellen. Die syntaktischen Hilfsmittel des Scriptes und des *GOMS***–Ansatzes sind sich sehr ähnlich. (Vergleiche Kapitel 5.3.) Im *GOMS***–Ansatz wird gleichzeitig das Wissen über die Aktionen, z. B. Selektion von Methoden, betrachtet.

Teilweise werden interessante Aspekte anderer Modelle weiter beachtet, ohne explizit als Modell verwendet zu werden, so die Einteilung der Information in *task representation* und *device representation* [KP85] und syntaktische, semantische sowie lexikalische Strukturen [Shn85, FvD82].

Transitions–/*RFA*–Netze werden zur Analyse herangezogen, um die Zusammenhänge der Aktionen und Objekte darzustellen und die verschiedenen Rollen des Benutzers zu erfassen. (Siehe z. B. Abbildungen 4.1 und 3.10.) Eine Zusammenfassung der Abläufe zu einer umfassenden Einheit wird dadurch möglich. Dieses Netzwerk enthält mögliche Zustände des Systems und der modellierten Benutzer. Eine Aktionsfolge des Benutzers ist dann durch einen Pfad innerhalb dieses Netzwerkes, der durch eine *GOMS***–Beschreibung realisiert wird, repräsentierbar. In Abbildung 4.2 sind die Analysen mit Transitionnetzen und dem *GOMS***–Modell in einem Beispiel gegenübergestellt.

Bei der Arbeit mit dem Rechner wird das System Entscheidungen protokollieren müssen, um aus statistischen Analysen dieser Beobachtungen dem Benutzer Veränderungen des Systems mitzuteilen, soweit sie die Interaktion mit dem Benutzer betreffen, bzw. diese selbständig anzupassen, wenn keine weitgreifenden Änderungen der Bedienung erfolgen, z. B. Prioritäten für zeitaufwendige Abfragen.

[1] Das *GOMS**–Modell wurde letztendlich modifiziert, um den Ansprüchen der Weiterverarbeitung durch einen Computer zu genügen. (Vergl. Kap.4.1.2.)

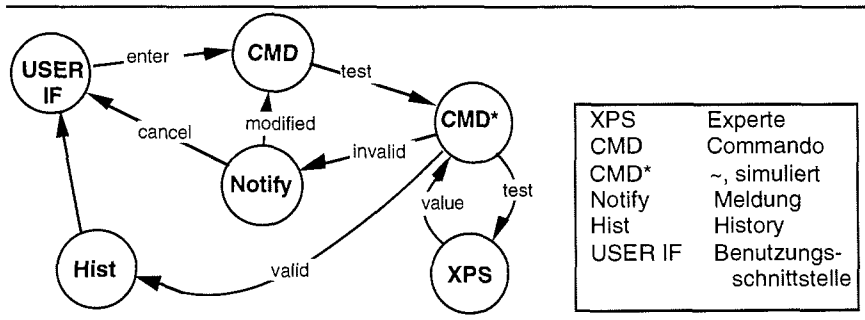


Abbildung 4.1: Das Transitions–Netz des Benutzerinterfaces aus der Analyse des Autors

```
GOAL: modellrechnen(model,alte_daten,neue_daten)
    [SELECTION_RULES:
        if neue_parameter
            then neu
            else alt
        endif
    ]
    [SELECT:
        alt: USE_METHOD:
            parameter_vorbereiten(alte_daten)
        neu:USE_METHOD:
            parameter_vorbereiten(neue_daten)
    ]
    [SELECTION_RULES:
        if interaktiv_bearbeiten
            then interaktiv
            else batch
        endif
    ]
    [SELECT:
        interaktiv:
            GOAL:
                DOACT interactiv(model,daten)
        batch:
            GOAL:
                DOACT batch(model,daten)
    ]
    USE_METHOD: plot(daten)
    ENDG
```

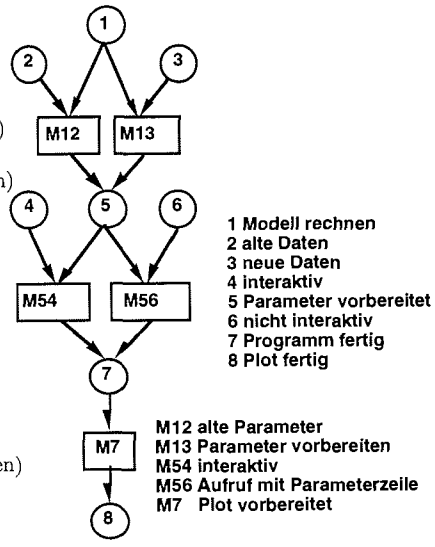


Abbildung 4.2: Beschreibung des Aktionsverhaltens eines Benutzers durch Transitionsnetze (rechts) und *GOMS***–Modell (links) (Auszug aus der Analyse von Benutzerdaten)

Tabelle 4.1: Beispiele für direkte und indirekte Zustandsvariablen

indirekte Zustandsvariable	direkte Zustandsvariable
Datei Datum/Uhrzeit	Modellvorstellung
Datei Typ	Speicherplatzanforderungen
Datei Versionsnummer	Parameter für den Programmablauf
Bearbeitungsmaske	Parameter für die benötigten Daten
Rechenleistung	Anforderung von Rechenleistung
	Compiler Optionen

Zustandsvariablen und Systemparameter

Die Netzwerkdarstellung der Benutzeraktionen bildet eine Zustandsbeschreibung des Systems *Benutzer–Maschine.* Zustandsvariablen beschreiben die Systemzustände, und deren Werte bilden die Grundlage für Entscheidungen. Die benötigten Variablen werden durch die im folgenden aufgeführten Begriffe klassifiziert.

Direkte Zustandsvariablen, die direkt vom Benutzer beeinflußt werden.

Indirekte Zustandsvariablen, die vom System bestimmt werden.

Statische Zustandsvariablen, die eine konstante Belegung während der Ausführung einer Aktionsfolge besitzen.

Dynamische Zustandsvariablen, die ihre Belegung während der Ausführung eines Planes ändern.

Lokale Zustandsvariablen, die bezüglich eines Planes lokal definiert sind.

Globale Zustandsvariablen, die global definiert sind.

Indirekte Variablen werden vom System gemäß den internen Zuständen belegt; danach werden Aktionen angenommen, die der Benutzer auszuführen beabsichtigt (Tabelle 4.1). Die Einführung von statischen und dynamischen Zustandsvariablen (Tabelle 4.2) wird beim Einsatz eines Benutzungsmodells relevant. Die Variablen erscheinen später bei der Definition von Scripten in Form von Parametern, Konstanten und Pseudo–Konstanten. Weiterhin muß der Geltungsbereich der Zustandsvariablen berücksichtigt werden (Tabelle 4.3). Die Aktionswahl wird durch die Belegung der direkten und dynamischen Variablen bestätigt oder falsifiziert.

Eine Beschreibung des Aktionsverhaltens des Benutzers muß daher die Zustandsvariablen liefern. Ausgehend von den Modellen *GOMS***, *RFA*–Netz und *ATN*–Netzen wird die Analyse bei der Entwicklung des *ResourcenManagers* wie folgt durchgeführt:

Tabelle 4.2: Beispiele für statische und dynamische Zustandsvariablen

statische Zustandsvariable	dynamische Zustandsvariable
Compiler Optionen	Datei 'creation date'
Rechenleistung	Datei Versionsnummer
Modell	aktueller Rechner
Datei Typ	aktuelles *Directory*

Tabelle 4.3: Beispiele für globale und lokale Zustandsvariablen

globale Zustandsvariable	lokale Zustandsvariable
Datei Typ	aktuelles *directory*
Datei 'creation date'	aktueller Parameter eines Scriptes
Rechenleistung	Compiler Option
Datei Versionsnummer	Programmname
Modell	aktueller Rechner

Standard: Pflichtenheft (Basisanforderungen)
In einem Pflichtenheft werden Anforderungen der Benutzer festgehalten. Hier werden erforderliche Geräte, Software, Eigenschaften, Wünsche beschrieben.

*GOMS***: Erfassung und Strukturierung der Benutzeraktionen im Vorfeld, Erkennen von Aktionsverzweigungen mit Regelerfassung, Scripte, Zustandsvariablen, die Entscheidungen beeinflussen.

Objektorientierte Analyse (OOA): Festlegung der Objekte, der Attribute, des Verhalten und der Beziehungen zwischen den Objekten.

Für weitere ausführliche Untersuchungen der Benutzerrolle und des Systemverhaltens sind die folgenden Methoden hinzuzuziehen.

RFA–**Netz:** Erfassung der Beziehungen zwischen den Objekten und Durchführung der Transaktionsanalyse. Erweiterte Darstellung der objektorientierten Beschreibung. Bestimmung von Zustandsvariablen, die Beziehungen charakterisieren.

ATN: Darstellung der Systemübergänge und Zustandsvariablen, die Zustandsübergänge beschreiben.

In der Abbildung 4.3 wird der Weg vom Benutzer, der geforderten Funktionalität, dem Ausgangssystem über die Analyse zur Wissensbasis und zum Zielsystem gezeigt.

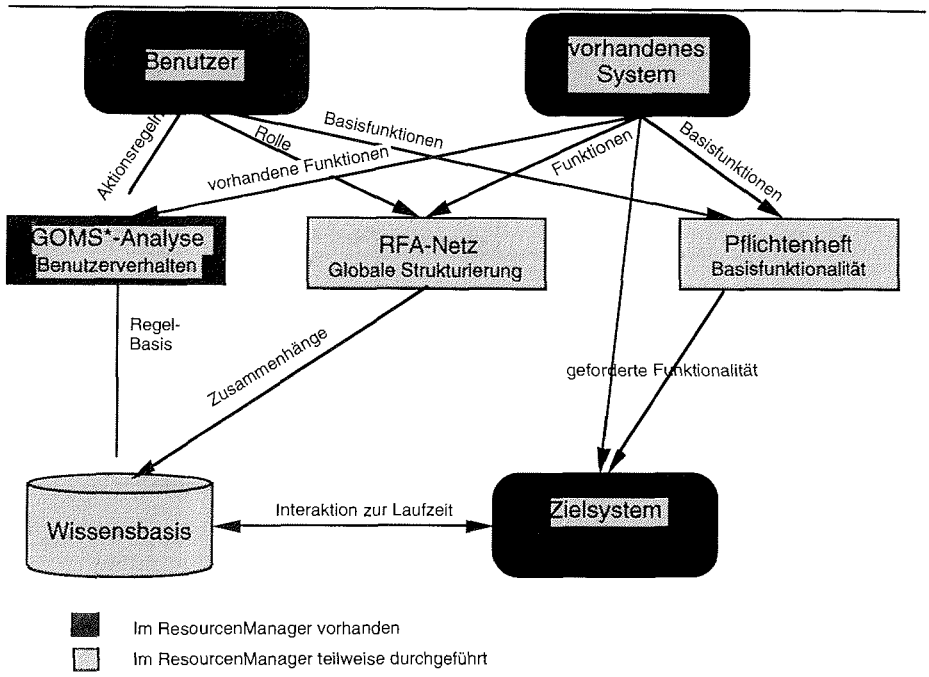


Abbildung 4.3: Das Konzept zur Analyse eines wissenschaftlichen Arbeitsplatzes am Rechner.

4.1.2 Analysemethode: *GOMS***

Die Akquisition und Repräsentation der Daten werden wir mit einem modifizierten *GOMS*–Modell durchführen. Ein Verbesserungsvorschlag des *GOMS*–Modells von Arend [Are89a] genügte nicht den hier vorgefundenen Anforderungen. Diese umfassen

- präzise Beschreibung einer Aktion mit Eingangs– und Ausgangsobjekten,
- Typisierung der Basisstrukturen,
- Variable,
- abstrakte Formulierung unabhängig vom Betriebssystem,
- Übersetzung der Aktionssequenzen zwischen den einzelnen Beschreibungen (z. B. VMS → *GOMS***→ UNIX).

Zu diesem Zweck wird auch von der separaten *SCHEMA*–Darstellung des Benutzerwissens im *GOMS**–Modell Abstand genommen. Dieses Wissen wird direkt

durch syntaktische Strukturen, die bereits teilweise in *GOMS* enthalten sind, und durch Datenstrukturen und deren Beziehungen ausgedrückt. Durch diesen Ansatz werden insbesondere Zustandsvariablen direkt darstellbar.

Prozeduren werden nicht mehr betrachtet, da deren Aufgabe von Methoden übernommen werden können. Dies gilt zunächst syntaktisch. Aber auch semantisch wird durch eine Prozedur als Zusammenfassung von Methoden, *Goals* und Operationen in einer Struktur keine zusätzliche Information erreicht.

Methoden und *Goals* werden typisiert und identifiziert. Die Typisierung gestattet es, Ergebnisse an den Aufrufer zurückzugeben. Dadurch wird leichter erkennbar, mit welchen Werten im nächsten Schritt weitergearbeitet wird. Die Identifizierung erlaubt die Verwendung derselben Methoden– oder *Goal*namen für unterschiedliche Zielbereiche. Diese Darstellung folgt dem Prinzip des *operator overloading*, das insbesondere in der Programmiersprache C++ zur Darstellung des Polymorphismus verwendet wird.

Die vollständige Notation der Syntax dieser Beschreibung, die den Namen *GOMS*** erhält, ist im Anhang A.1 zu finden.

Jede Methode *(METHOD:)*, jedes *Goal* und jeder Operator *(OPERATION:)* kann einen Wert zurückgeben. Diese Objekte besitzen eine Parameterliste. Dieser Übergabemechanismus wurde eingeführt, um deutlicher hervorzuheben, wie Parameter und Variablen eingesetzt werden. Die Typisierung/Identifizierung wird wie in der Programmiersprache C dem Objekt vorangestellt. Außerdem wurden Kommentare eingeführt, die *zeilenweise* mit der Zeichenfolge '/*' eingeleitet werden. Das Zeilenende terminiert einen Kommentar. Einige Prädikate in den Auswahlregeln *(SELECTION_RULES:)* werden nicht im *GOMS***–Modell dargestellt. Sie werden als Atome betrachtet. Das Prädikat 'is_new (model)' beispielsweise bezieht sich auf den Systemzustand mit der Aussage, ist das Modell bereits vorhanden, oder muß es erst generiert werden. Die Auswahlregeln werden durch einem Namen identifiziert. Die eigentliche Semantik der Regeln steht zwischen den Klammern '{' und '}'. Da die Beschreibung des Handlungsablaufes des Benutzers durch das *GOMS***–Modell der Ablaufplanerfassung dient, werden einige *Goals* und Operationen nur bis auf die Spezifikation des Kommandos aufgelöst, z. B. wird

GOAL: **COPY**(datei)

nicht weiter expandiert, sofern es eindeutig ist. Wenn weitere Einzelheiten zu beschreiben sind, wird das Ziel weiter aufgelöst, z. B.:

GOAL: COPY_CRAY_TO_VAX (datei)
    GOAL: **GET** (datei)
    GOAL: **SPECIFY** (datei)
    USE_METHOD: copy_ftp
ENDG.

Die Notation der mentalen Vorgänge ist hier nicht notwendig. Daher wird grundsätzlich darauf verzichtet, mentale Vorgänge in allen Einzelheiten zu beschreiben. Das obige ausführliche Beispiel wird daher zu:

GOAL: COPY_CRAY_TO_VAX (datei)
    USE_METHOD: copy_ftp
ENDG.

Zur Notation ist anzumerken, die Einrückung durch Punkte (...) wird hier von Tabulatoreinrückungen übernommen. An einigen Stellen in der Notation erfolgen Schreibweisen der Form:

METHOD: name (ident)
    ⋮
    GOAL: <ident> <type> name_des_goals (param_des_goals)
    ⋮
ENDM.

Die Punkte (:) stehen für nicht dargestellte Befehle. <type> steht für den Typ des *Goals*, <ident> für z. B. den Rechnertyp. Diese Notation soll semantisch gleiche *Goals* unter einem Namen zusammenfassen. z. B. wird '**DELETE** (file)' dann geschrieben als '<computer_type> `file` **DELETE** (file)', da die Semantik auf den Rechnern gleich ist, aber praktisch unterschiedlich ausgeführt wird. Diese Schreibweise führt zu einer Verkleinerung der Anzahl der *Goals*, Prozeduren und Methoden.

Spezielle Prädikate, die verwendet werden, sind:

DOACT	Vorbereiten eines Handlungsplanes, Beginn einer längeren Sequenz von Aktionen,
ENTER	Eingabe eines Strings oder einer semantischen Einheit, z. B. ein Systembefehl mit Parametern,
GET	GET und SPECIFY sind wie bei Arend definiert [Are89a].
SPECIFY	

Andere Prädikate werden gemäß ihrer semantischen Bedeutung eingesetzt wie:

DELETE	etwas, z. B. eine Datei, entfernen,
EXEC	etwas ausführen, z. B. ein Programm ,
COPY	etwas, z. B. eine Datei, kopieren.

Hinter den meisten Prädikaten verbergen sich bei den Benutzern weitere Makrodateien, die in der vorliegenden Analyse nur bezüglich deren Ergebnisse ausgewertet werden, z. B. *GOAL:* START (job0) wird nicht weiter aufgelöst. Entscheidend ist, daß eine bestimmte Aktion (Starten eines Jobs) durchgeführt wird und daß bekannt ist, welche Veränderungen am Systemzustand vorgenommen werden.

Bei der Betrachtung des Aktionsverhaltens mittels des *GOMS***–Ansatzes kann insbesondere festgestellt werden, wie das System bei einem Übergang vom Zustand vor der Ausführung des *Goals* in den Zustand nach der Ausführung verändert wird.

Diese Information wird später bei der Betrachtung von Übergangsfunktionen zwischen Systemzuständen interessant.

Die Beschreibung der Übergänge erfolgt in der Parameterliste. Diese Liste besteht aus zwei Teillisten. Die erste Liste enthält die Übergabeparameter. Die zweite Liste enthält die Parameter, die geändert wurden, und deren Werte. Hierzu ein Beispiel:

point *GOAL:* MacOSmoveCursor ((cursorPosition : point) (cursorPosition : point))

Der Rückgabewert des *Goals* ist die neue Position des Cursors. Ein weiteres, komplexeres Beispiel folgt.

```
/* Der Typ „file“ muß zunächst definiert werden.

type file = (name contents mode fileType)

/* Die Variablen „theUser“ und „system“ sind global definiert.
/* Die Variable „system“ wird durch die Aktionen des Benutzers gesetzt.

var theUser : user.

var system : systemDescriptor.

/* Die Zuweisung „system ← UNIX“ sei bei einer früheren
/* Aktion erfolgt.

/* UNIX unterstützt sowohl copy als auch duplicateMoveRename.

file GOAL: copyFile ((file1:file,file2:string) (newfile:file))
{ [SELECTION_RULES:
    if theUser uses copy
      then
        useCopy
        /* falls es eine Grundfunktion gibt, die eine Kopie anlegt,
        /* wie cp in UNIX
      else
        if theUser uses duplicateMoveRename
          then
            useDmr
            /* falls es nur eine Möglichkeit gibt, wie beim Apple Macintosh
          else default
        endif
    endif
  ].
```

```
  [SELECT:
    useCopy: newfile ← USE_METHOD: COPY ((file1,file2) (newfile)).
    useDmr:  newfile ← USE_METHOD: dmr ((file1,file2) (newfile)).
    default:  newfile ← file1
  ].
  copyFile ← newfile
}
file METHOD: dmr ((file1:file,file2:string) (file1,newfile:file)).
/* dmr liefert als Ergebnis den kopierten, d. h. duplizierten,
/* bewegten und umbenannten File.
{ var temp : file.
  /* temp ist der neue, durch Kopieren erzeugte file1.
  /* Er muß nicht entfernt werden.
  temp ← OPERATION: duplicateFile ((file1) (temp)).
  temp ← OPERATION: moveFile ((temp,file2.dir) (temp.name)).
  dmr ← newfile ← OPERATION: renameFile ((temp,file2) (temp.name))
}
file METHOD:COPY ((file1:file,file2:string) (newfile:file)).
{ [SELECTION_RULES:
  if system = UNIX
    then unixCopy
    else
              if system = vmsCopy
         then vmsCopy
         else
           if system = MacOS
             then macCopy
             else default
           endif
             endif
  endif
  ].
  [SELECT:
    unixCopy: newfile ← USE_METHOD: UNIXCOPY ((file1,file2) (newfile)).
    vmsCopy:  newfile ← USE_METHOD: VMSCOPY ((file1,file2) (newfile)).
    macCopy:  newfile ← USE_METHOD: MacOSCOPY ((file1,file2) (newfile)) .
    default:   newfile ← file1
  ].
COPY ← newfile
}

MacOSCOPY ((file1:file,file2:string) (newfile:file)).
/* Der Macintosh benutzt duplicateMoveRename.
{ newfile ← USE_METHOD: dmr (file1,file2)}
```

UNIX**COPY** ((file1:file,file2:string) (newfile:file)).
/* UNIX macht eine einfache Kopie.
{ newfile ← *OPERATION:* cp ((file1,file2) (newfile))}
VMS**COPY** ((file1:file,file2:string) (newfile:file)).

/* VMS macht eine einfache Kopie.
{ newfile ← *OPERATION:* copy ((file1,file2) (newfile))}

4.1.3 Anwendung des Beschreibungskonzepts

Für die Anwendung des oben entwickelten Konzepts im AWI müssen wir klären, welche Benutzer ausgewählt werden, um Daten für einen Prototypen zu gewinnen.

Die weitere Betrachtung des *ResourcenManagers* beruht auf der Berücksichtigung einer kleinen Gruppe von Wissenschaftlern des AWI, die zu einer aktiven Mitarbeit bereit war. Nach Bearbeitung der Daten wurde eine Kontrollgruppe betrachtet, um damit das Konzept zu validieren und vergleichbare Ergebnisse zu erhalten. Der Zielgruppe wurde ein kleiner Fragebogen vorgelegt, auf dem Wünsche bezüglich der Arbeit mit dem Rechner geäußert werden konnten. Außerdem sollte jeder Benutzer mittels eines Flußdiagramms seine Arbeit am Rechner beschreiben.

Ein sinnvoller Datenbestand, durch dessen Analyse der Benutzer nicht gestört wird, sind die vom Benutzer erstellten Makrodateien. Weitergehende Informationen, wie diese Makrodateien eingesetzt werden, mußten durch Befragen geklärt werden.

Die Analyse der Makrodateien führte zu Aktionsplänen, die mit dem *GOMS***–Modell dargestellt werden. In dieser Modellierung sind Verzweigungen in der Aktionsfolge erkennbar. Die Regeln, nach denen verzweigt wird, sind aus dem Text ablesbar. Durch Befragen des Benutzers konnten weitere Handlungsregeln gefunden werden.

Zusammengefaßt folgte die Erfassung der Informationen über die Benutzer den fünf angegebenen Schritten:

- Auswahl einer Benutzergruppe,
- kurze Befragung,
- Pflichtenheft aus bekannten Daten,
- Analyse der Makrodateien der Benutzer,
- Benutzerbefragung zu Makrodateien.

Vergleichbare Analysen, z. B. von Heeg [Hee88] sowie Hertweck und Stöhr [HS88], verwenden großangelegte Umfragen, um die erforderlichen Daten zu erhalten. Mit dem hier vorgestellten Konzept kann im wesentlichen auf einfache Analysen zurückgegriffen werden, da mit dem realisierten System weitere Informationen erfaßt werden können.

Tabelle 4.4: Auswahlkriterien für die Ziel- und Kontrollgruppe bei der Erstellung eines Benutzungsmodells

Kriterium	Beschreibung
Kenntnisse	Die Kenntnisse der Benutzergruppe im Umgang mit Rechenanlagen sollten möglichst in allen drei Kenntnisgruppen vertreten sein. So kann geprüft werden, ob das erstellte System den Anforderungen aller drei Gruppen gerecht wird.
Aktivitäten	Die Aktivitäten der Gruppen in bezug auf den Einsatz eines Rechners sollten möglichst getrennt sein, d. h. keine Anwendungen, die von Standardprogrammen erfüllt werden können.
Kooperationsmöglichkeit	Die Gruppen sollten bereit sein, eine sich anschließende Testphase bereitwillig mitzumachen, d. h. insbesondere, daß die tägliche Arbeit diesen Spielraum zuläßt.
Unterstützung	Die Unterstützung im gewählten Arbeitsbereich sollte nicht oder nur sehr unkomfortabel durch bisherige Anwendungen und Standardprogramme möglich sein.

4.1.4 Auswahl der Benutzergruppen

Die Begriffe Ziel- und Kontrollgruppe implizieren den Vergleich von Ergebnissen aus statistischen Erhebungen. Die besondere Situation am AWI erforderte jedoch ein anderes Vorgehen, da die Anwender nicht in dem erforderlichen Maße für statistische Erhebungen zur Verfügung standen. Die Begriffe Ziel- und Kontrollgruppe beschreiben hier zwei Benutzergruppen, deren Anforderungen und Benutzerverhalten analysiert werden, um ein System zu erhalten, das beiden Gruppen gerecht wird. Das realisierte System wurde für die Zielgruppe unter der Kontrolle der besonderen Anforderungen und Zielsetzungen der Kontrollgruppe entwickelt. Die Kontrollgruppe stellt nach einem Entwicklungszyklus die zu verifizierenden Eigenschaften bereit. Dieser Weg wurde beschritten, um nicht durch die speziellen Anforderungen der Zielgruppe ein einseitig ausgerichtetes System zu entwickeln. Die Auswahl der Ziel- und Kontrollgruppe erfolgte nach mehreren Kriterien (Tabelle 4.4). Hier werden die charakterisierenden Merkmale dargestellt.

Die größte Gruppe von Wissenschaftlern im AWI, der Fachbereich Biologie, die die geringsten Kenntnisse im Umgang mit Rechnern besitzt, konnte nicht als Zielgruppe gewonnen werden. Informationen über die Arbeitsweise dieser Wissenschaftler, die in Diskussionen und Vorgesprächen über Konzepte des Einsatzes von Arbeitsplatzrechnern unabhängig von der vorliegenden Arbeit erhalten wurden, dien-

ten daher zur Validierung des Befragungskonzepts. Die gewonnenen Informationen beeinflußten die Realisierung, da das System nicht nur nach einem Pflichtenheft für eine Gruppe von Wissenschaftlern erstellt wurde, sondern auch für andere Handlungspläne und Anforderungen offen gehalten werden mußte.

Damit ein Test des Systems am Arbeitsplatz erfolgen konnte, mußte eine Gruppe ausgewählt werden, die bereit war, das System im täglichen Einsatz zu testen. Die Wahl fiel auf den Fachbereich Physik/Ozeanographie/Meteorologie. Diese Benutzer sind im Umgang mit Rechnern bereits sehr erfahren. Dies verkürzt die Einarbeitungszeit und gibt die Möglichkeit, den Expertenmodus (*expert*-Modus) zu überprüfen.

Im folgenden werden die Grundsätze zur Auswahl einer Ziel- und Kontrollgruppe beschrieben. Am Beispiel des AWI werden die Schwierigkeiten deutlich, die diese Auswahl einschränken.

Konzept der Zielgruppe

Die Zielgruppe ist der Grundstock der Entwicklung des Systems. Die Ergebnisse der Untersuchungen beeinflussen die Entwicklung des Systems durch die Auswahl an unterschiedlichen Verfahren der Benutzer.

Die Zielgruppe sollte entweder den drei Benutzerklassen (*novice, intermittend, expert*) entsprechen oder einen guten Querschnitt aus den Benutzern des Instituts bilden.

Im ersten Fall können einzelne Konzepte anhand des tatsächlichen Kenntnis- und Erfahrungsstandes überprüft werden. Im zweiten Fall können statistische Ergebnisse über Akzeptanz gewonnen werden.

Die ausgewählte Zielgruppe, Fachbereich Physik des AWI, besteht aus Wissenschaftlern, die im Umgang mit Rechnern versiert sind. Umfangreiche manuelle Tätigkeiten bei der Entwicklung eines Programms werden von diesen Benutzern mit aufwendigen Mitteln zu Makropaketen zusammengefaßt.

Die Kenntnisse im Umgang mit dem System entsprechen meist denen eines Experten. Dies liegt unter anderem auch daran, daß auf den Rechenanlagen im AWI ein operateurfreier Betrieb durchgeführt wird. Der Benutzer ist gefordert, auch die kleinsten Tätigkeiten, wie z. B. Einlegen eines Magnetbandes, selbst auszuführen.

Auf der anderen Seite sind die Benutzer nicht bereit, mit diesen Tätigkeiten viel Arbeitszeit zu verlieren. Daher wird eine Unterstützung in anderer Form notwendig. Bemerkenswert ist jedoch, daß die graphischen Oberflächen von dieser Benutzergruppe zunächst abgelehnt wurden, da scheinbar nicht mehr die gesamte Funktionalität verfügbar ist und der Gewöhnungseffekt eine sehr große Rolle spielt. Die früher aufgestellte Forderung nach Überzeugung des Benutzers wird hier sehr deutlich.

Der Novizenstatus einer Benutzergrupe kann aus den Erfahrungen, die im AWI im Rahmen der Benutzerbetreuung gemacht wurden, abgeleitet werden. Insbesondere wenn man berücksichtigt, daß das Konzept des Apple Macintosh von der Mehrzahl der Benutzer akzeptiert wird.

Eine andere Zielgruppe, die mit Rechnern weniger vertraut ist, kommt im speziellen Fall dieser Arbeit dennoch nicht für die Untersuchung in Frage, da die Bereitschaft, eine neue Oberfläche zu testen, nicht vorhanden ist. Die Fähigkeit der ausgewählten Gruppe, sich auf veränderte Situationen einstellen zu können, macht es möglich, diesem Benutzerkreis eine neue Benutzungsoberfläche anzubieten. Um diese Gruppe einzubeziehen, wären insbesondere Testsituationen vorzusehen, wie sie in der Literatur beschrieben werden.

- Young, MacLean [YM88]: Analyse der Auswahlmethoden eines Benutzers bei der Auswahl verschiedener Lösungsmöglichkeiten für eine Aufgabe
- Greenberg, Witten [GW88]: Analyse der Methoden, nach denen der Benutzer seine Aktionen wiederholt
- Arend [Are89b]: Analyse des Einflusses von Visualisierung und Kommandostruktur auf das Problemlösen
- Kühn, Laurig, Schmidt [KLS85]: Analyse der Eingabe über Tastatur: Leistungen und Strategien
- Kunkel [Kun90]: Wird bei der Visualisierung der Daten übertrieben?
- Roberts, Moran [RM83]: Untersuchung verschiedener Texteditoren auf ihre Leistungsfähigkeit und Erlernbarkeit
- Card, Moran [CM80]: Analyse der Ausführungszeiten im Expertenmodus

Als Fazit kann notiert werden:

1. Die Zielgruppe soll einen Querschnitt der Kenntnisse und Erfahrungen der Benutzer liefern.
2. Die Zielgruppe soll die drei Bereiche *novice, intermittend, expertuser* abdecken.
3. Die Zielgruppe soll zur Mitarbeit bereit sein.
4. Die Datenerhebung sollte mit anderen Mitteln als durch umfangreiche Befragung möglich sein.

Konzept der Kontrollgruppe

Mit einer Kontrollgruppe, die in die erstmalige Analyse nicht mit einbezogen wurde, wird das Konzept überprüft. Die Kontrollgruppe wird ebenso analysiert wie die Zielgruppe. Mit diesem Ergebnis kann die Realisierung nach jedem Entwicklungszyklus überprüft werden. Insbesondere kann jeder Schritt der Implementierung auf Konsistenz mit den Anforderungen der Kontrollgruppe überprüft werden. Weiter wird

durch die Arbeit der Ziel- und der Kontrollgruppe mit dem Zielsystem relevantes Material zur Verbesserung des Systems erfaßt.

Wird eine Kontrollgruppe zum aktiven Test der Realisierung ausgewählt, d. h. das System dieser Grupppe zur Verfügung gestellt, muß die Bereitschaft der Gruppe zur Arbeit mit dem System geprüft werden. Die Kontrollgruppe für den Test des erarbeiteten Systems sollte daher nach den gleichen Kriterien wie die Zielgruppe ausgewählt werden. Die Bereitschaft, ein neues System zu testen, muß vorhanden sein. Durch diese Einschränkung werden die im Umgang mit Rechnern unerfahrenen Personen nahezu ausgeschlossen.

Im vorliegenden Fall konnten einige Besonderheiten berücksichtigt werden. Die laufenden Erweiterungen der Anlage im AWI erzeugen ein in Verhalten und Ausrüstung nicht statisches System, d. h., der Benutzer wird ständig mit Neuheiten konfrontiert. Mit Betriebssystemmitteln ist eine vernünftige Unterstützung des Benutzers nicht mehr zu erzielen. Der Benutzer wird daher seine Aufgaben nicht mehr ohne größeren Aufwand erledigen können. Es erfolgt eine Anpassung *des Benutzers* an die neue Umgebung.

Die Zusammensetzung der Gruppe ist über die Zeit nicht konstant. Die Fluktuation der Mitarbeiter hatte zur Folge, daß sich zwar die Intention der Gruppe nicht änderte, d. h., Arbeitsweisen der Benutzer waren in weiten Bereichen gleich. Jedoch waren die einzelnen Mitglieder der Gruppe in einem anderen Fähigkeitsbereich anzusiedeln.

Die neu hinzugekommenen Mitglieder der Zielgruppe konnten daher wieder als Novizen oder *intermediate user* angesehen werden; die Individuen wurden zwangsläufig ausgetauscht. Mit dieser Einschränkung war es möglich, daß Ziel- und Kontrollgruppe identisch blieben. Ein Urteil über Vereinfachung und Leistungsfähigkeit ist dann ebenso signifikant wie die Akzeptanz des Systems.

Die Anforderungen und Arbeitsweisen der anderen Benutzergruppen waren nur über Diskussionen und individuelle Befragungen zugänglich. Diese Ergebnisse zeigten jedoch eher auf, welche Applikationen gewünscht werden, als Hinweise, wie mit dem Rechner gearbeitet werden kann. Diese Gruppen sind als Novizen einzustufen, denen mit einer möglichst einfach zu bedienenden Oberfläche Unterstützung gewährleistet wird.

Als Fazit kann notiert werden:

- Die Punkte 1 – 4 der Zielgruppe gelten hier ebenfalls.
- Hinzu kommt: Die Kontrollgruppe kann die Zielgruppe mit einschließen.
  Um signifikante Ergebnisse zu erhalten, sollte die Kontrollgruppe von der Zielgruppe verschieden sein. Die obige Darstellung der Situation im AWI erlaubt es demgegenüber, die ehemalige Zielgruppe mit in die Kontrollgruppe hineinzunehmen, da sich die personelle Zusammensetzung und damit das Anforderungsspektrum mit der Zeit verändert hat.

Als Maßstab für Anforderungen bei Bedienungskomfort, Simplizität und intuitiv eingängiger Handhabung dienen die Ergebnisse der Diskussionen mit Anwendern

und die Tatsache, daß der Apple Macintosh II im Rahmen dieser Punkte im AWI spontane Akzeptanz erfahren hat.

4.1.5 Zusammenfassende Darstellung des Konzepts

Die Entwicklung einer Benutzungsoberfläche für das AWI erfordert ein mehrstufiges Vorgehen. Die Ergebnisse für den Fall des *ResourcenManagers* sind in Kapitel 6.3 dargestellt. Wir fassen die wesentlichen Punkte, die die Erfassung der notwendigen Daten ermöglichen, hier noch einmal zusammen.

Zunächst werden die Informationen als Beschreibung des *Ist*-Zustands und der Anforderungsanalyse in einem Pflichtenheft gesammelt.

Erstellung eines Pflichtenheftes	Erfassung der Benutzerforderungen Erfassung des *status quo* Erfassung der Basisfunktionalität Erfassung der gewünschten Funktionalität

Sofern nicht bereits in der Anforderungsanalyse bestimmte Anwender angesprochen werden, müssen nun die Ziel- und die Kontrollgruppe ausgewählt werden.

Auswahl einer Zielgruppe	Kenntnisse Aktivitäten Kooperationsmöglichkeiten Unterstützung
Auswahl einer Kontrollgruppe	wie Zielgruppe

Zu einer guten Analyse gehört auch die Identifikation und Beschreibung von Arbeitsabläufen. Dabei kann vielfach festgestellt werden, daß bestimmte Stereotypen existieren, die als Grundlage für die individuelle Gestaltung benutzt werden können.

Erfassung von Stereotypen	Aktionspläne Datenhaltung Datenbearbeitung

Aus den Stereotypen und der Anforderungsanalyse können veränderliche Parameter, die Aktionen näher spezifizieren, und festgelegte Regeln, nach denen ein Benutzer agiert, abgelesen werden.

Erfassung der Parameter (Variablen)	variable Parameter statische Parameter indirekte Parameter direkte Parameter
Erfassung von Regeln	Aktionsregeln Adaptierungsregeln Unterstützungsregeln (erzwingen Hilfe) Fehlerbehandlung Datenhaltung Bearbeitungsregeln

Die Daten werden mittels der in Kapitel 4.1.3 dargestellten Verfahren strukturiert und dargestellt. Daran schließt sich ein Implementierungsentwurf an, der mit einer geeigneten Programmierumgebung realisiert wird.

Das Konzept der Ziel– und Kontrollgruppe ermöglicht eine Simulation des Systems mit den aus den Untersuchungen entnommenen Vorgaben. Ist diese Simulation erfolgreich, wird das System an die Kontrollgruppe weitergereicht und dann von dieser getestet.

4.2 Benutzungsmodell für den wissenschaftlichen Arbeitsplatz am AWI

Das Benutzungsmodell ist die Zusammenfassung von Aktionsregeln und Handlungsplänen des Benutzers. Wir stellen hier die in der Literatur bekannten Untersuchungen und deren Ergebnisse vor. Zahlreiche Untersuchungen betrachten entweder den Novizen[2] [Shn83, Shn85] oder den Experten [CM80]. Fürst und Merle zeigen, daß die Betrachtung von Benutzern mit einem weitgefächerten Erfahrungshorizont bessere Ergebnisse für die Modellierung liefert [FM86]. Der hier beschriebene Ansatz zur Wissensstrukturierung von Balzert ist vom Erfahrungsstand des Benutzers unabhängig [Bal88].

Die Aufteilung des Wissens nach Balzert ist für die konzeptionelle Strukturierung der erfaßten Daten nützlich. Diese Strukturierung gibt dem Entwickler Richtlinien, nach welchen Angaben beim Benutzer gesucht werden muß, und erleichtert den Aufbau von Wissensbasen.

Eine Aufgliederung der Schnittstelle in Ein–/Ausgabe–Schnittstelle, Dialogbereich und Anwendungsbereich wird nach dem IFIP Model (Abb. 4.4) durchgeführt (Siehe auch das Seeheim–Modell *(Presentationlayer, dialog layer, application layer)* [Pfa85].). Anwendungszustand und Dialogzustand werden mittels einer Transaktionskomponente mit der Dialogkomponente und der Anwendungskomponente gekoppelt. Wichtig ist die Einführung einer Expertenkomponente, die das Wissen für die Transaktionskomponente bereithält. Die Transaktionskomponente kann durch ein Transitions–/Zustands–Netz repräsentiert werden. Die Trennung in Dialog und Transaktion erlaubt ein logisches Vorgehen auf zwei Ebenen – reine Dialogarbeit und Regel- bzw. Wissensüberprüfung.

Die Expertenkomponente besteht nach Waldhör aus einer Menge von Interaktionsexperten, die in die vier Experten Benutzungsmodell, Dialogwissen, Systemselbstbildnis und Interaktionsmethoden zerfallen [Wal89]. In dieser Arbeit wird, im Gegensatz zu Waldhör, keine Menge von gleichberechtigten Interaktionsexperten

[2]Die zitierten Arbeiten von Shneiderman sind ein Beispiel für die Darstellung dieses Themas. Die *direkte Manipulation* wird als besonders geeignet angesehen, um Novizen an ein System heranzuführen, da wenig Detailwissen über interne Vorgänge notwendig ist und das System intuitiv bedient werden kann.

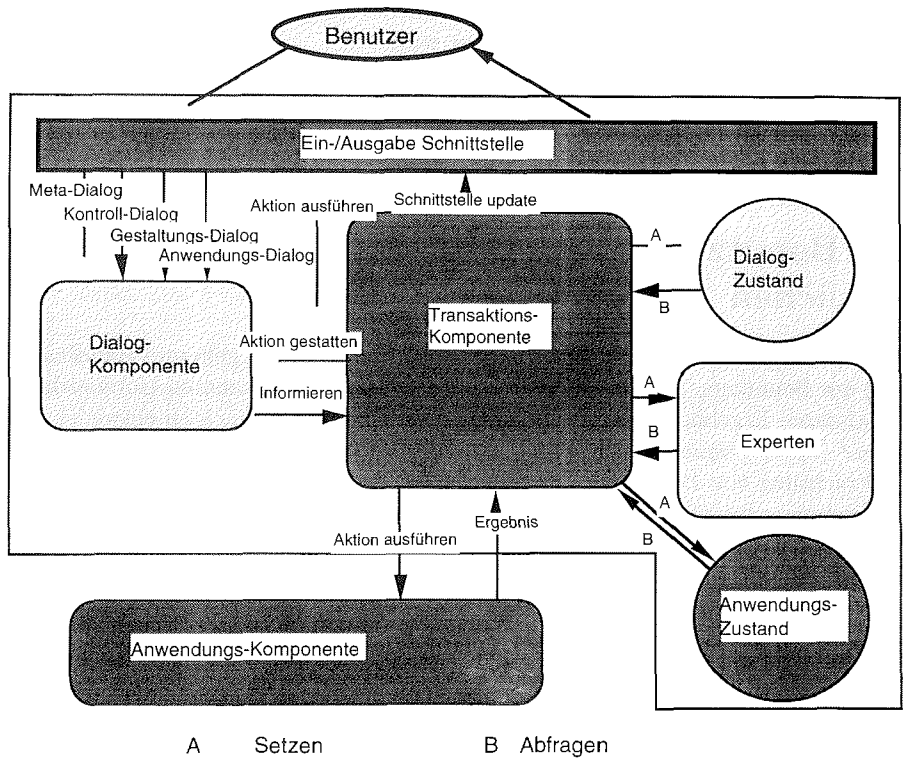


Abbildung 4.4: Verändertes IFIP Modell, auf die Bedürfnisse der vorliegenden Arbeit angepaßt. (Erläuterungen siehe Text.)

betrachtet. Ausgehend von den Anforderungen des Benutzers ist das Benutzungsmodell die zentrale Einheit, die mit den anderen Einheiten kommunizieren muß. Die Reihe der Experten wurde hier um das Anwendungswissen erweitert und stellt Detailwissen über die Applikation zur Verfügung. Dieses Wissen wird benötigt, da das System auf bereits bestehende Applikationen zurückgreifen muß (heterogene Umgebung). Diese Applikationen verlangen z. B. bestimmte Regeln beim Aufbau eines an sie abgesandten Kommandos. Die verbindende Rolle zwischen den Experten wird von der Transaktionskomponente übernommen.

4.2.1 Wissen

Konventionen

Konventionen bezeichnen Festlegungen, die zwischen Benutzern einer Gruppe getätigt werden, um die Kommunikation aufrecht zu erhalten und zu erleichtern. Die Erfassung kann durch Vergleich von Symboldefinitionen auf den Rechnern erfolgen, z. B. Durchsicht der `login`–Dateien.

Der einzelne Benutzer hat bestimmte Vorlieben für seinen Arbeitsablauf. In diesen Bereich fallen z. B. die Festlegung von Symbolen und der Aufbau von Makro–Dateien. Der gleiche Arbeitsgang wird bei verschiedenen Benutzern von unterschiedlichen Konventionen begleitet. Wenige sind hiervon in einem Pflichtenheft unterzubringen, da die Berücksichtigung der Individualität ein Pflichtenheft unübersichtlich werden läßt. Konventionen lassen sich auch für Benutzergruppen erfassen.

Strukturelles Wissen

Das strukturelle Wissen einer Benutzergruppe ist die gemeinsame Arbeitsbasis der Gruppe. Die Kenntnisse über die Zusammenhänge, die die Arbeit der Gruppe betreffen, werden verfügbar gemacht, so daß eine Aufgabe ohne großen Erklärungsaufwand durchgeführt werden kann.

Die Kenntnisse der Benutzer von strukturellen Zusammenhängen beziehen sich auf Arbeitsabläufe, Datenverteilung, benötigte Rechner und eingesetzte Betriebssysteme. Die Erfassung kann durch den Vergleich von Makro–Dateien der einzelnen Benutzer und durch Befragung erfolgen.

Der Benutzer besitzt eigene Verfahren, um Daten zu ordnen und Arbeitsabläufe zu organisieren. Dieses individuelle Wissen ergänzt das Wissen über die Gruppe.

Faktenwissen

Das Faktenwissen umfaßt die Kenntnisse der Benutzer von Einzelheiten, die Kenntnisse, was zu tun oder wie ein Schritt auszuführen ist, wo Daten zu finden sind etc. Zum Faktenwissen zählt auch die Kenntnis von Optionen zu Betriebssystembefehlen. Diese Kenntnisse sind für vielgenutzte Befehle ausgeprägt. Faktenwissen der Benutzer kann sehr leicht erfaßt werden. Statistiken über Hilfs–Funktionen oder Fehler können Auskunft geben.

Das Faktenwissen einer Benutzergruppe wird durch die Information gebildet, welche Arbeitsmittel zur Verfügung stehen, wie ein Arbeitsmittel angesprochen und mit ihm gearbeitet wird. Als Beispiel sei die Bereitstellung einer Bibliothek mathematischer Funktionen genannt.

Individuelles Faktenwissen wird durch Arbeitsmittel gebildet, die dem Benutzer allein bereitstehen, so z. B. spezielle Programmbibliotheken. Der Benutzer ist für diese Mittel allein verantwortlich.

Problemlösungswissen

Jeder Benutzer hat Kenntnis davon, welche Handlungen auf dem Rechner auszuführen sind, d. h., vollständige oder Teil–Handlungspläne werden erfaßt. Das Problemlösungswissen des Benutzers wird nicht vollständig durch eine Befragung des Benutzers oder einer Benutzergruppe erfaßt werden können. An dieser Stelle sind Ansätze wie die von Schwab [Sch89b], Hoppe [Hop88a] und *GOMS** [Are89a] o. ä. gefordert.

Das Problemlösungswissen einer Gruppe kann am einfachsten an einem Beispiel erläutert werden. Der Einsatz von Bildsequenzen (Filmen) zur Verfolgung von Veränderungen im ausgewerteten numerischen Modell ist ein Verfahren, um das Modell zu überprüfen. In einem schnell ablaufenden Film lassen sich Fehler leichter erkennen, als durch den Vergleich von Einzelbildern. Das Problemlösungswissen liefert insbesondere Informationen, wie ein Film aus den Daten mit einem gewählten Modell erzeugt wird.

Regeln

Fakten dienen als Grundlage für Handlungsentscheidungen (Regeln). Regeln entscheiden über den Ablauf eines Handlungsplanes. Die Befragungen der Benutzer mittels Modellen wie *GOMS** liefern Grundregeln, nach denen ein Benutzer seine Aufgabe am Rechner ausführt. Mit diesem Ansatz werden jedoch nur Handlungspläne erfaßt, die dem Benutzer zur Zeit der Befragung bekannt sind. Jede Veränderung einer Regel wird das System zum Scheitern verurteilen, da implementierte Handlungspläne nicht mehr gültig sein müssen.

Ein Benutzer entscheidet bei der Ausführung von Aktionen nach bestimmten Fakten, welche Aktion folgen wird. Diese Fakten bestimmen für die gesamte Gruppe gleichlautende Aktionssequenzen. Dies schließt insbesondere ein, welche Programme für einen vorliegenden Anwendungsfall in jedem Falle zu benutzen sind. Regeln können auch für Benutzergruppen zutreffen.

4.2.2 Erfassung des Wissens

Die weitgefächerte Darstellung des Wissens bedeutet in keinem Fall, daß dieses Wissen immer in den entsprechenden Kategorien verfügbar ist. Die Kategorisierung ermöglicht uns eine Strategie der Erfassung und der Implementierung, d. h.:

- Interviews, um eine globale Übersicht zu erhalten, d. h. tatsächliches Wissen und geplante Fakten, Regeln, Strategien,
- *GOMS***–Analyse der Makro/Script–Dateien, um die tatsächlich vorhandenen Daten zu erhalten, d. h. tatsächlich genutztes Wissen,
- Kategorisierung, um mit effizienten Hilfsmitteln das Wissen bereitzustellen, d. h., logische Fakten in einer Regelbasis und symbolische Fakten in einer Datenbank zu halten.

4.3 Dialoghistorie

Die Bedeutung eines Protokolls der Benutzeraktionen ist bereits in den vorhergehenden Kapiteln erläutert worden. Eine Strategie zur effizienten Erfassung der Dialoghistorie mit Einsatz von *UnDo/ReDo*–Mechanismen, Unterstützung des Benutzers durch aktive und passive Hilfe und Methoden zur Handlungsplanverfolgung werden wir hier erarbeiten.

Definition 1 *Eine Dialoghistorie ist ein Objekt, mit dem die Aktionen des Anwenders protokolliert werden. Es stellt Operationen zur Verfügung, mit denen Aktionen zurückgenommen (UnDo), wiederhergestellt (ReDo) oder noch einmal ausgeführt werden können (Again).*

Mehrere verschiedene Möglichkeiten der Dialogdatenspeicherung sind verfügbar:

- Aktionsprotokoll
- Zustandsprotokoll
- Funktionsprotokoll

Die Protokolle können linear oder als Baum oder auch als allgemeiner Graph strukturiert sein. Auf dieser Struktur gibt es Operationen, die mit den Informationen in der Historie arbeiten z. B. *UnDo*. Die Verwaltung einer Dialoghistorie, der Einsatz und die Problematik von *UnDo/ReDo*–Operationen wurden bereits von Herczeg, Rathke und Yang mehrfach dargestellt [Her86a, Rat86, Rat89, Yan88a, Yan88b]. Herczeg führt ein *UnDo*–Rahmensystem ein. Dabei wird gleichzeitig gezeigt, daß ein allgemeines *UnDo* nicht möglich ist. Rathke betrachtet *UnDo/ReDo* bei der Navigation in einer Literaturdatenbank. Eine Erweiterung der *UnDo/ReDo*–Funktionalität wird von Rathke vorgestellt [Rat89]. In der genannten Arbeit wird ein Historierahmensystem beschrieben, das den Softwareentwickler von der Notwendigkeit befreit, ein komplettes *UnDo/ReDo*–System zu erstellen. Es sind nur die für die Anwendung spezifischen Komponenten einzusetzen. Die Arbeiten von Yang beschreiben die Entwicklung eines *UnDo*–Systems durch Formalisierung der Vorgänge des *UnDo* und *ReDo* [Yan88a, Yan88b]. Die Arbeiten von Archer, Conway und Schneider und Vitter beschäftigen sich mit der Implementierung eines Scripteditors, mit dem eine Dialoghistorie vom Benutzer modifiziert werden kann [ACS84, Vit84].

4.3.1 Protokolle

Aktionsprotokoll

In einem Aktionsprotokoll werden die Aktionen des Benutzers mit allen notwendigen Parametern protokolliert. Man erhält eine Liste von ausgeführten Kommandos.

Die Speicherung der ausgeführten Aktionen läßt keine Rückschlüsse auf den Systemzustand zu. Einzig die Daten, mit denen gearbeitet wird, müssen vor der

Ausführung der Aktion vorhanden sein (Ausnahme: Erzeugungsbefehle, z. B. `copy`, `create`).

Eine Möglichkeit, die fehlenden Informationen bereitzustellen, ist eine Wissensbasis, die für jeden Befehl die modifizierten Daten beschreibt. Da abgeschlossene Zustände des Systems beschrieben werden, sind die Unterschiede der Zustände aber nicht die expliziten Änderungen bekannt.

Zustandsprotokoll

In einem Zustandsprotokoll werden die Zustände des Systems abgelegt, die durch die Ausführung einer Aktion erzeugt wurden.

Eine Mitschrift der Zustände allein läßt keine Rückschlüsse auf die Aktionen des Benutzers zu, da Daten auf verschiedene Arten gewonnen und gelöscht werden können[3].

Funktionsprotokoll

Mit dem Funktionsprotokoll wird versucht, die Nachteile der beiden vorgenannten Verfahren zu beseitigen. In einer Historiestruktur wird sowohl eine Information über den Zustand als auch über das ausgeführte Kommando abgelegt.

Auch hier ist eine Wissensbasis nötig, die die Veränderung von Daten beschreibt. Jedoch kann diese Wissensbasis einfacher gestaltet werden, da viel Information bereits durch den Befehlsaufruf bereitgestellt wird. Diese Information entspricht der eines Aktionsprotokolls. Jedoch ist hier ein Vergleich des vorhergehenden mit dem nachfolgenden Zustand möglich, so daß genauere Angaben über Veränderungen gemacht werden können.

4.3.2 Historiebäume oder Aktionslisten

Die vorgestellten Verfahren, eine Dialoggeschichte aufzuzeichnen, haben Vor– und Nachteile. Eine elegante Lösung ist die Sicherung der Zustände des Systems, da der Benutzer im Falle der Rücknahme einer Aktion nur einen Zustand auswählen muß, der dann vom System wieder generiert wird. Eine Wiederholungsfunktion ist hier ohne weiteres nicht möglich, da keine Informationen über die durchgeführten Aktionen bereitliegen. Insbesondere wird durch die Speicherung vollständiger Systemzustände die Datenhaltung unrealistisch.

Ein Ausweg ist die Haltung der Zustandsänderungen entsprechend einem Funktionsprotokoll. Hier werden die Aktionen, analog zum Aktionsprotokoll, und die hervorgerufenen Änderungen des Systemzustands gesichert. Dieses Verfahren ermöglicht die Beschreibung eines Systemzustands und gleichzeitig die Anwendung bzw. Betrachtung der durchgeführten Aktionen.

[3]Eine Datei kann eine Ausgabe eines Programms sein oder mit einem Editor erstellt werden. Der Zustand des Systems kann *vor* und *nach* der Ausführung eines Befehls identisch sein.

Die Frage nach der Art der Datenhaltung in dieser Dialoghistorie stellt sich, wenn Aktionsfolgen analysiert werden. Die Aktionen eines Benutzers bilden lineare Folgen von Handlungen. Innerhalb dieser linearen Aktionslisten kann *UnDo/ReDo* einfach eingeführt werden.

Bäume entstehen, wenn der Benutzer die Möglichkeit hat, zu einem Zeitpunkt verschiedene Aktionspfade anzuwenden. Dies wurde von Rathke im Rahmen einer Literaturrecherche gezeigt [Rat86]. Da in diesem Beispiel keine Daten verändert wurden, ließ sich auch der Systemzustand einfach beschreiben.

Kellerman et al. beschreiben im Rahmen des Projektes *X-AiD* eine Dialoghistorie, die die lineare Folge von Benutzeraktionen auf einer übergeordneten Ebene in eine logische Struktur umsetzt [KHTF87a]. Dabei wird darauf geachtet, daß parallel ausführbare Aktionen im Baum in parallelen Zweigen auftreten und voneinander abhängige Aktionen in einer linearen Folge dargestellt werden. Ähnliche Betrachtungen wurden von Archer, Conway und Schneider [ACS84] und, erweitert, von Vitter [Vit84] vorgenommen.

In der vorliegenden Arbeit werden wir die Probleme bei der Parallelisierung und Sequentialisierung von Aktionen in der Dialoghistorie untersuchen. Die Voraussetzungen für parallele Darstellung und Abhängigkeiten bei der Beibehaltung *eines* Systemzustands werden herausgearbeitet.

4.3.3 UnDo/ReDo im System

Die Einführung der *UnDo/ReDo*–Fähigkeit in eine Benutzerschnittstelle ist gekennzeichnet durch

- die Unmöglichkeit eines allgemeinen *UnDo*–Ansatzes,
- fehlende inverse Operationen einiger Aktionen,
- den Speicheraufwand für ein mehrschrittiges *UnDo*.

Die Einführung geeigneter *UnDo/ReDo*–Mechanismen wurde bereits von Herczeg [Her86a], Rathke [Rat86, Rat89] und Yang [Yan88a, Yan88b] ausführlich diskutiert. Während Herczeg und Yang eine allgemeine Formalisierung des *UnDo* auf linearen Handlungsabfolgen zeigen, wurde von Rathke [Rat86], bedingt durch das Zurückführen einer Dialoghistorie auf Systemzustände, die Einführung einer Baumstruktur notwendig. An dieser Stelle wird eine, der Beschreibung der linearen Handlungsabfolge analoge, Formalisierung einer Baumstruktur vorgenommen. Die Schwierigkeiten, die bei einer baumartigen Historie mit gleichen Zuständen in parallelen Ästen auftreten, konnten bei Rathke [Rat89] unberücksichtigt bleiben, da parallele Zustände, bedingt durch die Aufgabe, nie gemeinsame Datenbestände enthalten, so daß eine Kollision ausgeschlossen ist. Die von Herczeg, Rathke und Yang formal beschriebenen Systeme sind Rahmensysteme für *UnDo/ReDo*.

Ansatz nach Herczeg

Bevor die Formalisierung beschrieben wird, erfolgt eine kurze Definition des *UnDo*. Herczeg definiert drei verschiedene *UnDo*–Verfahren [Her86a]:

Retract–UnDo: Wiederherstellung des Systemzustands *vor* der Ausführung des letzten Befehls und Beseitigen des gelöschten Zustands aus der Dialoghistorie.

Travel–UnDo: Wie *Retract–UnDo*, aber *ohne* Löschen des Zustands aus der Dialoghistorie. Ein *ReDo* ist möglich.

Recall–UnDo: Dieses *UnDo* ist auf sich selbst anwendbar und erweitert die Dialoghistorie

Die Einführung von *UnDo/ReDo*–Mechanismen erfordert die Mitschrift des Dialogs. Diese Dialoghistorie wurde in [Her86a] bereits für eine einfache, linear geordnete Folge von Zuständen oder Systemfunktionen dargestellt.

Herczeg betrachtet die verschiedenen Verfahren des *UnDo/ReDo* auf der Basis der Implementierung der Verfahren. Dabei zeigt er die Äquivalenz und Existenz von Funktionen, die einem einfachen, schrittweisen bzw. einem springenden, mehrstufigen, *UnDo/ReDo* entsprechen.

Wird in einer Dialoghistorie ein *Travel–Undo* zugelassen, so kann durch nachfolgende Aktionen und Zustandsänderungen, z. B. *Again*[4], eine Verzweigung der Historie entstehen. Die lineare Form der Dialoghistorie erweitert sich dann zu einer baumartigen Struktur. Die Handhabung dieser Struktur erfordert eine genaue Formalisierung der Abläufe innerhalb des Baumes. Im Anhang A.3 sind die benötigten Hilfsmittel definiert.

Ansatz nach Archer, Conway, Schneider und Vitter

Bei der konzeptuellen Sicht der Interaktion wird jedes Kommando des Benutzers als abgeschlossen betrachtet. Dabei wird ein Kommando zunächst vorbereitet, dann ausgeführt. Es scheint daher sinnvoll zu sein, die Folge der Kommandos bearbeiten zu können, bevor sie ausgeführt wird. Verschiedene Aktionsfolgen können denselben Zustand erreichen. Eine Unterscheidung von beteiligten Objekten wird von Thimbleby getroffen [Thi90]:

- Historie: Eine vollständige Aufnahme der Benutzeraktionen
- Script: Eine Aufnahme der Interaktion, die das System als effektive Historie benötigt (behandelt)
- State: die Äquivalenzklasse von Script

[4] *Again* ist die wiederholte Ausführung von Aktionen oder Aktionssequenzen. Die einfachste Form ist die Wiederholung der letzten Aktion, z. B. mehrfaches Suchen einer Zeichenkette in einem Text.

Nach Archer, Conway und Schneider konstruiert der Benutzer ein *script* und das System interpretiert dieses. Das Benutzerinterface schaltet zwischen zwei Aktivitäten:

- *edit activity:* Der Benutzer bearbeitet das Script. Jede *edit*–Aktivität wird durch ein spezielles Signal beendet, z. B. `enter`–Kommando.
- *execute activity:* Das System interpretiert das Script. Die Phase wird durch Übergabe der Steuerung an den Benutzer beendet.

Zu einem interaktiven System wird ein Script interaktiv erstellt, im Batch–Betrieb bevor die Ausführung gestartet wird. Wenn hier *UnDo* betrachtet wird, besteht die Möglichkeit einer weitergehenden Bearbeitung der Scripte. Die Abarbeitung eines Scriptes kann in zwei Teile zerfallen. Eine Sequenz von Aktionen wurde vorbereitet. Der erste Teil dieser Folge wurde terminiert, d. h. abgearbeitet. Der zweite Teil wird vor der weiteren Arbeit noch einmal vom Benutzer überarbeitet. Archer, Conway und Schneider unterscheiden zwischen ausgeführten Befehlen A und wartenden Befehlen W[5]. Dann ist ein Script durch $S = A \circ W$ repräsentiert. Wir nehmen nun an, daß der Benutzer die wartenden Befehle modifiziert. Mit einem Apostroph werden alle Befehle nach Ausführung und Bearbeitung gekennzeichnet. Das Script ist dann $S = A \circ W' = U' \circ V'$. U steht für unveränderte, V für veränderte Kommandos. Die Ausführung eines Scripts ist

- vollständig, wenn $W = \emptyset$ und $A = S$,
- partiell, wenn $W \neq \emptyset$ oder $A \neq S$,
- nicht *UnDo*–fähig, wenn der Benutzer ausgeführte Kommandos nicht erreichen kann, d. h., $U' = S$ und M' ist Erweiterung von S.
- Ein Teil des Scripts ist akzeptiert, wenn der Benutzer es nicht mehr ändern kann.
- Ein Script kann nur erweitert werden, wenn $U' = S$ ist.
- Wenn AE kein Präfix von $U' \circ M'$ ist, dann ist das System in einem inkonsistenten Zustand. Der Benutzer möchte Kommandos ändern, die das System bereits ausgeführt hat.

Vitter ergänzt die Betrachtungen um die Möglichkeit, Alternativen darzustellen. Das Resultat ist nach wie vor ein Zustand, der aus einer linearen Folge von Aktionen zusammengesetzt ist.

command insertion: Ein Kommando kann an einer beliebigen Stelle im Script eingefügt werden.

[5]Die ursprünglichen Zeichen E und P wurden nicht verwendet, da deren Gebrauch mit späteren Definitionen kollidiert.

change of heart: Eine Alternative kann nach vorhergehendem *UnDo* der bisherigen Aktionsfolge erneut ausgeführt werden.

command substitution: Ein Kommando oder eine Kommandofolge kann durch ein anderes Kommando oder eine Kommandofolge ersetzt werden.

selective undos: Ausgewählte Kommandos können zurückgenommen werden.

command rearrangement: Die Reihenfolge der Kommandos in einer Kommandosequenz kann verändert werden.

selective rearrangement: Einzelne, ausgewählte Kommandos können umgeordnet werden.

Ansatz von Yang

Die Voraussetzungen von Archer, Conway, Schneider und Vitter sind Grundlage der Betrachtungen von Yang [Yan88a]. Dabei verzichtet sie auf eine lineare oder baumartige Struktur des Scripts. Die durchgeführten Aktionen befinden sich nach wie vor in einem linearen Script und repräsentieren den Zustand des Systems. In einer Ringliste werden die zurückgenommenen Aktionen abgelegt. Durch Manipulationen auf der Ringliste kann jede beliebige in ihr enthaltene Aktion wieder an beliebiger Stelle in das Script übernommen werden. Die Metaaktionen von Vitter werden in diesem Modell auf ihre Gültigkeit hin überprüft.

Weitere Überlegungen

Systemzustände in einer Benutzerhistorie ebenso wie Funktionsaufrufe des Benutzers unterliegen einer zeitlichen Abfolge. Jedes Dialogereignis (eine Aktion oder eine Zustandsänderung) wird durch eine ordnende Relation *(zeitlich nach)* in Beziehung zu anderen Dialogereignissen gesetzt. Mit einer Ordnung der zeitlichen Abfolge der Zustandsänderungen erhalten wir einen geordneten Baum[6].

Die eigentliche Baumstruktur kommt nach Durchführung von *UnDo/ReDo* und *Again* zustande, wie auch von Herczeg deutlich gemacht wurde [Her86a]. Es ist auch denkbar, z. B. um alternative Pläne zu verfolgen, einen beliebigen Zustand in der

[6]Ein Baum enthält eine Wurzel, die hier immer als oberstes Element dargestellt wird. Die Wurzel des geordneten Ereignisbaumes wird gemäß Definition 20 bestimmt. (Siehe Anhang A.3.) Bei Ereignisabfolgen, wie z. B. Dialogereignissen, kann ein Ursprungs– oder Ausgangsereignis angenommen werden. Wenn als ordnende Relation R_z die zeitliche Abfolge der Ereignisse betrachtet wird, die Elemente der Menge M mit den Dialogereignissen identifiziert werden und ein Ursprungsereignis betrachtet wird, erhalten wir eine externe Ordnung der Dialoghistorie. Die Ordnung R_z impliziert im allgemeinen *nicht* die Vorgänger/Nachfolger–Relation der Zustände, wie sie in einem Baum der Dialoghistorie gebraucht wird. Die Ordnung R_z bestimmt nur, daß zwei Ereignisse zusätzlich in einer festgesetzten Reihenfolge auftreten. Dadurch wird insbesondere impliziert, daß die wiederholte Ausführung erneut in den Baum eingetragen werden muß, da sonst Widersprüche bezüglich der zeitlichen Reihenfolge auftreten.

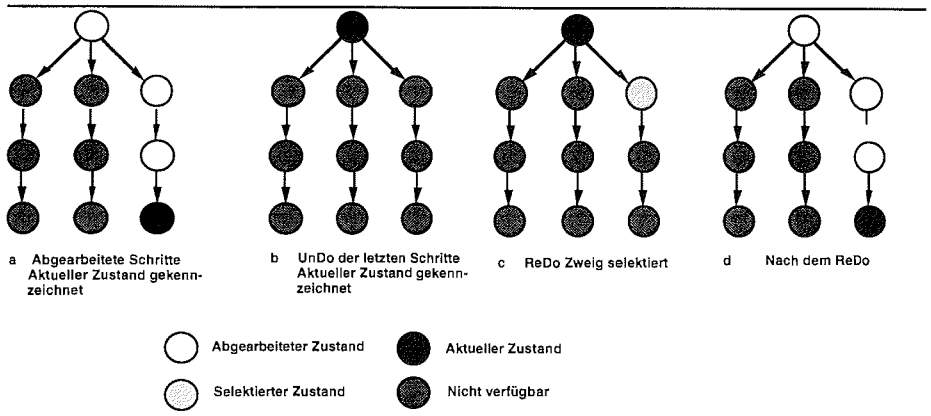


Abbildung 4.5: Eine *UnDo/ReDo*–Sequenz wird ausgeführt.

Dialoghistorie auszuwählen und mit einer von diesem Zustand noch nicht getätigten Zustandsänderung fortzufahren.

In Abbildung 4.5 ist ein Szenario einer baumartigen Dialoghistorie abgebildet. Innerhalb dieses Szenarios ist interessant, was tatsächlich bei einem *UnDo* geschieht, wie der Benutzer bei einem *ReDo* in den jeweiligen Zweig kommt. Hier ist es besonders wichtig herauszustellen, was mit Zweigen geschieht, die von einem Knoten ausgehen, der von einem *UnDo* bzw. *ReDo* berührt wird.

Bei der Betrachtung des *UnDo/ReDo*–Szenarios wird davon ausgegangen, daß für den Benutzer ein springendes *UnDo* möglich ist, d. h., der Benutzer ist nicht gezwungen, ein *UnDo* für jeden Zustand oder jede Funktion durchzuführen, sondern kann einen beliebigen Zustand auswählen, bis zu dem ein *UnDo* durchgeführt werden soll. Es ist zu überlegen, ob ein *ReDo* auf die gleiche Weise ausgeführt werden soll, d. h., der Benutzer selektiert einen bereits verlassenen Zustand und setzt ein *ReDo* bis zu diesem Zustand in Gang, gegebenenfalls mit nachfolgendem aktivierten *ReDo* von dem selektierten Zustand aus. Dies bedeutet, daß Zweige wiederholt mit gleichen Parametern durchlaufen werden.

Ein besonderes Augenmerk ist auf die Knoten zu richten, an denen durch bereits ausgeführte *ReDo*– und *Again*–Aktionen mehrere Aktionszweige entstanden sind. Ein *Again* verändert den Zustand des Systems. Daher wird beim *Again* ein neuer Zweig eingefügt. Beim *ReDo* muß der Benutzer einen Zweig für das *ReDo* auswählen, wenn nicht dieser Verzweigungsknoten der letzte Knoten der *ReDo*–Sequenz ist. Durch das freie Navigieren innerhalb des Baumes treten Probleme auf, die bei der Behandlung linearer Dialoghistorien nicht berücksichtigt werden mußten.

Die folgenden Punkte kennzeichnen diesen Problemkreis:

- *ReDo* einer Aktionssequenz,
- *Again* einer Aktionssequenz,
- inverse Funktionen nur innerhalb eines linearen Teilbaumes,
- *UnDo* bei Zweigen, die durch *UnDo/ReDo*–Sequenzen entstanden sind,
- *UnDo* bei Zweigen, die durch *Again* entstanden sind,
- System muß prüfen, ob die Zustände bereits aktiviert sind,
- gegebenenfalls müssen sie neu aktiviert werden.

Die in einem Historiebaum notwendigen Funktionen können im Rückgriff auf die Formalisierung von Herczeg definiert und beschrieben werden [Her86a]. Bevor die Funktionen beschrieben werden, sind noch einige Definitionen notwendig. (Siehe Anhang A.3.)

Definition 2 (Selektiertes, aktives und aktuelles Element)
Ein Element im Baum heißt selektiert, wenn es der zuletzt angefügte Knoten ist, falls einer seiner Nachfolger gelöscht wurde oder falls es aktiv vom Benutzer ausgewählt wurde.

Ein Element im Baum heißt aktiv, wenn er durch ein UnDo/ReDo nicht zurückgenommen wurde.

Ein Element wird aktuell, wenn durch eine geeignete Operation des Benutzers das System auf den Zustand, den das Element beschreibt, gesetzt wird.

D. h., ein selektiertes Element kann aktuelles Element werden, ein aktuelles Element ist gleichzeitig auch selektiert, sofern kein anderes Element selektiert wird. Doch nun zurück zu den notwendigen Funktionen:

- Selektieren eines Elements
- Aktivieren eines Elements
- Einfügen eines Elements
- *UnDo* eines Elements oder Bereichs
- *ReDo* eines Elements oder Bereichs
- *Again*
- Makro–Generierung

Selektieren: Die Selektion eines Elements erfolgt durch Eingabe einer Identifikation des Elements. Die Funktion entspricht einer Suchfunktion auf dem Baum[7].

$$select(Identifikation, Baum) := \begin{cases} k \epsilon \mathcal{K} & \text{falls } Identifikation \text{ vorhanden} \\ \kappa & \text{falls } Identifikation \text{ nicht vorhanden} \end{cases}$$

Aktualisieren: Das Aktualisieren eines Elements besteht aus einem *UnDo* des bisherigen aktuellen Bereichs und nachfolgenden *ReDo* des neuen zu aktualisierenden Zweiges (Notation nach Herczeg [Her86a][8]).

$$\begin{aligned} act(z_{neu}) \quad := \quad & redo(sup(pf(r, z_{alt}) \cap pf(r, z_{neu})), z_{neu}) \ \circ \\ & un(sup(pf(r, z_{alt}), pf(r, z_{neu}))) \end{aligned}$$

UnDo **in Verzweigungen:** Eine *UnDo*–Funktion in einer Baumstruktur muß auch gegebenenfalls vorhandene Verzweigungen des Baumes beachten. Zunächst existiert nach Def. 22 ein Pfad $pfad(r, z_{sel})$ von der Wurzel zum selektierten Element als Kette. Für Verzweigungen gilt, entweder ist der Zweig aktiv, oder es wurde bereits ein *UnDo* darauf ausgeführt, d. h., die *UnDo*–Funktion muß den *aktiven* Zweig bearbeiten. Daraus kann abgeleitet werden, daß die einfachen *UnDo*–Funktionen von Herczeg Anwendung finden.

Zusammenfassung

Nach den vorhergehenden Ausführungen sind mehrere Verfahren möglich, mit denen *UnDo/ReDo* erfaßt werden kann. Vorherrschend ist eine lineare Abfolge von Aktionen, deren Resultat jeweils der Systemzustand ist. Die zurückgenommen Aktionen werden gekennzeichnet. In der vorliegenden Arbeit werden wir in beiden Fällen einen Baum verwenden, zwischen denen Aktionen durch Metaoperationen ausgetauscht werden.

Im weiteren wird eine Dialoghistorie benutzt, um aus getätigten Aktionen eine Folge zu selektieren, die dann, den Bedürfnissen des Benutzers entsprechend, angepaßt werden kann. Jede Umordnung oder Veränderung der Historie widerspricht dem Aspekt der Dokumentation von Aktionen. Jede Aktionssequenz des Benutzers ist ein Experiment, das zu einem gewünschten Ergebnis führen soll. Alternativen sollen Fehler deutlich machen und Veränderungen darstellen. Dabei helfen Vergleiche zwischen einzelnen Alternativen, die richtige zu entwickeln. Daher wird in den folgenden Ausführungen auf Umordnungen und Ersetzungen im Sinne von Vitter verzichtet.

[7] *Identifikation* identifiziert das Element, *Baum* ist der Baum der Historie, $\mathcal{K}$ ist die dem Baum zugrundeliegende Knotenmenge. Das Zeichen κ repräsentiert ein nicht definiertes Element.

[8] Definitionen nach Herczeg

z	Zustand	$pf(r, z)$	Pfad von der Wurzel r zum Zustand z
r	Wurzel der Historie	$sup(pf_1)$	zuletzt erreichter Zustand im Pfad pf
z_{alt}	vorheriger Zustand	$redo(z_1, z_2)$	Wiederholung der Zustände von z_1 bis z_2
z_{neu}	der neue Zustand	$un(z_1, z_2)$	Rücknahme der Zustände von z_1 bis z_2

Regel 1: Kein Roboter darf einem Menschen Schaden zufügen oder durch Untätigkeit zulassen, daß einem Menschen Schaden zugefügt wird.

Regel 2: Ein Roboter muß dem ihm von einem Menschen gegebenen Befehl gehorchen, es sei denn, ein solcher Befehl würde nit Regel Eins kollidieren.

Regel 3: Ein Roboter muß seine Existenz beschützen, solange dieser Schutz nicht mit Regel Eins oder Zwei kollidiert.

Isaac Asimov

Kapitel 5

Dialoggeschichte und Handlungspläne

5.1 Das Datenmodell für die Dialoghistorie

Aus der vorhergehenden Formalisierung ist ein abstrakter Datentyp ADT eines Elements des *UnDo/ReDo*–Mechanismus abzuleiten. Eine Übersicht über anwendbare Datenmodelle ist bei Brodie zu finden [Bro86]. Shaw stellt eine Übersicht über die Realisierung von Konzepten in Programmiersprachen dar [Sha86]. Da ein abstrakter Datentyp in seiner Definition sowohl notwendige Daten als auch Operatoren enthält, sind die Voraussetzungen für eine spätere objektorientierte Realisierung gegeben. Mit einem abstrakten Datentyp werden alle Anforderungen einer objektorientierten Umgebung wie Verstecken der Daten *(data hiding)*, Bereitstellen von Nachrichten und Vererbung erfüllt. Ein besseres Wort für *Abstrakter Datentyp* in diesem Zusammenhang ist *Objekt*. Dieser Begriff soll im weiteren Verwendung finden, damit keine Verwechselung mit der Definition des abstrakten Datentyps *HistorieBaum*[1]

[1]Treten im Text zusammengesetzte Worte auf wie z.B. *HistorieBaum*, so sind dies bereits Hinweise auf Klassennamen, mit denen die beschriebenen Objekte im Programm realisiert werden. Zeichenerklärungen sind im Anhang C zu finden.

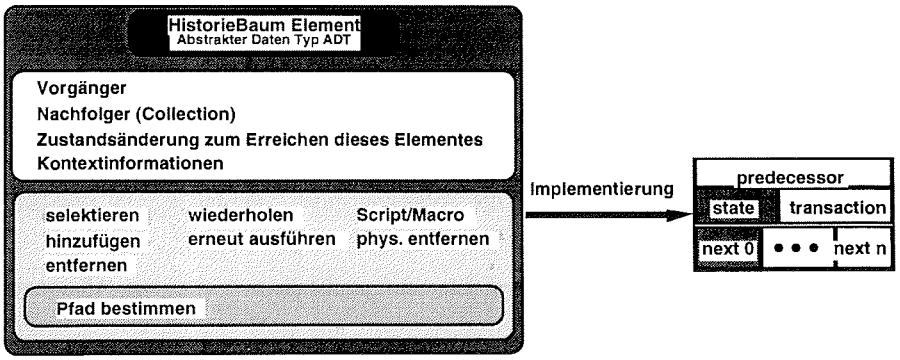


Abbildung 5.1: Die Abbildung zeigt die Datenstruktur eines *HistorieBaumElementes* und die Implementierung.

stattfindet. Die Konstruktion eines abstrakten Datenmodells bei der Entwicklung eines objektorientierten Datenbanksystems wird von Banerjee et al. beschrieben [BCG+87]. Eine Diskussion konzeptioneller Ansätze zur Datenmodellierung finden sich bei Brodie [BMH85]. Die Beschreibung und Anwendung der konzeptionellen Datenmodellierung ist von Vetter dargestellt [Vet90].

Im Anhang A.3 wird eine Datenstruktur der Dialoghistorie mathematisch formuliert. Der mathematische Ansatz der Formalisierung soll helfen, die Struktur besser handhabbar zu machen und gegebenenfalls die Implementierung zu überprüfen.

In Abbildung 5.1 ist das Objekt *HistorieBaumElement (HBE)* gezeigt. Die notwendigen Daten des Elementes werden wie folgt gespeichert:

[**Vorgänger**] Jedes Element im Historiebaum hat einen Vorgänger. Ein Verweis auf diesen wird benötigt, um im Baum die Suche nach einem Element nicht immer bei der Wurzel beginnen zu müssen *(doppelt verkettete Liste)*.

[**Nachfolger**] Jedes Element kann mehrere Nachfolger besitzen. Bei der weiteren Betrachtung wird die Repräsentation durch einen binären Baum vorgezogen. Die Erzeugung und Verwendung von Nachfolgern ist eine direkte Ableitung aus den Aktionen *Again* und *ReDo*.

[**Zustandsänderung**] Beim Erzeugen des Elementes wird der Systemzustand in signifikanter Weise verändert. Diese Änderungen werden hier abgelegt, damit bei einem folgenden *UnDo/ReDo* der Systemzustand durch Anwendung einer inversen Operation wiederhergestellt werden kann.

[**Kontext**] Jedes Element wird in einem Systemkontext eingesetzt. Dieser Kontext ist durch einen Zustand beschreibbar.

Ein Objekt stellt eine Reihe von Operatoren (Methoden, Funktionen und Prozeduren) zur Verfügung, mit denen auf seine Daten zugegriffen werden kann bzw. über die Aktionen des Objekts aktiviert werden. Die Operatoren des *HistorieBaumElements* leisten folgendes (Vergleiche Abbildung 5.1):

[Selektieren] Ein Element im Baum wird vom Benutzer selektiert, um eine Aktion damit durchzuführen. Die Selektion eines Elementes erzwingt die Ausführung eines privaten Operators. Auch die Selektion eines Teilpfades muß möglich sein.

[Pfad bestimmen] Der Pfad zu einem Element muß bekannt sein, um Aktionen ausführen zu können.

[Hinzufügen] Ein Element muß zum Baum hinzugefügt werden, z. B. der nächste Dialogschritt.

[Entfernen] Ein Element wird als *gelöscht* markiert. Dies entspricht einem *UnDo*.

[Physikalisches Entfernen] Nicht mehr gebrauchte Elemente werden aus dem Baum entfernt.

[Wiederholen] Die Aktionen eines selektierten Elements werden wiederholt, gegebenenfalls mit neuen Parametern. Dies entspricht einem *Again*. Die Aktion wird als neue Änderung betrachtet.

[Erneut ausführen] Selektierte Aktionen werden wiederholt. Dies entspricht einem *ReDo*. Die Aktion wird als Wiederherstellung eines bekannten Zustandes betrachtet.

[script/macro] Eine Sequenz von *HistorieBaumElementen*, die eine Handlungsfolge repräsentieren, werden zu einem Makro zusammengefaßt. Dieser Makro enthält Parameter, die bei der Ausführung des Makros vom Benutzer anzugeben sind bzw. aus dem Kontext belegt werden.

Neben den Elementen einer Dialoghistorie, die die Knoten eines Dialogbaumes repräsentieren, wird noch eine Datenstruktur benötigt, die als Ganzes vom Benutzer angesprochen wird. Diese Datenstruktur ist der *DialogHistorieBaum*. Die Einführung dieses Objekts dient der leichteren Strukturierung des gesamten Baumes in Teilbäume, der besseren Zugriffskontrolle und dem Kapseln der Information (*data hiding*). Mit dem *DialogHistorieBaum* wird erreicht, daß sich im System zwar identische Zustände befinden können, diese aber nur referenziert werden müssen. Dies betrifft z. B. vordefinierte Symbole eines Betriebssystems, festgesetzte Aufrufe für Compiler und Linker, aber auch komplette Systemzustände. Der Kontext des Zustands befindet sich im *DialogHistorieBaum* oder im *HistorieBaumElement*. Im *DialogHistorieBaum* werden globale Daten

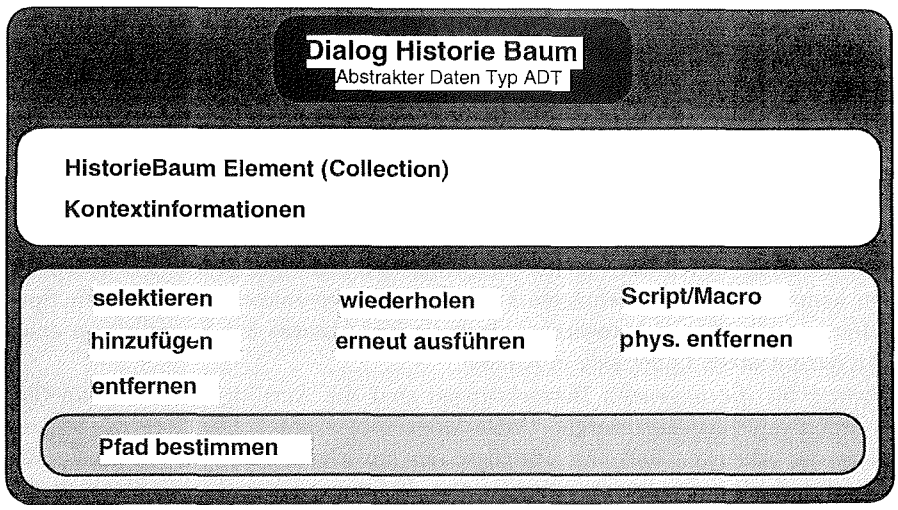


Abbildung 5.2: Die Abbildung zeigt die Struktur des *DialogHistorieBaumes* mit Attributen und Methoden.

gespeichert, im *HistorieBaumElement* lokale. Eine algebraische Definition des *DialogHistorieBaumes* als abstrakter Datentyp folgt im nächsten Abschnitt.

Ein *DialogHistorieBaum* ist eine Struktur, die es ermöglicht, Methoden zur Navigation und Manipulation bereitzustellen, ohne auf den Inhalt des gezeigten Elements zugreifen zu müssen. Der *DialogHistorieBaum* referenziert einzig und allein die Informationen über Vorgänger und Nachfolger beim Navigieren im *DialogHistorieBaum*. Erst bei der Manipulation — dies umfaßt insbesondere das *UnDo* — muß auf den Inhalt des *HistorieBaumElementes* zugegriffen werden. Die Struktur des *DialogHistorieBaumes* vereinfacht daher den Umgang mit der Dialoghistorie. In Abbildung 5.2 ist das Objekt *DialogHistorieBaum* abgebildet.

Bei den nachfolgenden Betrachtungen der Aktionsfolgen eines Benutzers wird *nicht* von parallel ausgeführten Prozessen ausgegangen. Wenn parallele Aktionen möglich sind, wird ausdrücklich versucht, diese zu sequentialisieren. Dadurch werden viele Arten mit dem Rechner zu arbeiten a priori untersagt. Wir treffen hier die Voraussetzung, daß der Benutzer abgeschlossene Aktionen besitzt, auf denen er seine nächsten Aktionen aufbaut. Eine abgeschlossene Aktion in diesem Zusammenhang werden wir dann als Aktion definieren, deren Änderungen zum Zeitpunkt des Absetzens bekannt sind, die Aktion kann jedoch quasiparallel ausgeführt werden.

5.2 Abstrakte Datentypen und der *HistorieBaum*

Im folgenden skizzieren wir einen abstrakten Datentyp *(ADT)*, der eine Dialoghistorie repräsentiert. Die Definitionen können so weit abstrahiert werden, daß die Ansätze von Archer, Conway und Schneider, Vitter und Herczeg mit eingeschlossen werden. Am Ende der Ausführungen werden diese Korrespondenzen gezeigt. Insbesondere werden Voraussetzungen erarbeitet, unter denen die von Vitter geforderten Metaaktionen gültig sind oder nicht.

Die algebraische Spezifikation einer Datenstruktur oder eines Systems ist für die Implementierung vorteilhaft, da mit mathematisch-logischen Methoden überprüft werden kann, ob eine neu definierte Funktion oder Operation konsistent zur Spezifikation ist. Ein Vergleich mehrerer algebraischer Ansätze zur Spezifikation einer Benutzerschnittstelle findet sich in [Chi85].

Wichtig für die folgenden Betrachtungen sind die Ausführungen in [Yan88b]. Hier werden verschiedene generelle *UnDo*-Modelle betrachtet. Das Ergebnis ist eine Ringliste der Aktionen und eine Ringliste der zurückgenommenen Aktionen. Resultierend verbleibt eine lineare Struktur der Aktionen. Diese lineare Struktur der Aktionsfolge wird in den folgenden Ausführungen zugunsten einer Baumstruktur aufgegeben. Die Folge der Zustände des Systems *(states)* bleibt linear. Hier ist die Abhängigkeit einer Aktion vom Zustand des Systems zu klären.

Die Motivation für die Baumstruktur ergibt sich aus dem Wunsch, den Anwender bei der Arbeit mit der Dialoghistorie zu unterstützen. Eine Baumstruktur liefert dem Benutzer Informationen über den Kontext einer Aktion oder einer Sequenz von Aktionen. Sowohl bei ACS wie auch bei Vitter und Chan sieht der Benutzer nur die fehlerfreie Sequenz, kann aber auf Alternativen nicht mehr (ACS, Chan) oder nur schwer (Vitter) zurückgreifen. Mit einem Baum sind auch erfolgreiche, aber wieder verworfene Alternativen weiter verfügbar.

Zur Identifikation der Elemente *(Knoten)* in der Historie werden Identifikatoren eingeführt, aus denen sofort die Position des Knotens im Baum ablesbar ist. Das Anfügen und Löschen eines Knotens wird definiert. Diese Punkte sind zunächst unabhängig von den Daten, die ein Knoten beschreibt, definierbar.

Wird jedoch später der Knoten im Kontext der veränderten Daten oder des veränderten Gesamtzustands des Systems betrachtet, ergeben sich zusätzliche Bedingungen für die Ausführbarkeit der einzelnen Operationen. Diese Bedingungen werden dargestellt.

Als erstes werden die Grundstrukturen und Funktionen erklärt, dann wird unter Berücksichtigung der Semantik der Ansatz verifiziert. Abschließend wird die Historie bei der Anwendung der *UnDo/ReDo*-Funktionen betrachtet.

Bisherige Betrachtungen ließen außer acht, daß die Benutzer nicht mehr nur in einer Applikation arbeiten und diese bis zum Abschluß ihrer Aufgabe nicht mehr verlassen. Bedingt durch die neuen Fensteroberflächen kann zwischen Applikationen gewechselt werden, und Aufgaben können parallel ausgeführt werden. Wird diese Parallelität von Aktionen zugelassen, dann kann die Synchronisation dieser

Aktivitäten nicht mehr ohne Aufwand in einer Dialoghistorie repräsentiert werden. Daher wird eine Voraussetzung getroffen, die ermöglicht, parallele Aktionen in beschränktem Umfang zuzulassen. (Siehe Seite 93ff.)

Die im weiteren definierte Dialoghistorie wird Zustandsänderungen speichern. Daraus sind lokale Änderungen des Gesamtzustands ableitbar. Der Gesamtzustand ergibt sich in der Regel *nicht* durch Betrachtung der Änderungen entlang eines Pfades, sondern auch die aktiven Änderungen in parallelen Pfaden tragen zum Gesamtzustand bei.

5.2.1 Die Darstellung der Dialoghistorie

Eine Dialoghistorie wird in der vorliegenden Arbeit als binärer Baum betrachtet. Ein Knoten im Baum repräsentiert dabei eine Zustandsänderung, z. B. ein Kommando. Bedingt durch die Operationen, die auf der Dialoghistorie erlaubt werden, kann ein Knoten mehrere Nachfolger besitzen. Daher werden sogenannte *dummy*–Knoten eingeführt, durch die der Baum eine binäre Struktur erhält. In den folgenden Ausführungen werden einige Begriffe kurz definiert. Es werden wenige formale Darstellungen gezeigt. Diese sind insgesamt im Anhang A.3 zu finden.

Eigenschaften im Baum

Eine Systemzustandsänderung wird durch folgende Attribute innerhalb einer Dialoghistorie gekennzeichnet:

- Art der Änderung
- Zeitpunkt der Änderung
- Zuordnung innerhalb der Historie

Die Zuordnung innerhalb der Historie wird durch die Position, die die Systemzustandsänderung innerhalb des gebildeten Baumes hat, erreicht. Dazu wird ein Knoten definiert, der die Änderung mit der Zeitinformation und die Position enthält. Ein Knoten ist dann durch die Position bestimmt. Durch Betrachtung der Positionen verschiedener Knoten lassen sich Rückschlüsse auf die Abhängigkeiten zwischen den Knoten gewinnen. Die Position ist eine Zeichenkette über $\mathcal{M}'$[2].

Im folgenden bezeichnet *Baum* einen binären Baum, wenn nichts anderes vermerkt wird, *Knoten* eine Struktur, die als Knoten in einem Baum benutzt wird, *Position* eine Identifikation eines Knotens in einem Baum, und *Datum* eine allgemeine Datenstruktur. Die hier zugrundeliegende Relation ist die Inklusion der Zeichenketten „$\sqsubset$“.

Die Vorgänger und Nachfolger eines Knotens k werden mit $pred(\mathcal{K}, k)$ bzw. $succ(\mathcal{K}, k)$ bezeichnet, speziell bei binären Bäumen die *linken* bzw. *rechten* Nachfolger mit $lsucc(\mathcal{K}, k)$ bzw. $rsucc(\mathcal{K}, k)$. Die Mengen der linken, rechten Nachfolger, Vorgänger und Nachfolger werden mit $\mathcal{L}(t, k)$, $\mathcal{R}(t, k)$, $\mathcal{V}(t, k)$ und $\mathcal{S}(t, k)$ bezeichnet.

[2]Siehe Anhang A.3.

Es existieren Operationen, die einen Baum verändern, d. h. die Menge der Knoten vergrößern oder verkleinern und die Relation neu festlegen. Daher muß bei den meisten Operationen *der* Baum als Argument mit angegeben werden, auf dem die Operation ausgeführt wird.

Für weitere Betrachtungen ist die Vereinigung von Bäumen relevant. Die Verbindung von Bäumen wird benötigt, wenn Knoten, die Zustandsänderungen enthalten, aus dem Baum der zurückgenommenen Aktionen in den Baum der Dialoghistorie eingebunden werden. Die Vereinigung läßt sich als Mengenvereinigung der zugrundeliegenden Knotenmengen unter Beibehaltung der Relation darstellen[3]. Das Entfernen eines Teilbaumes erfolgt über die Bildung der Mengendifferenz.

Zum Aufbau des binären *HistorieBaum* wird ein spezieller Knoten eingeführt, der durch eine Duplikat–Funktion generiert wird. Das Duplizieren eines Knotens ist eine Hilfsfunktion. Mit dieser Funktion wird ein Baum erzeugt, der einen einzelnen Knoten enthält, der später das gleiche Datum erhält, wie der, dessen Nachfolger er wird. Dieser Knoten besitzt daher weder Vorgänger noch Nachfolger.

Der Algorithmus, der das Anfügen eines Baumes an einen Zielbaum beschreibt, wird durch Betrachten einzelner Teilfunktionen wesentlich vereinfacht. Zunächst wird das Anfügen eines Knotens definiert *(append)*. Diese Operation prüft nach, ob der Zielbaum leer ist. Der Zielbaum wird dann gleich dem Quellbaum gesetzt. Ist der Zielbaum nicht leer, wird das Anfügen des Baumes als linksseitiger Teilbaum versucht. Falls kein linker Nachfolger des ausgewählten Knotens existiert, ist die Operation erfolgreich, sonst wird versucht, den Baum als rechtsseitigen Teilbaum anzufügen. Gibt es zu dem gewählten Knoten keinen rechten Nachfolger, so wird ein *Duplikat* des Knotens rechtsseitig angefügt und der Baum als linksseitiger Teilbaum an dieses Duplikat angefügt. Ist diese Operation nicht erfolgreich, d.h., es existiert ein rechter Nachfolger, so wird versucht, den Baum linksseitig an diesen Nachfolger anzufügen *(lappend)*. Die Operation *lappend* ist rekursiv definiert. Da die Anzahl der Knoten eines Baumes endlich ist, wird der Algorithmus terminieren.

Der direkte rechte Nachfolger eines Knotens ist nach dieser Konstruktion *immer* das Duplikat des Knotens. Die Einführung des Duplikates erleichtert die Bestimmung der Position eines Knotens und die Bestimmung der Vorgänger–/Nachfolger-Beziehung. Die Aufeinanderfolge von Aktionen bleibt ebenfalls sichtbar. Die Eigenschaften eines *binären Baumes* können ausgenutzt werden. Duplikate werden im Baum besonders gekennzeichnet. (Siehe auch Abbildung A.3.1.) Durch diesen Algorithmus wird auch sichergestellt, daß die Position signifikanter Knoten, d. h. Knoten, die Zustände beschreiben, immer auf „0“ endet.

Die Daten und der Zugriff

Die Knoten in der Dialoghistorie beschreiben einzelne Schritte des Benutzers, der Baum die Gesamtheit alle Dialogschritte innerhalb einer Sitzung. Bedingt durch die besondere Art der in den Knoten abgelegten Daten ergeben sich spezielle Funktionen

[3]Einzelheiten siehe Anhang A.3.

auf den Knoten und auf dem Baum.

Ein Zustand ist durch den aktuellen Status der vom Benutzer erreichbaren Rechnerkonfiguration bestimmt, d. h. (deskriptiv, vergleiche auch Definition 5):

- Zustand der Daten auf den erreichbaren Rechnern: $\mathcal{D}'$ *(Daten)*
- `login`–Status: $\mathcal{S}$ *(Status)*
- falls vorhanden: geöffnete Fenster auf Daten, Rechnern und Directories: $\mathcal{W}$ *(Windows)*
- Symboldefinitionen und Umgebungsvariablen: $\mathcal{E}$ *(Environment)*
- Optionen von Befehlen $\mathcal{O}'_C$

Im folgenden fassen wir die Mengen $\mathcal{D}'$, $\mathcal{O}'_C$ und $\mathcal{E}$ zu der Menge der Daten des Systemzustands zusammen: $\mathcal{D} := \mathcal{D}' \times \mathcal{O}'_C \times \mathcal{E}$.

Gemäß den Ausführungen in Kapitel 4.3.2 sind die Änderungsfunktionen des Systemzustands zu protokollieren. Eine Änderungsfunktion, z. B. ein Betriebssystembefehl, wird durch die Daten, auf denen sie operiert, und durch Optionen bestimmt. Es bleibt zu diskutieren, ob die Befehle (VMS–Befehle)

```
copy a b
copy/log a b
copy/log=logfile a b
```

durch Angabe der Optionen unterschiedliche Befehle darstellen oder ob die Optionen die Auswirkungen eines Befehls näher präzisieren. Der letzte Befehl ändert auf jeden Fall den Datenbestand, da immer eine Protokolldatei `logfile` angelegt wird. In der vorliegenden Arbeit werden als Befehle die Grundänderungen betrachtet, so daß im Beispiel der `copy`–Befehl als Änderungsfunktion und die Dateien `a` und `b` als Daten sowie die Option `/log` oder `/log=logfile` als Bestimmungsteil des Befehls in der Menge $\mathcal{D}$ zusammengefaßt werden. In jedem Fall können für die `copy`–Befehle des Beispiels die Eingabeparameter und die Ausgabeparameter bestimmt werden.

Definition 3 (Zeitpunkt) *Ein Zeitpunkt wird durch das zugrundeliegende Rechnersystem bestimmt. Ein Zeitpunkt ist der Abstand eines Ereignisses zu einem Startpunkt, gemessen in ganzzahligen Vielfachen des Systemtaktes bzw. einer daraus abgeleiteten Größe. Daraus ergibt sich eine Folge von Zeitpunkten. Die Zeitpunkte werden durch die Menge $\mathcal{Z}$ beschrieben.*

Definition 4 (Systemzustand)
Ein Systemzustand ist ein Tupel $x = (d, s, w, z)$ mit $d \in \mathcal{D}$, $s \in \mathcal{S}$, $w \in \mathcal{W}$, $z \in \mathcal{Z}$.

Nach jeder ausgeführten Aktion ist im Hinblick auf diesen Zustand ein Ausschnitt aus den durch die Werte d, s, w und z beschriebenen Daten modifiziert. Im Falle disjunkter Teilbereiche ist die Vereinigung aller Modifikationen zum Beobachtungszeitpunkt eine Beschreibung des Gesamtzustands.

Mit jeder Aktion verändert der Benutzer den Zustand seiner Arbeitsumgebung auf dem Rechner. Damit erreichte Zustände unterscheidbar bleiben, wird folgende Voraussetzung getroffen:

Grundannahme 1 *Im Gesamtsystem existiert ein einziger Zeittakt, der die Folge von Zeitpunkten generiert.*

Dieser Zeittakt kann durch Bezug auf die Systemuhr des lokalen Rechners, auf dem der Benutzer seine Arbeit ausführt, hergestellt werden[4].

Hier stößt man unweigerlich auf die Problematik, die auch bei der Entwicklung von Betriebssystemen [Ric84, Neh87] und Datenbanken bekannt ist: Ermöglichen und Synchronisieren von zeitgleich auftretenden Prozessen. Nach Nehmer ist eine vollständige Beschreibung eines aktuellen Gesamtzustands in aller Allgemeinheit nicht möglich [Neh87, S. 55]. Wir beschränken uns daher auf einige signifikante Aspekte, für die damit eine Beschreibung des Gesamtzustands erreicht wird.

Die Veränderung des Systemzustands, d. h. der Daten, des Status, der Oberfläche und der Umgebung, ist eine erwünschte Folge einer Benutzeraktion. Die Änderungen δ von einem Zustand x auf einen Zustand y werden durch Übergangsfunktionen beschrieben. Aktionen können sich beeinflussen, wenn sie den gleichen Datenbestand benutzen. Diese Abhängigkeiten in einer Folge von Aktionen sind nun zu diskutieren. Wir betrachten unterschiedliche Typen von Aktionsfolgen. (Siehe Tabelle 5.1.)

Wegen der in der Regel fehlenden globalen Sicht auf einen konsistenten Zustand muß der Typ 4 aus unseren Betrachtungen herausgenommen werden [Neh87, S. 55]. Die Typen 2 und 3 stellen uns vor die Problematik der Synchronisation und der Darstellung. Aktionsfolgen gemäß Typ 2 können allerdings sequentialisiert werden. Dadurch müssen auch Sperrmechanismen eingeführt werden. Die folgende Voraussetzung dient als Hilfsmittel zur Identifikation einer Aktion. Aktionen vom Typ 1 werden nur streng sequentiell abgearbeitet, d. h., es ist kein *multiprozessing* erlaubt z. B. Warten beim Drucken eines Textes aus einem Dokumentensystem. Wenn das System *multiprocessing* erlaubt, kann auf mehreren Datenbereichen gleichzeitig gearbeitet werden, wenn die Bereiche disjunkt sind z. B. Übersetzen eines Programms und gleichzeitige Bearbeitung eines Dokuments.

Grundannahme 2 (Eindeutigkeit einer Zustandsänderung)
Zu einem beliebigen Zeitpunkt kann nur eine Aktion durchgeführt werden, d. h., zwei Aktionen unterscheiden sich mindestens durch den Zeitpunkt ihres Auftretens.

[4] Da jeder Rechner, den der Benutzer erreichen kann, seinen eigenen Systemtakt besitzt, wird von der Tatsache ausgegangen, daß das Historiesystem auf einem bestimmten Rechner implementiert ist. Der Systemtakt dieses Rechners liefert die Zeitinformation für das Historiesystem.

Tabelle 5.1: Die verschiedenen Typen von Aktionsfolgen

Typ 1	definierter Eingangszustand endlich (terminiert) deterministisch definierter Ausgangszustand	streng sequentiell
Typ 2	wie Typ 1 disjunkte Datenbereiche	quasi–parallel
Typ 3	wie Typ 1 nicht disjunkte Datenbereiche	parallel
Typ 4	wie Typ 1 mit Einschränkungen nicht deterministisch	allgemeine Parallelität

Obwohl die Forderung der Grundannahme deutlich ist, muß bei der Betrachtung realer Aktionen zwischen durchgeführten und den vom Befehl bewirkten Änderungen unterschieden werden. Die Grundannahme erfaßt nur die abgesetzten Befehle.

Parallel auftretende Aktionen werden, soweit möglich, sequentialisiert (Prinzip des v. Neumann-Rechners). Konkurrierende Aktionen warten, falls nötig, aufeinander. (Vergleiche auch [KHTF87b].) Werden Arbeitsschritte in den Hintergrund verlegt *(Batch)*, dann wird die Aktion nach dem Absetzen des Befehls als abgeschlossen betrachtet. Jedoch muß vermerkt werden, welche Daten von den Hintergrundprozessen beeinflußt werden, damit der Benutzer benötigte Daten nicht überschreibt oder Namen fehlerhaft belegt. Am Ende des Hintergrundprozesses wird der Benutzer benachrichtigt. Die Änderungen sind von nun an verfügbar.

Das Problem der Synchronisation von Aktionen auf nicht abgeschlossenen Zustandsänderungen wird weiter unten bei der Betrachtung der Gesamtzustände diskutiert. Eine Unterscheidung von Zustandsänderungen ist notwendig, um deutlich zu machen, wie Änderungen in der Historie behandelt werden.

Als *reale Zustandsänderung* bezeichnen wir eine vollständig durchgeführte Änderung. Eine begonnene, aber noch nicht beendete Änderung bezeichnen wir als *virtuelle Zustandsänderung*[5].

Virtuelle Zustandsänderungen sind den Änderungen durch *lazy evaluation* vergleichbar [Thi90]. So entspricht dem einfachen copy-Befehl eine reale Zustandsänderung, da der Zustand des Systems nach Abschluß des Befehls vollständig geändert wurde. Wird der gleiche Befehl im Hintergrund ausgeführt, so liegt eine virtuelle Zustandsänderung vor, da zwar die Auswirkungen der Änderung bereits bekannt sind aber real noch nicht durchgeführt wurden. Durch die Fortsetzung unterscheidet sich die virtuelle Zustandsänderung von der *lazy evaluation*. Für die Dialoghistorie bedeutet dies, daß die virtuelle Änderung eingetragen wird und für die folgenden Aktionen Beschränkungen existieren. Insbesondere bedeutet dies, daß ein Zeitpunkt

[5]Vergleiche Anhang A.3, Definition 33.

bekannt ist, zu dem zumindest die Beschränkungen festgesetzt sind, die Änderung also als abgeschlossen betrachtetet werden kann. So wie bei Archer, Conway, Schneider und Vitter erwähnt, muß auch hier die Historie die Aufhebung der Beschränkungen bekanntgeben. Grundsätzlich heißt dies, daß ein Transaktionskonzept eingeführt werden muß. Die angegebenen Beschränkungen sind dem Benutzer bekannt, dadurch werden einige Aktionen automatisch verhindert, z. B. das Edieren einer gelöschten Datei, wenn das Löschen noch nicht bearbeitet wurde, aber die Beschränkung, daß die Datei nicht mehr existieren wird, bereits bekannt ist.

Wenn nicht anders vermerkt, sind im folgenden ausschließlich virtuelle Zustandsänderungen gemeint. Systemzustandsänderung beschreiben wir formal als ein Tupel $\delta = (\phi, z)$. Dabei beschreibt ϕ die Änderung des Systemzustandes und z den Zeitpunkt der abgeschlossenen Änderung. Die Gesamtheit der möglichen Systemzustandsänderungen ist durch die Menge Δ' gegeben[6].

Die Abfolge von Benutzeraktivitäten (Zustandsänderungen) impliziert eine Ordnung der Systemzustände. Dieser Ordnung liegt wiederum der Zeitpunkt der Aktivität zugrunde. Im Vergleich dazu steht die Ordnung im Baum, die die direkten Zusammenhänge der Aktionen deutlich macht.

Für die weiteren Betrachtungen benötigen wir einen *HistorieKnoten*, der ein Knoten ist, bei dem das Datum einen Systemzustand repräsentiert[7].

Im folgenden sei $\mathcal{C}$ die Menge der Namen der Betriebssystembefehle und Benutzer- und Systemmakros. Die Menge $\mathcal{D}$ der Daten umfaßt die Dateien auf dem jeweiligen Rechner und die Eingabe und Ausgabe über das Terminal. Mit diesen Mengen kann die Ausführung eines Befehls aus $\mathcal{C}$ als Funktion auf den Daten aus $\mathcal{D}$ dargestellt werden. Die Menge aller Daten auf dem jeweiligen Rechner zu einem Zeitpunkt ist $\mathcal{D}' = \mathcal{P}(\mathcal{D})$ (Potenzmenge).

Definition 5

Befehle	$\mathcal{C} := \{c \mid c \textit{ ist Betriebssystembefehl oder Makro}\}$
Daten	$\mathcal{D} := \{d \mid d \textit{ ist Datei auf der Rechenanlage}\}$
	$\mathcal{D}' := \mathcal{P}(\mathcal{D})$
Umgebung	$\mathcal{E} := \{e \mid e \textit{ ist Umgebungsvariable}\}$
	$\mathcal{E}' := \mathcal{P}(\mathcal{E})$
Optionen	$\mathcal{O}_c := \{o \mid o \textit{ ist Option des Befehls } c\}$
	$\mathcal{O}'_C := \mathcal{P}(\mathcal{O}_c)$
Daten und Optionen	$\mathcal{D}^* := \mathcal{D}' \times \mathcal{O}'_C \times \mathcal{E}'$

Im weiteren benötigen wir Operationen auf den Mengen der Datenbestände der Zustände oder Zustandsänderungen. Dazu werden der Schnitt und die Vereinigung von Knoten definiert. Der Schnitt zweier Knoten enthält Informationen über gemeinsame Daten. Die Vereinigung enthält Informationen über die Daten in beiden Knoten. Die Differenz $\diamond$ zweier Knoten gibt die Daten wieder, die nicht zu beiden Knoten gleichzeitig gehören[8].

[6] Vergleiche Anhang A.3, Definition 34.

[7] Vergleiche Anhang A.3, Definition 36.

[8] Näheres hierzu siehe Anhang A.3.

Wir benötigen bei der Betrachtung einer Zustandsänderung auch die Möglichkeit festzustellen, ob eine Änderung im System einen Widerspruch erzeugt, d. h. Daten verändert, z. B. löscht, die an anderer Stelle referenziert werden.

Wir nennen eine Änderung δ widerspruchsfrei bezüglich einer anderen Änderung δ', wenn die Änderungen keinen gemeinsamen Datenbereich besitzen[9].

Wir nennen eine Änderung $\delta(x)$ des Systemzustands x fehlerfrei, wenn das Betriebssystem die Änderung fehlerlos ausführen kann, die Änderung widerspruchsfrei ist, die Oberfläche $\mathcal{W}$ keine Fehler meldet, und es keinen Konflikt mit der Umgebung $\mathcal{E}$ gibt[10]. Die Gesamtheit aller fehlerfreien Zustandsänderungen wird durch die Menge Δ beschrieben.

Im folgenden verwenden wir einen HistorieBaum gemäß Definition 43[11]. Jeder Nachfolger eines Knotens ist durch die Zustandsänderung seines Vorgängers bestimmt. Dieser Zusammenhang wird durch die Identifikatoren beschrieben.

5.2.2 Benutzeraktionen

Allgemeine Betrachtungen

Bisher wurde nur die Struktur untersucht, die der Dialoghistorie zugrunde liegt. Es wurden Bedingungen herausgearbeitet, wann Operationen ausführbar sind und wie sich die Dialoghistorie aufbauen läßt. Im weiteren werden nun die Operationen des Dialogs und die Zusammenhänge mit der Dialoghistorie untersucht. Dabei wird auch auf die Systembefehle eingegangen und festgestellt, welche Befehle redundant sind oder gesondert behandelt werden müssen. Redundanz ist auch bei der Betrachtung von Makros wichtig.

Die Benutzeraktionen sind bezogen auf das System durch die vier Grundfunktionen *undo, redo, again* und *do* gekennzeichnet. Die Bedeutung der Funktionen ist in Tabelle 5.2 erläutert[12].

Tabelle A.4.2 im Anhang A.4.2 (Seite 174) enthält eine Übersicht über die Befehle des Betriebssystems VMS 5.x der Firma Digital Equipment [Dig90]. In dieser Übersicht werden die Befehle im Zusammenhang mit Aktionen in Aktionsprotokollen betrachtet. Aus der Übersicht ist erkennbar, daß einige Befehle nicht mitprotokolliert werden müssen (durch '−' in der Spalte „Hist" gekennzeichnet), andere können ohne Betrachtung des Systemzustandes ausgeführt werden, z. B. der Befehl `LINK`[13].

[9] Vergleiche Anhang A.3, Definition 40.

[10] Vergleiche Anhang A.3, Definition 41.

[11] Vergleiche Anhang A.3.

[12] Näheres hierzu im Anhang A.3.

[13] Eine äquivalente Darstellung für das Betriebssystem *Concentrix 5.0* der Firma Alliant (Berkeley UNIX BSD 4.3) [All89] ist in Tabelle A.4.3 (Seite 174) dargestellt. Diese Darstellung der wichtigsten UNIX–Befehle kann für alle UNIX–Rechner übernommen werden. Die Befehle einer direktmanipulativen Oberfläche sind im wesentlichen *selektieren, bewegen, verändern.* Dabei entspricht der Befehl *verändern* einer großen Gruppe von Befehlen, mit denen Eigenschaften des Objekts modifiziert werden können z. B. *schließen* oder *vergrößern* eines Fensters.

Tabelle 5.2: Bedeutung der Funktionen *do, undo, redo, again*

do	Ausführen eines Betriebssystembefehls. Dieser Befehl kann auch ein Makro mit definiertem Ausgangs- und Zielzustand sein.
undo	Zurücknehmen eines Betriebssystembefehls oder eines Systemzustandes. Es ist zu berücksichtigen, wie weit die Historie verfolgt werden soll, und insbesondere welche Befehle *UnDo*–fähig sind.
redo	Wiederholen einer mit *UnDo* zurückgenommenen Sequenz. Hier ist zu berücksichtigen, welche Befehle *ReDo*–fähig sind.
again	Wiederholen einer Befehlsfolge oder Zustandssequenz. Im Gegensatz zum *redo* wird die Historie erweitert, auch, wenn die Systemzustände bereits in einem Knoten des *HistorieBaumes* enthalten sind. Die Unterscheidung der Zustände auf Systemebene erfolgt durch die zeitliche Abfolge zweier Zustände.

Die in diesen Tabellen zusammengestellten Daten sind Grundlage für die erwähnten Benutzeraktionen im *HistorieBaum*. Mit dieser im folgenden ausgeführten Betrachtung soll die Definition der Benutzeraktionen im *HistorieBaum* vereinfacht werden. Für den *HistorieBaum* gilt insbesondere die Definition 46[14] für das Anfügen eines *HistorieKnotens*.

Einzelne Knoten und Zustände im Baum

Im *HistorieBaum* gibt es aktive und aktuelle Zustände (respektive Elemente). Der aktuelle Zustand ist derjenige Zustand, auf dem die nächste Aktion des Benutzers operiert. Dieser Zustand ist im Gesamtkontext nur an Hand einer Markierung, d. h. dem Zeitpunkt des Erreichens, erkennbar. Daher muß diese Markierung bestimmt werden.

Zuvor muß der Zeitpunkt einer Operation, die den Baum verändert, bestimmt werden.

Definition 6
Sei a eine Aktion auf der Dialoghistorie t, d. h. do, undo, redo, again. Die Projektion ζ ordnet einem Zustand x in der Historie den Zeitpunkt der abgeschlossenen Aktion a zu.

Wir präzisieren nun Definition 2.

Definition 7 (aktive und aktuelle Zustände)

[14]Siehe Anhang A.3.

aktiv *Ein Zustand im HistorieBaum heißt aktiv, wenn er durch ein undo nicht zurückgenommen wurde.*

aktuell *Ein Zustand y im HistorieBaum ist aktuell*
$:\Longleftrightarrow$ *er wurde gerade eingefügt (do)* $\vee$
er wurde durch ein undo erreicht $\vee$
er wurde durch ein redo wieder aktiviert.

Eine einfache Folgerung, die sich sofort aus den Definitionen ergibt, ist das folgende Lemma.

Folgerung 1 *Ein aktuelles Element ist auch selektiert.*

Wir zeigen nun, daß die aktiven Zustände in einer Historie einen Baum bilden.

Satz 1 *Sei t ein HistorieBaum. Sei $\mathcal{K}^*$ die Menge der aktiven Knoten im Baum $\mathcal{K}$. Dann ist $\mathcal{K}^*$ ein Baum im Sinne der Definitionen.*
Beweis: Es ist zu zeigen, daß

(i) eine Wurzel in $\mathcal{K}^*$ existiert,

(ii) die Positionen verschiedener Knoten in $\mathcal{K}^*$ nicht identisch sind und

(iii) es genau einen Pfad von der Wurzel zu jedem Knoten in $\mathcal{K}^*$ gibt.

Der Punkt (ii) ist trivialerweise erfüllt, da $\mathcal{K}^*$ eine Teilmenge von $\mathcal{K}$ ist. Sei $k \in \mathcal{K}$ ein beliebiger aktiver Knoten, dann ist auch sein Vorgänger aktiv. Denn wäre der Vorgänger nicht aktiv, ist er durch ein *undo* deaktiviert worden. Dies ist richtig, da nach Def. 46[15] nur aktive Knoten angefügt werden. Damit ist nach Def. 32[16] auch k inaktiv. Dies kann bis zur Wurzel w von $\mathcal{K}$ verfolgt werden, d. h., $w \in \mathcal{K}^*$. $\mathcal{K}^*$ besteht daher aus der Wurzel von $\mathcal{K}$ und deren Nachfolgern, jeweils bis zu einem durch $\mathcal{K}^*$ bestimmten Knoten in den einzelnen Pfaden. Damit sind auch (i) und (iii) erfüllt. ⊙

Mit t^* bezeichnen wir den Baum aktiver Zustände, mit $\mathcal{T}^*$ die Menge der möglichen aktiven Bäume.

Die folgenden Befehle sind Systemaktionen, die vom Benutzer durch eine Aktion auf dem Baum bzw. unter Verwendung des Baumes ausgeführt werden. Die Befehle werden wie folgt beschrieben:

doit(c,d)	Entspricht dem *do*. Die Benutzeraktion *do* wird in die durch c und d bestimmte Systemaktion überführt.
undoit(c,d)	Entspricht dem *undo*. Die Benutzeraktion *undo* wird in die durch c und d bestimmte Systemaktion überführt.
redoit(c,d)	Entspricht dem *redo*. Die Benutzeraktion *redo* wird in die durch c und d bestimmte Systemaktion überführt.
doitagain(c,d)	Entspricht dem *again*. Die Benutzeraktion *again* wird in die durch c und d bestimmte Systemaktion überführt.

[15]Siehe Anhang A.3.
[16]Siehe Anhang A.3.

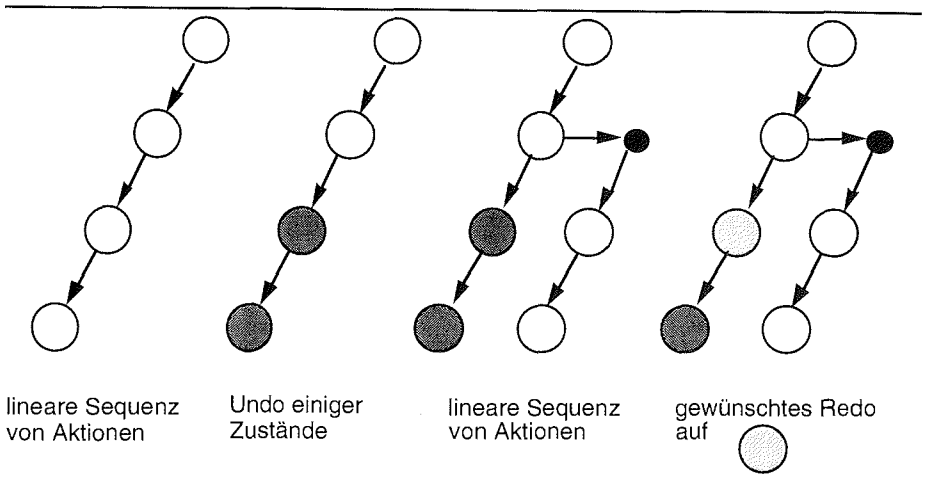


Abbildung 5.3: Diese Darstellung zeigt das Problem des *ReDo* in einem *HistorieBaum*. Einzelheiten siehe Text.

Ein Befehl oder eine Aktion des Benutzers wird durch die Funktion *doit* auf die Systemebene geführt. In dieser Funktion sind alle Informationen über das zugrundeliegende System enthalten. Das Ergebnis ist ein Zustand. Die beschriebenen Funktionen sind Metafunktionen, die der Ausführung des Befehls mit den übergebenen Parametern entspricht. *doit* ist die Ausführung der Zustandsänderungsfunktion δ auf Systemebene. Bezüglich des *HistorieBaumes* kann die Befehlsausführung *do* durch eine Hintereinanderausführung der Funktionen *doit* und *append* definiert werden.

Die Funktion *undoit* führt die reverse Aktion der im Zustand beschriebenen Aktion aus. Die *UnDo*–Operation auf dem *HistorieBaum* nimmt einen aktiven Zustand zurück.

Bei der Ausführung von Befehlen durch *do* wird mittels der Funktion *doit* eine Zustandsänderung $\delta(x) = y$ durchgeführt. Diese Zustandsänderung ist nur dann gültig, wenn durch die Änderung Daten in anderen Zuständen nicht unzulässig beeinflußt werden. D. h., eine komplexe Operation, die Dateien, die bei der Zustandsänderung zu einem Blatt erzeugt wurden, entfernt und die nicht der Nachfolger des einen Widerspruch erzeugenden Blattes ist, wäre danach eine ungültige Zustandsänderung. (Siehe Abbildung 5.3.). Die soeben beschriebene Zustandsänderung nennen wir *gültig*.

Das in der Abbildung 5.3 gezeigte Problem besteht in den Fragen „Darf auf den gekennzeichneten Zustand die *ReDo*–Operation angewandt werden?“, „Welche Widerprüche entstehen im System?“. Die bisherige Sprechweise impliziert einen globalen Zustand, der durch die Vereinigung der Blätter des Baumes repräsentiert

wird. Eine Systemzustandsänderung kann auf diesen Zustand angewandt werden. Ein eventueller Widerspruch wird sich dann, wenn er auftreten sollte, sofort zu erkennen geben. Für die Zustände im Baum heißt dies aber, daß nur Teile des Gesamtzustands gezeigt werden und die wesentliche Information in dem Übergang zwischen den Zuständen besteht, wie es bereits vorher richtig definiert wurde.

Ausgehend von der Situation, daß in einem *HistorieBaum* die Zustandsinformationen in den Blättern enthalten sind, muß nun ein Übergang zu einem *globalen Zustand* $\mathcal{G}$ erfolgen.

Definition 8
Seien $b_1, \ldots, b_k \in t$ Blätter im Baum t. Der globale Zustand $\mathcal{G}$ wird wie folgt definiert.

$$\mathcal{G} := \bigcup_{1 \leq i \leq k} \mathcal{D}(b_i)$$

Der globale Zustand hat Ähnlichkeit mit den *states* im Modell von Archer, Conway und Schneider. Im vorliegenden Fall gibt es aber die Möglichkeit, den Zustand entlang eines Pfades oder sogar eines Teilbaumes zu betrachten. Zwei Knoten haben mindestens die Wurzel des Baumes als gemeinsamen Vorgänger. Sonst muß ein *maximaler gemeinsamer Vorgänger* bestimmt werden. (Siehe 49 im Anhang.)

Der *globale Zustand* $\mathcal{G}$ ist dynamisch, d. h., er ändert sich nach jeder Benutzeraktion. Der globale Zustand ist von nun an Grundlage der weiteren Betrachtung. Unter der Annahme des globalen Zustands kann eine gültige Zustandsänderung definiert werden.

Definition 9 (Gültige Zustandsänderung)
Eine Zustandsänderung heißt gültig, wenn der von der Änderung erzeugte Zustand in keiner Weise dem globalen Zustand des Baumes t^* widerspricht mit Ausnahme des Zustands des Blattes, auf dem die Zustandsänderung operiert.

In jedem Falle gibt es Knoten, deren Zustandsbeschreibungen unabhängig sind. Diese Unabhängigkeit wird mit Hilfe des Datenbestandes zu einem Knoten definiert. (Vergleiche Definition 52.)

Die Betrachtung des *HistorieBaumes* nach einem *do* ist einfach, da sich der Baum ständig vergrößert. Auch die Selektion eines Elementes und ein folgendes *do* beinträchtigen den Baum kaum. Nach einem *undo* ist jedoch ein Element oder ein ganzer Teilbaum verändert. (Vergleiche Abbildung 5.4.). Dieses Element (der Teilbaum) wird *deaktiviert*, d. h., die in ihm definierten Operationen (Zustandsänderungen) müssen rückgängig gemacht werden. Bei einem *redo* wird entprechend ein Element (ein Teilbaum) wieder *aktiviert*, d. h., es müssen die entsprechenden Operationen (Zustandsänderungen) wieder ausgeführt werden.

Die Freiheit, Aktionen an einem beliebigen Platz im Baum zu plazieren, wird durch die Bedingung der Unabhängigkeit eingeschränkt. Bei der Bestimmung der Unabhängigkeit ist die Semantik der Mengenoperationen zu beachten. Hier kann folgendes unterschieden werden:

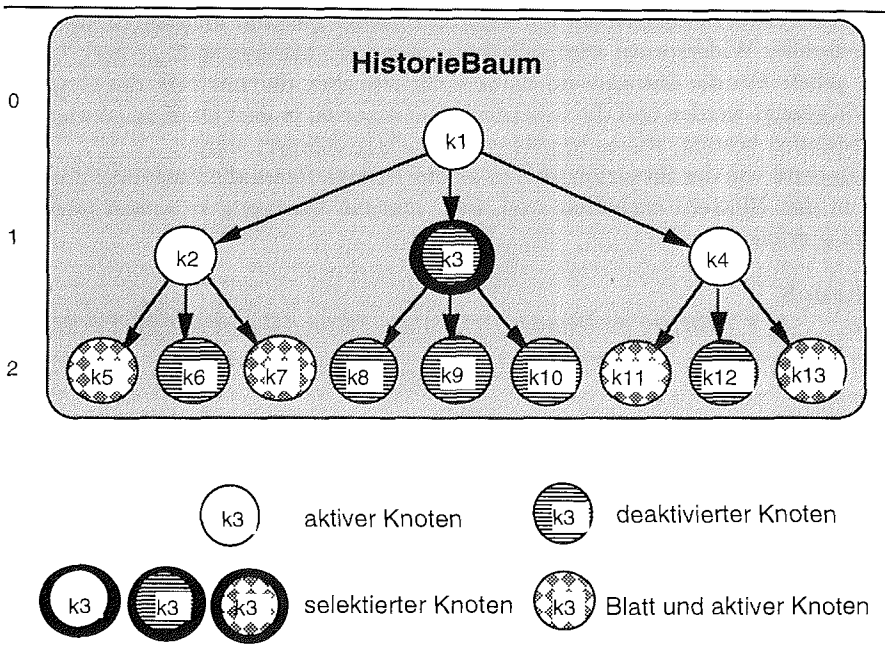


Abbildung 5.4: Die dargestellten Blätter und aktiven Knoten seien im Sinne von Definition 52 unabhängig. Wenn Knoten k_3 aktiviert wird, d. h., ein *ReDo* ausgeführt werden soll, dann findet eine Überprüfung gemäß Definition 52 statt.

Dateien: Die Überprüfung von Dateien wird über zwei Kriterien durchgeführt.

Dateiname: Dateien werden nur auf Grund der Existenz überprüft. Eine Datei, d. h. der Dateiname, kann vorhanden sein oder nicht. Hier ist das Ergebnis der Mengenoperationen leicht zu entscheiden.

Dateninhalte: Eine Prüfung der Dateninhalte von Dateien wird nicht vorgenommen. Jedoch kann durch Vergleich des Änderungszeitpunkts einer Datei auf eine Modifikation geschlossen werden und damit ein Unterschied festgestellt werden.

Optionen: Eine Option steht immer im Kontext des Befehls oder gegebenenfalls eines Parameters. Es gilt hier der direkte Vergleich der Aktionen, der das Ergebnis der Mengenoperation liefert.

Umgebungsvariablen: Umgebungsvariablen können im Verlauf einer Sitzung ihren Wert verändern. Bei der Überprüfung werden die Werte der Variablen miteinander verglichen und nicht deren Existenz. Insbesondere sind dabei nicht definierte Variable gesondert zu berücksichtigen. Eine in einem Teilzustand

Tabelle 5.3: Eine Befehlssequenz, die ausgeführt, zurückgenommen und wiederholt wird:

Befehl ausführen	$do(t, k, k')$
	$doit(c, d)$
Befehl zurücknehmen	$undo(t, k)$
	$undoit(c, d)$
Befehl wiederholen	$redo(t, k)$
	$redoit(c, d)$

nicht existierende Variable wird als existent mit einem nicht definierten Wert angesehen.

Die Abfolge einer Befehlssequenz, die ausgeführt, zurückgenommen und wiederausgeführt wird, ist in der Tabelle 5.3 dargestellt. Die Sequenz wird vom *HistorieBaum* aus betrachtet.

Eine für das System relevante Operation ist das Abschneiden weit zurückliegender Ereignisse. Die Funktion *cut* spaltet den Baum in mehrere Teilbäume auf. Diese Funktion ist wegen der begrenzten Speicherkapazität der Rechenanlagen einzuführen.

Die Funktion *cut* sammelt alle Blätter des Baumes t^* auf, die nicht Element des Teilbaumes t_s sind. $t_s \subset t$ ist der Teilbaum, der als Wurzel das selektierte Element s hat. Dann erstellt *cut* einen Zustand, der der Vereinigung aller unabhängigen Blätter und der Wurzel s entspricht.

Die Aufgaben der Funktion *cut* umfassen also folgende Aspekte:

- Abschneiden der vorhergehenden Historie
- Beibehalten des aktuellen globalen Systemzustands
- Beseitigen der freigesetzten *HistorieKnoten*

Die Funktion *cut* betrachtet den *globalen Zustand* $\mathcal{G}$, sichert diesen Zustand und stellt den Zustand $\mathcal{G}$ durch chronologische Vereinigung verbliebener Zustände und durch das *cut* verlorengeganger Zustände wieder her. Die Blätter des Baumes müssen hier gewählt werden, da sie allein als Endpunkte einer Benutzeraktion den globalen Zustand repräsentieren.

Eine Folge dieses Verfahrens zur Durchführung der Funktion *cut* ist, daß der Benutzer auf der entstehenden Wurzel kein *undo* ausführen kann. In den Abbildungen 5.5 und 5.6 (Seiten 103 bzw. 104) ist die Situation noch einmal dargestellt.

Eine Funktion *cutit* ist hier nicht erforderlich, da sich am Datenbestand keine Änderungen ergeben. *cutit* wäre daher die identische Abbildung.

Definition 10 (Cut–Funktion)
Sei t' der Ausgangszustand des Baumes, $\mathcal{G}'$ der globale Zustand des neuen Baumes t, s der selektierte Zustand.

$$cut \ : \ \mathcal{T} \times \mathcal{K} \rightarrow \mathcal{T}, (t', x) \mapsto t$$

Das Funktionsergebnis wird nach folgenden Regeln generiert:

1. *Der Teilbaum mit der Wurzel s bleibt unverändert.*

   $$\forall k = (p, x, f) \in t' : k \in \mathcal{S}(t', s) \wedge (k \text{ ist Blatt in } t') \Longrightarrow (k \in t) \wedge (k \text{ ist Blatt in } t)$$

2. *Die Positionen in diesem Teilbaum werden auf die Position der Wurzel reduziert.*

   $$\forall z \in \mathcal{S}(t', s) \ : \ p(z) \leftarrow p(pred(s)) \mid p(z)$$

3. *Die nicht in diesem Teilbaum enthaltenen Blätter bilden einen neuen Zustand.*

   $$\mathcal{G}' \ = \ \bigcup_{\substack{x \notin \mathcal{S}(t', s) \ \wedge \\ x \text{ ist Blatt}}} \mathcal{D}(x)$$

4. *Eine neue Wurzel wird generiert.*

   $$w \leftarrow (0, \mathcal{G}')$$

5. *Der Teilbaum t^* mit s als Wurzel wird an die neue Wurzel angefügt.*

   $$t = append(\{w\}, t^*, k(w))$$

6. *Alle Knoten des nun entstandenen Teilbaumes müssen an den globalen Zustand angepaßt werden.*

   $$\begin{array}{l} \text{Sei } \mathcal{G}'' := \mathcal{G} \diamond \mathcal{G}'. \\ \forall x \in t \ \forall g \in \mathcal{G}'' \ : x \ \succ\!\!\succ \ g \Longrightarrow \mathcal{D}(x) := \mathcal{D}(x) \cup \mathcal{D}(g) \end{array}$$

Die Funktion *cut* ist nicht auf jeden Knoten einer Dialoghistorie anwendbar. Im schlimmsten Fall muß der Baum auf einen Knoten, der den momentanen, globalen Zustand beschreibt, reduziert werden. Im folgenden zeigen wir, wann *cut* anwendbar ist und welche Eigenschaften sich daraus ergeben. Dazu benötigen wir die folgenden Lemmata.

Lemma 1
Seien δ_0 und δ_1 gültige Zustandsänderungen. Dann ist die Zustandsänderung $\delta := \delta_1 \circ \delta_0$ ebenfalls gültig.
Beweis: (Beweis durch Kontraposition) Ist δ nicht gültig, dann existiert ein Widerspruch in den Daten nach Definition 9. Ist δ_0 gültig, so folgt, daß δ_1 den Widerspruch erzeugt. Ist

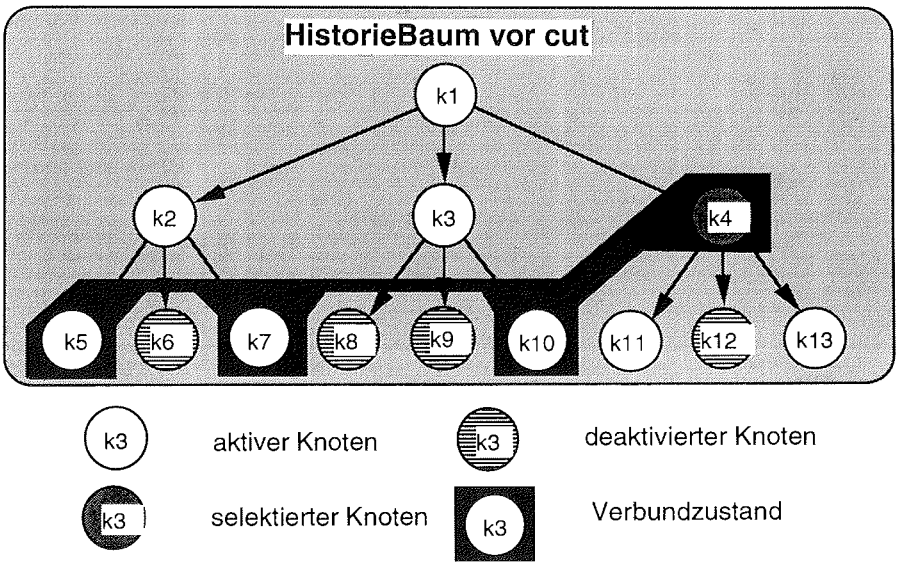


Abbildung 5.5: Der *HistorieBaum* ist hier mit aktiven, deaktivierten Elementen und einem selektierten Element vor der Ausführung der *cut*–Operation gezeigt.

hingegen δ_0 nicht gültig, dann sind folgende Fälle zu betrachten:
(i) δ_1 ist von δ_0 unabhängig. Dann bleibt der Widerspruch auch nach der Ausführung von δ_1 erhalten.
(ii) δ_1 ist von δ_0 abhängig. Dann kann δ_1 fehlerfrei ausgeführt werden.
In beiden Fällen ist mindestens eine der Änderungen nicht gültig. ⊙

Es folgt durch Induktion sofort das folgende Lemma.

Lemma 2
Seien $\delta_0, \delta_1 \ldots \delta_n$ gültige Zustandsänderungen. Dann ist die Zustandsänderung $\delta := \delta_n \circ \ldots \circ \delta_0$ ebenfalls gültig.

Satz 2 (Ausführbarkeit von *cut*)
Sei s das selektierte Element im aktiven Baum t. Die cut–Funktion ist ausführbar, wenn gilt

$$z(s) = \max\{\zeta(t)\}$$

oder

$$\forall k \in t,\ k \notin \mathcal{S}(t,s)\ \forall k' \in \mathcal{S}(t, pred(s)) \cup \{s\}$$
$$(k \textit{ ist Blatt}) \wedge (k \textit{ und } k' \textit{ sind unabhängig})$$

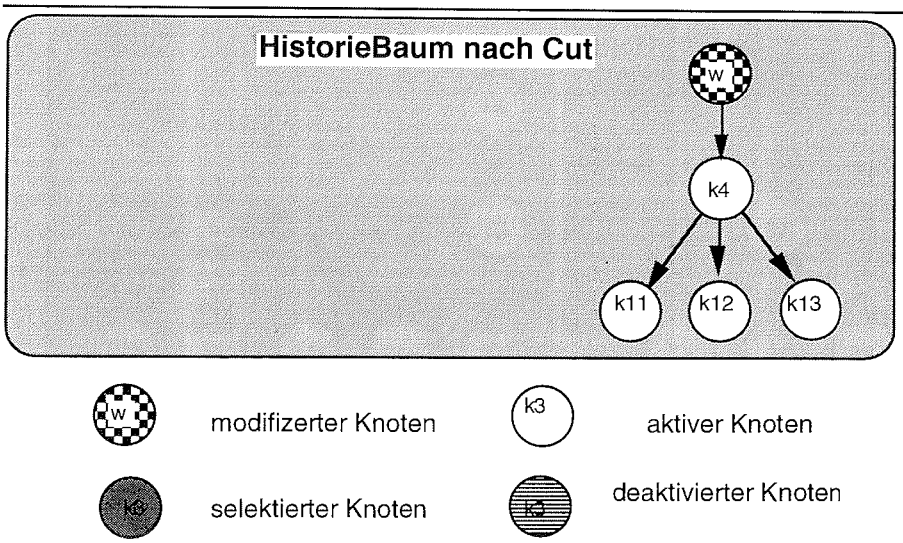


Abbildung 5.6: Der *HistorieBaum* nach der Ausführung der *Cut*–Operation. Die aktiven Elemente außerhalb des Bereiches des selektierten Elementes werden zu einem modifizierten Element zusammengefaßt.

Beweis: Die erste Aussage ist trivial. Die zweite Aussage wird durch Kontraposition bewiesen. Sei t^* der abgetrennte Teilbaum. Sei $k \in t$ ein Blatt mit $k \notin \mathcal{S}(t,s)$. Sei $k' \bullet \mathcal{S}(t, pred(s))$. Seien k und k' nicht unabhängig. Dann existiert eine nicht gültige Änderungsfunktion $\delta(k')$ zum Knoten k'. Nach Definition 41[17] ist $\delta(k')$ nicht fehlerfrei. Nach Definition 46[18] ist ein Anfügen nicht möglich, d. h., ein *cut* ist nicht ausführbar. $\odot$

Der Satz sagt aus, daß ein *cut* auf dem zeitlich letzten aktiven Knoten ohne Probleme durchführbar ist. Der verminderte globale Zustand $\mathcal{G}' = \mathcal{G} \diamond \mathcal{D}(\mathcal{S}(t,s))$ beschreibt akurat die Situation *nach* Anwendung von $\delta(s)$.

Weiterhin kann ein *cut* ausgeführt werden, wenn keine Beeinflussung von Daten aus Knoten des Teilbaumes mit Wurzel s und den übrigen Knoten besteht. Dann wird ein globaler Zustand $\mathcal{G}'$ erzeugt, der den Zustand des Systems beschreibt, nachdem $\delta(s)$ angewandt wurde.

In allen anderen Fällen, in denen eine Beeinflussung besteht, muß der selektierte Teilbaum verändert werden. Dies ist ein Eingriff in die Dialoghistorie, da die Zustandsänderungen δ für die Generierung des jeweiligen globalen Zustands angepaßt werden müssen. Die Änderung δ', die durch diesen Vorgang entsteht, entspricht nicht mehr der Aktion des Benutzers. Daher wird dieser Weg ausgeschlossen.

[17] Siehe Anhang A.3.
[18] Siehe Anhang A.3.

Wenn vorausgesetzt wird, daß die Funktion im Sinne von Satz 2 ausgeführt wurde, kann gefolgert werden, daß die Funktion *cut* einer Zustandsänderungsfunktion δ' äquivalent ist, die aus einem Grundzustand das System in den durch *cut* erzeugten Zustand versetzt (Vergleiche Lemma 2).

Neben der Funktion *cut* kann eine Funktion *combine* definiert werden, die ähnlich wie die Funktion *cut* arbeitet, jedoch ohne Daten aus dem Baum zu vergessen. Die *Combine*–Funktion erweitert die Funktionalität für die Fälle, in denen die nächste Aktion auf dem bestehenden Baum einen Widerspruch erzeugen würde.

Wenn die aktiven Zustände zu einem Zustand zusammengefaßt werden, kann eine beliebige neue Aktion ausgeführt werden, ohne einen Widerspruch zu erzeugen. Die Wirkung der *Combine*–Funktion kann durch ein *uncombine* zurückgenommen werden. Sowohl *combine* als auch das *uncombine* operieren *nicht* auf den Zuständen, sondern auf dem Baum. Die Funktion *combine* entspricht der *Checkpoint*–Funktion, die von Archer, Conway und Schneider sowie Thimbleby eingeführt wurde, um ein Rücksetzen auf bestimmte Zustände zu ermöglichen. Diese ersetzt nicht die von Herczeg geforderten verschiedenen *UnDo/ReDo*–Operationen. Ein *uncombine* auf eine früheres *combine* setzt alle seitdem generierten Aktionen zurück.

Bisher wurden einzelne Zustandsänderungen betrachtet. Durch eine kanonische Erweiterung der definierten Funktionen können auch Anweisungssequenzen bzw. Folgen von Zustandsänderungen betrachtet werden.

5.2.3 Ergebnis

Ziel der Einführung des *DialogHistorieBaumes* war, aus verschiedenen Ansätzen ein implementierbares Modell zu erzeugen, das in der Lage ist, die von [Her86a] definierten Mechanismen für *UnDo*, *ReDo* und *Again* anzuwenden.

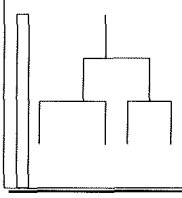
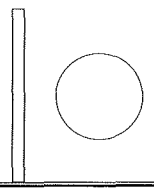
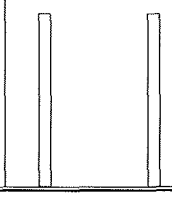
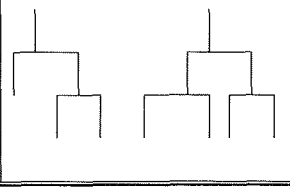
Die Funktionen für *UnDo*, *ReDo* und *Again* können auf die Herczegs'schen Funktionen zurückgeführt werden, wenn die selektierten Pfade keine *aktiven* Verzweigungen haben. Die Funktionen gehen über in *backtracking*–ähnliche Funktionen, wenn *aktive* Verzweigungen vorliegen. Jeder Pfad wird bis zu einer aktiven Verzweigung mittels Herczeg'scher Funktionen zurückgenommen, dann erst werden die Verzweigungen entfernt, bevor auf dem Pfad weiter zurückgegangen wird. Eine Konsistenzüberprüfung ist auf jeden Fall vorzunehmen.

Außerdem wird *UnDo* etc. zunächst simuliert. Erst wenn keine Widersprüche auftreten, wird die Aktion an das Betriebssystem weitergeleitet.

Durch die Konstruktion der Dialoghistorie wird sowohl eine lineare Historie wie auch eine Baumstruktur unterstützt. Bei den Historiebäumen können zwei Fälle unterschieden werden. Im ersten Fall besteht die Historie aus voneinander unabhängigen Handlungsfolgen, z. B. Verzweigung einer Literaturrecherche nach unterschiedlichen Themengebieten. Im zweiten Fall, der insbesondere in dieser Arbeit untersucht wurde, werden Abhängigkeiten zwischen Aktionssequenzen berücksichtigt.

Die Tabelle 5.4 stellt die diskutierten Modelle in einer Übersicht gegenüber. Damit wird zum Einen ein Zusammenhang der Modelle deutlich, andererseits aber

Tabelle 5.4: In der Tabelle werden die Modelle von Vitter [Vit84], Yang [Yan88b], Herczeg [Her86a] und das in der vorliegenden Arbeit benutzte Modell gegenübergestellt.

Vitter	Yang	Herczeg	LPK
Baum	Ringliste	Lineare Liste	Baum
	meta undo = travel undo		
history script state	 script state	history state	history script state
state script	state *UnDo/Re-Do*–script	state *UnDo/Re-Do*–script	history *UnDo/Re-Do*–script
			

auch die Differenz in der Betrachtungsweise, Anwendung und Funktionsweise.

5.3 Scripte

Der Benutzer, der sein System an seine Bedürfnisse anpassen möchte, kann dies auf den meisten Rechnern durch Erstellen von Profilen, sogenannten *startup*– oder Kommando–Dateien verwirklichen. Diese Dateien, die beim Anmelden an das System einmal ausgeführt werden, setzen bestimmte, vom Benutzer einmal festgelegte Parameter des Systems. Änderungen sind aufwendig, das System ist statisch.

Eine Erleichterung ist die automatische Parametrisierung einer Aktionsfolge, wenn sie in einer Kommandofolge zusammengefaßt werden. Dabei sind sowohl die Befehle als auch die Parameter Beschränkungen unterworfen, z. B. redundante Befehle.

Mittels vom Betriebssystem ausführbarer Kommando–Dateien kann ein Benutzer Makros bereitstellen, also Folgen von Befehlssequenzen, die in gleicher Folge immer wieder auftreten werden, so z. B. Dateien, die im Hintergrund *(Batch)* ablaufen sollen.

Die vorgestellte Dialoghistorie ist ein geeignetes Hilfsmittel, um aus durchgeführten Sequenzen Makros zu generieren. Dabei muß auf Redundanzen ebenso geachtet werden wie auf die Parametriesierung der Befehle. Ein Makro wird aus einer Beispielsequenz generiert, abstrahiert und parametrisiert. Dazu und zur Erkennung sind geeignete Werkzeuge wie Parser, Compiler und Erkenner notwendig.

Der *Make*–Befehl der UNIX–Welt ist eine Ablaufsteuerung für Kommando–Dateien, die z. B. den Compilations- und Bindevorgang eines Programms steuern.

In diesen speziellen Kommando–Dateien werden parametrisierte Befehle des Betriebssystems mit Kontrollflußsteuerbefehlen zu einem Programm zusammengefügt. Unter UNIX heißen diese Dateien *Script*.

Der Begriff des *Scripts* wird häufig mit unterschiedlicher Bedeutung benutzt. Für die weitere Betrachtung benötigen wir daher eine Definition dieses Begriffs.

Definition 11 (Script)
Ein Script ist eine Wissensstruktur, die stereotype Folgen von Aktionen enthält. (Siehe [Sha87, SA77].) Ein Script wird hier als eine Zusammenfassung von parametrisierten Handlungen des Benutzers betrachtet. Diese Zusammenfassung erhält einen neuen Namen, unter dem das Script aufgerufen werden kann.

Diese Definition schließt den Gebrauch des Begriffes *Script* in UNIX ein. Ein *Script* in diesem Sinne wird häufig auch *Makro* genannt. Um Verwechselungen auszuschließen, wird im weiteren der Begriff *Makro* verwendet. In einem ergonomischen System werden feste Anforderungen an einen Makro gestellt. Diese Anforderungen erleichtern dem Benutzer die Erstellung und den Umgang mit Makros.

Funktionen: Die in einem Makro enthaltenen Funktionen entsprechen den vom Benutzer einzugebenden Befehlen. Diese Befehle sind:

- Befehle der Oberfläche
- Befehle des aktiven Betriebssystems

Änderungen des Makros: Der Benutzer sollte Änderungen am erstellten Makro leicht durchführen können. Dieses Verfahren kann aber die Redundanzfreiheit und Korrektheit des Makros beeinträchtigen. Im einzelnen heißt dies:

- *Makros sind edierbar.* Der Text eines Makros kann vom Benutzer mit einem Editor bearbeitet werden.
- *Makros sind auf einer Metaebene edierbar.* Der Benutzer kann einen Makro mit Metabefehlen verändern, z. B. werden Abfragen von Zuständen in einem Makro mit IF... THEN... ELSE–Konstrukten erstellt.

Änderungen durch den Makro: Die durch Anwendung des Makros erzeugten Änderungen werden in der Makro–Spezifikation bereits angegeben.

Parameter: *Makros sind parametrisierbar.* Ein Makro kann mit Parametern versehen werden, wie z. B. Dateinamen oder Zustandsvariablen, die aus dem Kontext gewonnen werden.

Redundanz: *Makros sind redundanzfrei.* Ein zu erstellender Makro wird vor der Speicherung auf mögliche Redundanzen geprüft. Dabei sind Befehle oder Befehlsfolgen gemeint, die keine wesentliche Änderung des Zustands bewirken wie z. B. ein `type`- oder `dir`–Befehl unter `VMS`, mit dem ein Text bzw. der Inhalt eines Verzeichnisses nur angezeigt wird. Diese Prüfung wird in vielen Fällen negativ ablaufen, wie z. B. die Textbearbeitung mit LaTeX, wenn das endgültige Dokument erstellt wird[19].

Korrektheit: *Makros sind syntaktisch korrekt.* Eine Überprüfung eines Makros auf Korrektheit ist notwendig, um auf Fehlverhalten angemessen zu reagieren. Fehlverhalten sollte nicht auftreten. Gegebenenfalls ist eine ausreichende Fehlerbehandlung vorzusehen.

5.3.1 Aufbau von Makros

Ein Makro besteht aus einer Folge von Befehlen, die vom Benutzer unter einem neuen Namen zusammengefaßt werden. Die urprünglichen Befehle haben Parameterlisten, die leer oder recht lang sein können. Eine Analyse der Parameterlisten, die die Listen auf identische Parameter durchsucht, ist erforderlich. Das Ergebnis ist ein parametrisiertes Makro, innerhalb dessen die Parameter korrekt zugeordnet werden. Dazu ein Beispiel:

Original Sequenz	Parametrisierter Makro
`copy file1 file2`	`copy p1 p2`
`edit file1`	`edit p1`
`compile file1 file3`	`compile p1 p3`
`link file1 file3 lib1`	`link p1 p3 p4`
`run file1`	`run p1`

Der neue Makro wird nun durch `script p1 p2 p3 p4` aufgerufen bzw. interaktiv durch Auswahl des Makros. Um eine vollständige Analyse durchführen zu können, ist ein umfangreicher *parser* für jedes Betriebssystem erforderlich. Im Rahmen dieser Arbeit ist eine Beschränkung auf einige wesentliche und häufig benutzte Befehle notwendig geworden.

Wir müssen uns einige Begriffe kurz in Erinnerung rufen. Ein Befehl ist ein Symbol (eine Zeichenkette), das eine Aktion im System repräsentiert. Ein Befehl benötigt grundsätzlich nähere Angaben über die Objekte, mit denen er operiert. Diese Angaben können zusätzlich weiter spezifiziert werden. Dies erfolgt über Parameter und Optionen.

Ein Parameter ist ein Platzhalter für ein Symbol (eine Zeichenkette). Ein Parameter variiert einen Befehl oder einen Makro in seiner Wirkung. Ein Parameter

[19]Der Anwender muß, um ein vollständiges Dokument mit korrektem Inhaltsverzeichnis und korrekten Verweisen zu erhalten, LaTeX *dreimal* mit den gleichen Parametern aufrufen. Das bedeutet, daß zwei Aufrufe bei einer einfachen Redundanzprüfung entfernt würden.

wird durch vom Benutzer definierte Werte instanziiert. Ein Parameter oder ein Befehl können weiter spezifiziert werden.

Eine Option variiert Eigenschaften eines Befehls, Makros oder eines Parameters[20].

Dazu ein kurzes Beispiel, das die Verwendung der Begriffe Befehl, Parameter und Optionen verdeutlichen hilft.

`BACKUP/LABEL=BCK001/BLOCK=65024 *.* TAPE:GRAPHBACKUP/SAVE`

In diesem Fall wurde eine Sicherungskopie auf das Bandlaufwerk `TAPE:` auf die Datei `GRAPHBACKUP` geschrieben. Der Befehl ist `BACKUP`, die Parameter sind `*.*` und `TAPE:GRAPHBACKUP`. Die Befehlsoptionen sind `/LABEL=BCK001` und `BLOCK=65924`. Der erste Parameter hat keine Optionen, der zweite Parameter wird durch die Parameteroption `/SAVE` als Sicherungskopie gekennzeichnet.

`BACKUP/LABEL=BCK001/BLOCK=65024 TAPE:GRAPHBACKUP/SAVE KRITZEL:*`

In diesem Fall wurde eine Sicherungskopie wieder eingespielt *(restore)*. Vom Bandlaufwerk `TAPE:` wurde die Datei `GRAPHBACKUP` gelesen und auf den Plattenspeicher `KRITZEL:` geschrieben. Der Befehl ist `BACKUP` mit den Optionen `LABEL=BCK001` und `BLOCK=65024`. Die Parameter sind `TAPE:GRAPHBACKUP` und `KRITZEL:*`. Der erste Parameter wird durch die Parameteroption `/SAVE` als Sicherungskopie gekennzeichnet, der zweite Parameter hat keine Optionen.

Eine wesentliche Erleichterung bei der Erstellung des *parsers* wird durch die interaktive Auswahl der Befehle bei menügeführten Systemen erreicht:

- Einschränkung der Befehlszahl,
- Einschränkung der Schalter für einzelne Befehle,
- Sinnvolle Vordefinitionen von Befehlen.

Einerseits können hier nur bestimmte Befehle zugelassen werden, andererseits kann der Aufbau der Befehle intern den Erfordernissen einer schnellen Parametererkennung angepaßt werden.

Die Erkennung der Befehle wird durch syntaktische Regeln beim Aufbau des gespeicherten Befehls ermöglicht. Die Erkennung der Parameter erfolgt durch *pattern matching*.

Der Parser ist vollständig in *Smalltalk* geschrieben und unabhängig von der zu analysierenden Syntax des jeweiligen Betriebssystems[21]. Da eine Befehlszeile mit dem Namen eines Befehls beginnt, ist es naheliegend, den Namen als Schlüssel in

[20]Unter VMS gibt es bei Befehlen positionsabhängige und positionsunabhängige Optionen. Positionsunabhängige Optionen wählen gezielt Eigenschaften des Befehls aus, positionsabhängige Optionen wählen Eigenschaften zu Parametern aus.

[21]In der Version 4.x von Smalltalk existiert eine Parser–Generator. Dieser Generator wurde in Verbindung mit der *ABS*–Definition auf Verwendung geprüft. Er eignete sich insbesondere zum Überprüfen der Definition der *ABS*–Beschreibung.

einem Wörterbuch zu verwenden. Der abgespeicherte Wert ist dann eine Liste von Parametern mit zusätzlichen syntaktischen Informationen. Zusätzlich wird die Liste der Änderungen spezifiziert. Die Analyse läuft dann wie in Abbildung 5.7 gezeigt ab. Die Funktionsweise des *parsers* wird im folgenden Abschnitt näher erläutert.

Ein Wörterbucheintrag mit Befehl und Parameterliste ist wie folgt aufgebaut: Die Spalte *Beschreibung* ist in der Tabelle nur zur Verdeutlichung enthalten, kann aber bei einer Realisierung durch einen Verweis in ein Hilfesystem ergänzt werden.

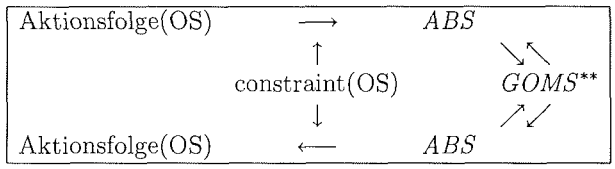
Schlüsselwort	Separator	Liste	Beschreibung
`ls`	–	l a	Dateiauflistung
`mv`	$-_{\sqcup}$		Datei umbenennen
`delete`	$/_{\sqcup}$		Datei löschen

5.3.2 Makros zwischen verschiedenen Systemen

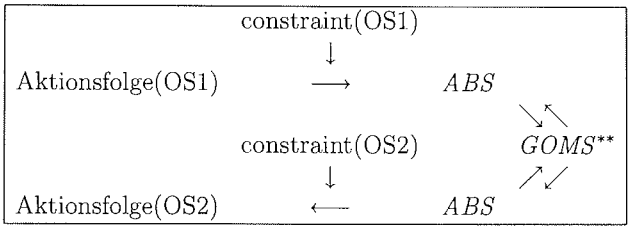
Ausgehend von den unterschiedlichen Makrodateien der Benutzer, `.COM`–Dateien unter `VMS`, *cshell–scripts* unter `UNIX`, wurde eine *GOMS***–Beschreibung der Handlungen des Benutzers angefertigt und analysiert. Diese Beschreibung liefert

- eine Liste der ausgeführten Aktionen,
- eine Liste der Parameter jeder Aktion,
- eine Liste der ausgewählten Optionen jeder Aktion,
- als Folgerung die unterschiedlichen Aktionen unter verschiedenen Umgebungen,
- Regeln, nach denen unter unterschiedlichen Bedingungen Aktionen und Parameter ausgewählt werden.

Mit dem Einsatz von Erkennungsregeln und Expansionsregeln lassen sich aus den Aktionen des Benutzers die abstrakten Beschreibungen, die dann in eine Wissensbasis eingetragen werden, gewinnen und umgekehrt aus den abstrakten Beschreibungen die konkreten Aktionen unter der Bedingung der dann gewählten Umgebung recompilieren. Die Aktionen werden in einer Aktions–Beschreibungs–Sprache notiert *(ABS)*. Dies kann wie folgt dargestellt werden:



Beim Umsetzen auf ein anderes System sind die Bedingungen des Zielsystems zu berücksichtigen.



In der Literatur gibt es weitere Methoden, um Handlungspläne zu beschreiben. Es seien hier erwähnt LEXITAS [HP89], DELTA [Sch89b] und *GOMS** [CMN83, Are89a]. *ABS* ist eine Vereinfachung der *GOMS***–Methodik. In *ABS* werden nur die Befehle notiert. Handlungspläne sind eine sequentielle Folge von *ABS*–Ausdrücken. *GOMS*** stellt auch das Wissen zu Aktionen und Objekten dar. *ABS* wird eingesetzt, wenn konzeptionelle Probleme, wie z. B. Parametererkennung, dargestellt werden müssen.

5.3.3 Makros und Pläne

Analyse und Synthese

Die folgende Darstellung beschreibt die Suche nach einer geeigneten Repräsentation der Befehlsfolgen.

Ein wesentlicher Punkt in der Betrachtung von Aktionsfolgen ist die Erkennung der semantischen Blöcke. Eine Wissenskomponente wird die Aktionsfolge aufnehmen und unter der Prämisse der aktuellen Umgebung analysieren.

Jede Aktion dieser Folge wird auf ihren Typ (Systembefehl oder Applikation) geprüft. Abhängig von diesem Ergebnis werden unterschiedliche Experten konsultiert. Die Wissensbasis *System* überprüft den Befehl auf syntaktische Korrektheit und teilt die Aktion in die Komponenten Aktionsbezeichner (A) und Parameterliste (PL) auf. Der Aktionsbezeichner wie auch die Parameter (P) können Optionslisten (POL, OL) besitzen.

Der syntaktische Aufbau kann Abbildung 5.8 auf Seite 120 entnommen werden. In dieser Abbildung wird die Zerlegung eines Kommandos in seine Bestandteile gezeigt. Diese Syntaxdiagramme liegen den folgenden Beispielen zugrunde. Das erste Beispiel zeigt die Zerlegung eines Kopierbefehls unter dem Betriebssystem VMS.

Aktion	Constraint	Zerlegung			Kommentar
`copy␣a␣b`	VMS	A	:	`copy`	Befehl
		PL	:	`(a b)`	Parameterliste
		P1	:	`a`	Parameter 1
		P2	:	`b`	Parameter 2
		POL	:	`(() ())`	Parameteroptionen
		OL	:	`()`	Befehlsoptionen

Im Beispiel ist zu erkennen, daß der Befehl keine Optionen hat, die Befehlsoptionsliste OL ist leer. Ebenso haben die beiden Parameter keine Optionen, die Parameteroptionsliste enthält nur leere Listen. Aus dieser Zerlegung wird die *ABS*–Beschreibung gewonnen.

MACRO kopiere_file von p1 nach p2
    copy_file <p1> <p2>

Die Methode *copy_file* ist bereits in der *ABS*–Beschreibung definiert, daher ist die Aktion vollständig beschrieben. Eine längere Aktionsfolge wird in eine Folge von *ABS*–Aktionen übersetzt.

Aktionsfolge	Bedeutung	*ABS–Notation*
`copy␣a␣b`	kopieren	copy_file <p1> <p2>
`copy␣c␣d`	kopieren	copy_file <p3> <p4>
`cc␣b␣d␣lib`	compilieren	compile_list <p2> <p4> <p5>
`r␣b`	ausführen	run_program <p2>

Diese Aktionsfolge kann als Makro in einer Wissensbasis unter einem Namen (z. B. `mtest`) abgelegt werden. Unter dem Namen `mtest` kann die Aktionsfolge wieder aufgerufen werden. Die Parameter werden dann abgefragt.

Ein Problem beim Eintragen der Folge in die Wissensbasis ist die Behandlung der Parameter. Der Benutzer muß die Möglichkeit haben, Parameter als konstant zu kennzeichnen. Diese Parameter werden dann bei einem erneuten Aufruf mit dem angegebenen Wert vorbesetzt. Durch explizites Überschreiben kann der Parameter jedoch neu gesetzt werden.

Weitere Probleme bereiten bei dieser Analyse die Optionen zu den einzelnen Befehlen. Nicht alle Optionen sind überall verfügbar. Aus diesem Grunde müssen die Optionen in einer weiteren Wissensbasis überprüft und mit einer Kennzeichnung versehen werden, die auf die Verwendbarkeit schließen läßt. (Vergleiche auch Kapitel 4.3.)

In Unterschied zu einem Makro, der eine feststehende Reihenfolge von parametrisierten Aktionen beschreibt, wird unter einem Plan ein Objekt verstanden, das auf von Benutzer eingegebene Aktionen reagieren kann. Ein Plan ist in der Lage, die eingegebene Aktion mit den Aktionen, die er enthält, zu vergleichen und gegebenenfalls eine Erkennung an den Benutzer zu melden. Ein Plan kann mehrere Makros repräsentieren. Dies hängt jedoch vom internen Aufbau eines Planes ab. Wir definieren einen Plan als technisches Objekt.

Definition 12 (Plan) *Ein Plan ist eine Repräsentation einer Aktionsfolge, die als Objekt betrachtet in der Lage ist, auf vom Benutzer eingegebene Aktionen zu reagieren.*

Ebenfalls problematisch ist die Erkennung einer Aktionsfolge in einem neue Plan. Dazu muß jeder Plan redundanzfrei sein, d. h., Aktionen, die den Systemzustand

nicht verändern, sind bereits entfernt. Auch werden diese Aktionen nicht zum *pattern matching* herangezogen, es sei denn, ein Plan enthält diese als letzte Aktion. Durch *pattern matching* kann dann die Übereinstimmung zwischen den abgelegten Aktionsfolgen und Teilfolgen im neuen Makro geprüft werden. Die Teilfolge wird dann durch den Namen der bekannten Folge ersetzt. Interessant ist, ob die Reihenfolge, in der die Aktionen eingegeben werden, für das Erkennen eines Plans relevant ist. Hier gibt es zwei Verfahrensweisen.

Die Reihenfolge der Aktionen ist relevant. Dann wird ein Plan nur erkannt, wenn der Benutzer die Aktionen in exakt dergleichen Reihenfolge nutzt, wie sie im Plan stehen.

Die Reihenfolge der Aktionen kann vertauscht werden. Dann muß ein Verfahren gefunden werden, daß diese Vertauschungsmöglichkeit innerhalb eines Plans repräsentiert.

Das erste Verfahren ist einfach zu realisieren. Die Erkennung einer Aktionsfolge, bei der Aktionen als nicht vertauschbar vorausgesetzt werden, ist einfach zu realisieren. Die eingegebene Aktion wird übersetzt, und deren Parameter werden innerhalb des zu überprüfenden Plans gemäß den Vorgaben des Plans instanziiert. Dies betrifft insbesondere die Belegung der Parameter. Stimmt die Aktion mit der gespeicherten Aktion überein, bleibt der Plan aktiv, ansonsten wird er als nicht erkannt eingestuft und zurückgesetzt.

Das zweite Verfahren ist in Abbildung 5.9 erläutert. Dabei wird vorausgesetzt, daß ein Plan (1) bestimmte Aktionen enthält und (2) als Übergang von einem Anfangs- in einen Endzustand betrachtet werden kann. Da Aktionen, die sich nicht gegenseitig beeinflussen, wie z. B. `copy a b` und `copy c d`, vertauscht werden können, ohne daß sich der Endzustand ändert, wird eine neue Repräsentation eines Plans gewählt. Vertauschbare Aktionen stehen in Ebenen, abhängige Aktionen stehen in Pfaden. Der Erkennungsmechanismus durchläuft folgende Schritte:

1. Erkenne Aktionen in einer Ebene.

2. Eine Ebene besteht aus allen im nächsten Schritt erreichbaren Aktionen.

3. Wenn eine Aktion eingegeben ist und nicht mit einer Aktion in einer Ebene übereinstimmt, dann wird der Plan nicht erkannt.

4. Bei der Planfortsetzung werden alle nicht erkannten Aktionen einer Ebene zuerst ausgeführt, dann wird der Plan vollständig beendet.

Die bekannte Folge sei, wie oben angegeben, unter dem Namen `mtest` verfügbar.

Die neue Folge ist:	Diese Folge kann dann ersetzt werden durch:
edit_file <p1>	edit_file <p1>
edit_file <p2>	edit_file <p2>
copy_file <p1> <p3>	mtest <p1> <p2> <p3> <p4> <p5>
copy_file <p2> <p4>	
compile_list <p1> <p3> <p5>	
run_program <p1>	
edit_file <p2>	edit_file <p2>
copy_file <p1> <p3>	mtest <p1> <p2> <p3> <p4> <p5>
copy_file <p2> <p4>	
compile_list <p1> <p3> <p5>	
run_program <p1>	

Die aufgetretenen Parameter sind bisher frei instanziierbar. Aktionen können jedoch auch auf Konstanten und Pseudo–Konstanten zurückgreifen. Diese Konstanten werden dann in den Makro mit aufgenommen und bei einem Aufruf verwendet. Eine gewisse Flexibilität sollte jedoch möglich sein. Daher sind Pseudo–Konstanten substitutionsfähig, d. h., ist bei einem Aufruf der Wert für den Parameter neu gesetzt, wird dieser neue Wert verwendet. Im bisherigen Beispiel kann die hinzugeladene Bibliothek als Pseudo–Konstante angesehen werden. Das einfache Beispiel ist dann:

copy_file <p1> <p2>
copy_file <p3> <p4>
compile_list <p2> <p4> <s5>
run_program <p2>

Definition 13 (Pseudo–Konstante)
Eine Pseudo–Konstante wird syntaktisch wie ein Parameter behandelt (laufende Nummer), jedoch durch eine andere Benennung (s) *gekennzeichnet. Der Wert der Pseudo–Konstanten ist durch die Belegung festgelegt, kann aber durch Überschreiben verändert werden.*

Sinnvoller als die Möglichkeit, den Wert der Pseudo–Konstanten zu überschreiben, ist die Festsetzung einer Konstanten, deren Wert nicht nur ein einfaches Objekt sein kann, sondern auch eine logische Beschreibung des Objekts. Wieder ist das Beispiel die hinzuzuladende Bibliothek. Wird die Bibliothek als Konstante angesehen und eingefügt, so wird bei einem Aufruf der Bibliotheksname eingesetzt, wie er gespeichert wurde. Wird die Bibliothek als logisches (semantisches) Objekt betrachtet, so wird bei einem Aufruf eine Übersetzung in den gültigen Namen vorgenommen. Da dieser zweite Fall mehr Flexibilität zeigt, insbesondere bei der Umsetzung in verschiedene Zielsysteme, wird er hier verwendet. Das Beispiel ist dann:

copy_file <p1> <p2>
copy_file <p3> <p4>
compile_list <p2> <p4> <c5>
run_program <p2>

Definition 14 (Konstante)
Eine Konstante wird syntaktisch wie ein Parameter behandelt (laufende Nummer), jedoch gekennzeichnet durch eine andere Benennung (c). *Der Wert einer Konstanten ist durch die Belegung festgelegt.*

Das vorliegende System registriert jede Aktion des Benutzers als Liste von Befehlen und Ergänzungen (Optionen). Dabei sind die Parameter durch die vom Benutzer eingesetzten Werte belegt. Der Makro–Parser muß nun aus den Aktionen die *ABS*–Beschreibung gewinnen. Ein Beispiel soll dies verdeutlichen.

Eingabe	Liste im System	*ABS*
copy a b	copy ! 'a' 'b'	copy_file <p1> <p2>
copy c d	copy ! 'c' 'd'	copy_file <p3> <p4>
cc b d lib	compile ! 'cc' 'b' 'd' 'lib'	ccompile_list <c5> <p2> <p4> <c6>
r b	execute ! 'b'	run_program <p2>

Im obigen Beispiel wird die Konstante als eine zusätzliche Bedingung für den Compiler benutzt. Die Programmiersprache ist durch die Angabe des *ABS*-Befehls ccompile_list bereits spezifiziert. Der Inhalt der Konstanten c5 gibt an, welcher Compiler genutzt werden muß[22]. Durch dieses Verfahren kann die Anzahl der *ABS*–Befehle eingeschränkt werden.

Die Rückübersetzung ist der umgekehrte Weg. Aus der *ABS*-Beschreibung wird unter der Prämisse des Zielsystems die korrekte Aktionsfolge gewonnen.

Um die Übersetzungen korrekt ausführen zu können, werden Compiler für jedes System, für jede Applikation nach *ABS* und umgekehrt benötigt. Weiter müssen diese Compiler von Expertensystemen unterstützt werden, die die Gültigkeit von Parametern und Optionen erkennen.

Die folgenden beiden Aktionen werden vom System völlig unterschiedlich behandelt, obwohl derselbe Befehls*name* verwendet wird:

Aktion	*ABS*
`cat a`	list_file <p1>
`cat a b > c`	append_file <p1> <p2>

Im Beispiel enthält der Parameter p1 eine Liste von Dateien, die an die Datei p2 angefügt werden sollen.

In Kapitel 4.1.2 wurde die neue Beschreibung der Benutzeraktionen mittels des *GOMS***–Modells vorgenommen. Bei der Implementierung der Aktionserkennung ist

[22]Unter UNIX existieren zumindest zwei verschieden C–Compiler, cc und gcc.

zu berücksichtigen, daß der Benutzer nur eine Aktion durchführt und das zugehörige Wissen nicht explizit ausdrückt. Daher reicht hier eine Betrachtung der durchgeführten Aktion allein aus. Ebenfalls ist für die durchführbaren Aktionen dem System das Wissen bereits bekannt. Bei der Ausführung des copy–Befehls ist bekannt, wie Ziel und Quelle angegeben werden, welche Parameter übergeben werden und welche Änderungen auftreten. Bei neu definierten Makros muß der Benutzer dieses Wissen angeben bzw. das System erkennt dieses Wissen automatisch[23].

Die vollständige Beschreibung der Aktionen erfolgt jedoch immer im *GOMS***.

Die beiden Darstellungen korrespondieren miteinander. Die in *ABS* ausgedrückte Aktion entspricht in *GOMS*** der Definitionszeile eines Plans *(eines Goals, einer Methode)* ohne die zugehörige Änderungsliste und Angabe der Datentypen.

ABS wurde insbesondere bei der Überführung von Aktionen innerhalb des *ResourcenManagers* benutzt, um die über Menüs selektierten Befehle zu notieren.

Bearbeitung von Makros

Zum Arbeiten mit Makros und Plänen gehört auch, daß sie modifiziert werden können. Dabei muß im hier beschriebenen System eine Aktionsfolge zunächst in einen Makro überführt werden. Die Begriffe *Makro* und *Plan* sind keine Synonyme. Ein Makro ist eine Liste von Aktionen. Ein Plan enhält die Aktionsfolge eines Makros, kann jedoch auch auf Aktionsfolgen reagieren, die den gleichen Effekt hervorrufen, deren Reihenfolge also vertauscht sein kann.

Liegt eine Aktionsfolge in der Dialoghistorie vor, so kann sie in einen Makro überführt werden. Bei diesem Transfer werden die bisher genannten Regeln angewendet. Der Makro steht dann als systeminternes Konstrukt zum Aufruf bereit. Bei einem Aufruf werden die Befehle in einer Dialogbox angezeigt und die Parameter abgefragt. Dies entspricht dem oben aufgeführten ersten Verfahren zur Parameterabfrage. Der Makro in dieser Form ist eine Aktionsfolge mit instanziierbaren Parametern.

[23]Bei der automatischen Erkennung ist zu überlegen, wie die folgenden Befehle erfaßt werden können:

1. `cat a b c d e f > g`
2. `cat a b c d e f h > g`

Hier ist das System gefordert, die Semantik des Befehls zu erkennen, d. h., daß der erste Parameter des *append_file*–Befehls eine Liste von Dateien repräsentiert. Der *append_file*–Befehl ist daher als Befehl mit zwei Parametern darzustellen, d. h.:

1. append_file <p1> <p2> ≡ append_file ((a b c d e f) (g))
2. append_file <p1> <p2> ≡ append_file ((a b c d e f g) (h))

Diese Darstellung wird im *GOMS*** verwendet. Die *ABS*–Beschreibung ist aus technischen Gründen einfacher gehalten, arbeitet mit der Auflistung aller Parameter und erkennt implizit durch den *ABS*–Befehl die Bedeutung der einzelnen Parameter.

Dieser Makro kann im System in einen Plan überführt werden. Dabei findet die Übersetzung in *GOMS*** bzw. einfacher in *ABS* statt. Zur Bearbeitung des Plans werden zunächst nur die Befehle gezeigt. In Abbildung 5.10 ist ein Plan in der internen Notation abgebildet. Der Plan umfaßt die Befehle `edit`, `edit`, `compile`, `link` mit Parametern. Daneben wird die Beschreibung der Aktion `link` dargestellt. Um den Bearbeitungsvorgang für den Anwender lesbar zu gestalten, wurde die *GOMS***–Definition in die Syntax von Smalltalk umgeformt. Hier nun ein kurzes Beispiel:

Der Befehl `link a b c` wird nach der Parametrisierung im Planeditor wie folgt dargestellt.

```
(Action  withCommand:     'link')
         withParameter:   '<p1>';
         withParameter:   '<p2>';
         withParameter:   '<p3>'
```

Dies entspricht der *GOMS***–Darstellung *USE_METHOD:* link ((p1 p2 p3) (p1)). Dabei ist berücksichtigt, daß der Name des ersten Parameters auch gleichzeitig der Name des Ausgabeparameters ist.

Der Benutzer kann nun den Plan modifizieren. Er kann Aktionen hinzufügen, löschen oder die Reihenfolge vertauschen.

Diskussion der Verfahren

Die soeben beschriebenen Ansätze, einen Plan zu verfolgen, münden in einen einfachen Erkennungsmechanismus, der mit geringem Aufwand implementiert werden kann. Jede Aktion und jede Handlungssequenz oder jeder Plan können über den Namen und die vorhandenen Parameter identifiziert werden. Beim Vergleich werden die Parameter mit den in der aktuellen Handlungssequenz vorgegebenen Werten instanziiert. Durch ein Überprüfen der so aufgebauten *pattern* erfolgt die Erkennung einer Aktion. Im Falle einer linear abgelegten Aktionsfolge, d. h., jede in der Folge vorkommende Aktion ist in der Reihenfolge ihres Erscheinens ohne Rücksicht auf Parallelisierbarkeit gespeichert, sind keine weiteren Regeln zu berücksichtigen. Die Folge wird auch nur bei exakter Übereinstimmung erkannt. Anders ist es hingegen bei einer angenommenen Parallelisierbarkeit der Aktionen, d. h., bei der Speicherung wird die Folge auf möglicherweise parallel ausführbare Aktionen überprüft. Hier müssen Regeln für Kommutativität von Aktionen und gegenseitige Beeinflussung aufgestellt werden. Zusammenfassend kann folgendes gesagt werden:

Pläne und Aktionen sind Objekte des *ResourcenManagementSystems*: Durch Verwendung des Polymorphismus im objektorientierten Konzept verstehen beide Objekte, Plan und Aktion, die Nachricht `paObjekt match: anAction`[24]. Dies bedeutet, daß eine Aktion `anAction` bezüglich einer anderen Aktion

[24] `paObjekt` steht sowohl für einen Plan als auch für eine Aktion. `paObjekt` ist der Empfänger einer sogenannten *keyword*–Nachricht mit einem Parameter `anAction`.

`anotherAction` durch Senden der Nachricht

```
anAction match: anotherAction
```

auf Übereinstimmung geprüft werden kann. Wird einem Plan `aPlan` die Nachricht

```
aPlan match: anAction
```

gesendet, wird die aktuelle Aktion `anAction` mit der Definition des Plans verglichen.

Hierarchische Schachtelung: Kann eine Aktion mit der Definition eines Plans, d. h. dem Planaufruf, zur Deckung gebracht werden, gilt der Plan als erkannt im Sinne einer einfachen Aktion. Trifft dies nicht zu, so wird die erste Aktion des Plans mit der eingegebenen Aktion überprüft. Der Plan bleibt solange aktiv, bis er entweder vollständig erkannt wurde oder eine Nichtübereinstimmung eintritt. Dieser Ansatz ermöglicht, auch geschachtelte Pläne zu erkennen, und ist in beiden Verfahren einsetzbar.

Parallelisierung: Um eine Aktionsfolge zu parallelisieren, müssen geeignete Regeln für die Abhängigkeit und Kommutativität angegeben werden. Nach Richter sind Aktionen parallelisierbar, wenn sie vertauschbar sind (notwendige Bedingung) und wenn sich Eingangsbereich der einen und Ausgangsbereich der anderen Aktion nicht beeinflussen (hinreichende Bedingung)[25] [Ric84]. Ein Verfahren, daß auf diese Regeln achtet, kann implementiert werden. Dabei muß für jede *bekannte* Aktion und jeden *bekannten* Plan von Benutzer oder Entwickler angegeben werden, welche Eingangsbereiche und welche Ausgangsbereiche existieren und ob Vertauschbarkeit vorliegt. Eine vollständige Betrachtung dieser Problematik, insbesondere der automatischen Erkennung dieser Bereiche, würde den Rahmen dieser Arbeit sprengen.

Beschränkungen *constraints*: Die wesentlichen Beschränkungen liegen zum einen in der strengen zeitlichen Sequentialisierung der Aktionen des ersten Verfahrens. Hier werden parallelisierbare Aktionsseequenzen des Anwenders nicht erkannt. Zu anderen wird das zweite Verfahren dadurch beschränkt, daß für Aktionen und Makros die Erstellung der Wissensbasis für Eingangs– und Ausgangsparameter und Kommutativität der Aktionen nur ungenügend automatisiert werden kann.

Unter Berücksichtigung dieser Fakten konnte ein einfaches Werkzeug implementiert werden, das, unter Verwendung von vordefinierten Regeln für Eingangs– und Ausgangsbereiche und die Festlegung der Kommutativität für einige Betriebssystembefehle, in der Lage ist, aus einer Handlungsfolge ein parallelisiertes Planobjekt zu generieren.

[25] Diese Betrachtung deckt sich mit den Ausführungen in Kapitel 5.2.2, die dort für die Anwendung auf Aktionen in einer Dialoghistorie verfeinert wurden.

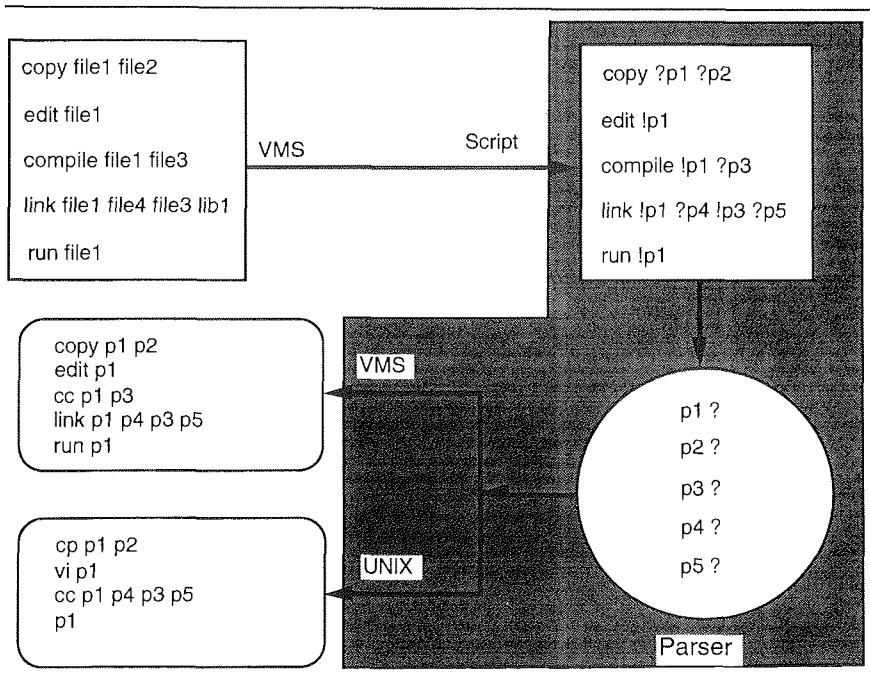


Abbildung 5.7: Der Ablauf des Makro–Parsings: Aus einer Befehlssequenz wird ein Meta–Makro erzeugt. Dieser Meta–Makro dient als Grundlage zur Umsetzung der Befehlssequenz in die Syntax des zugrundeliegenden Betriebssystems. In der Abbildung ist zum Vergleich mit der in der Arbeit gewählten Darstelung die Notation `?p1` für die Notwendigkeit der Abfrage des Parameters und `!p1` für die Ersetzung des Parameters benutzt worden. Bei den Ausführungen im Text wird deutlich, daß diese Darstellung nicht benötigt wird. Es gibt zwei Ansätze zur Vermeidung dieser Darstellung: (1) Die Parameter werden zu Beginn des Makros abgefragt und dann bei jedem Auftreten eingesetzt, oder (2) die Parameter werden beim ersten Auftreten innerhalb des Makros instanziiert.

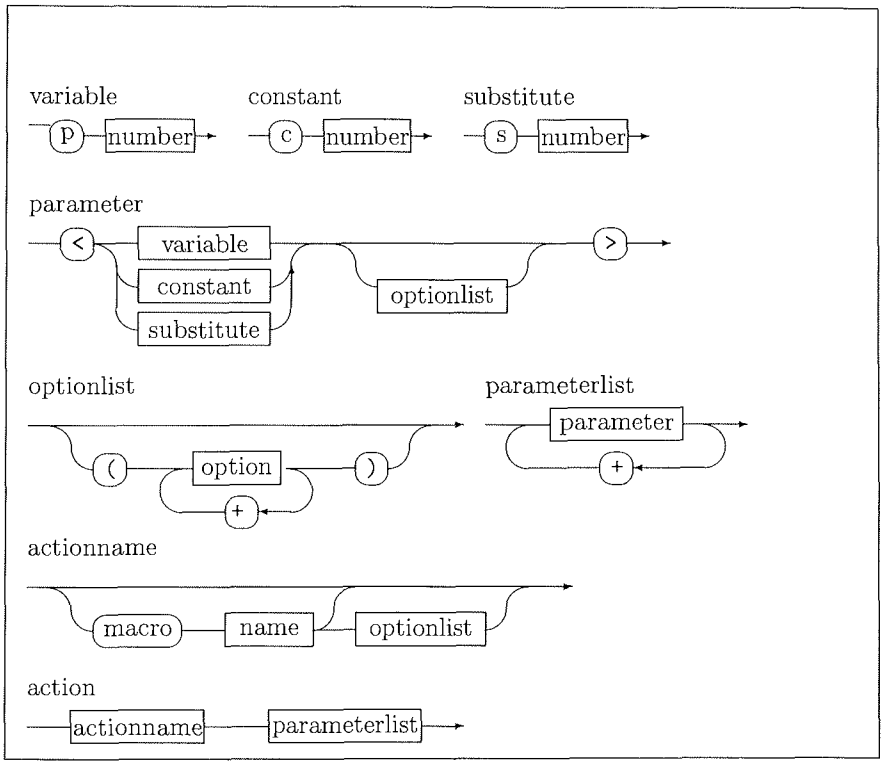


Abbildung 5.8: Die Abbildung zeigt die Syntaxdiagramme für die Zerlegung eines Kommandos in seine Bestandteile und in eine maschinenlesbare Form. Diese Form ist eine vereinfachte Darstellung der *GOMS***–Repräsentation zur Verdeutlichung konzeptioneller Aspekte, wie z. B. Parametererkennung.

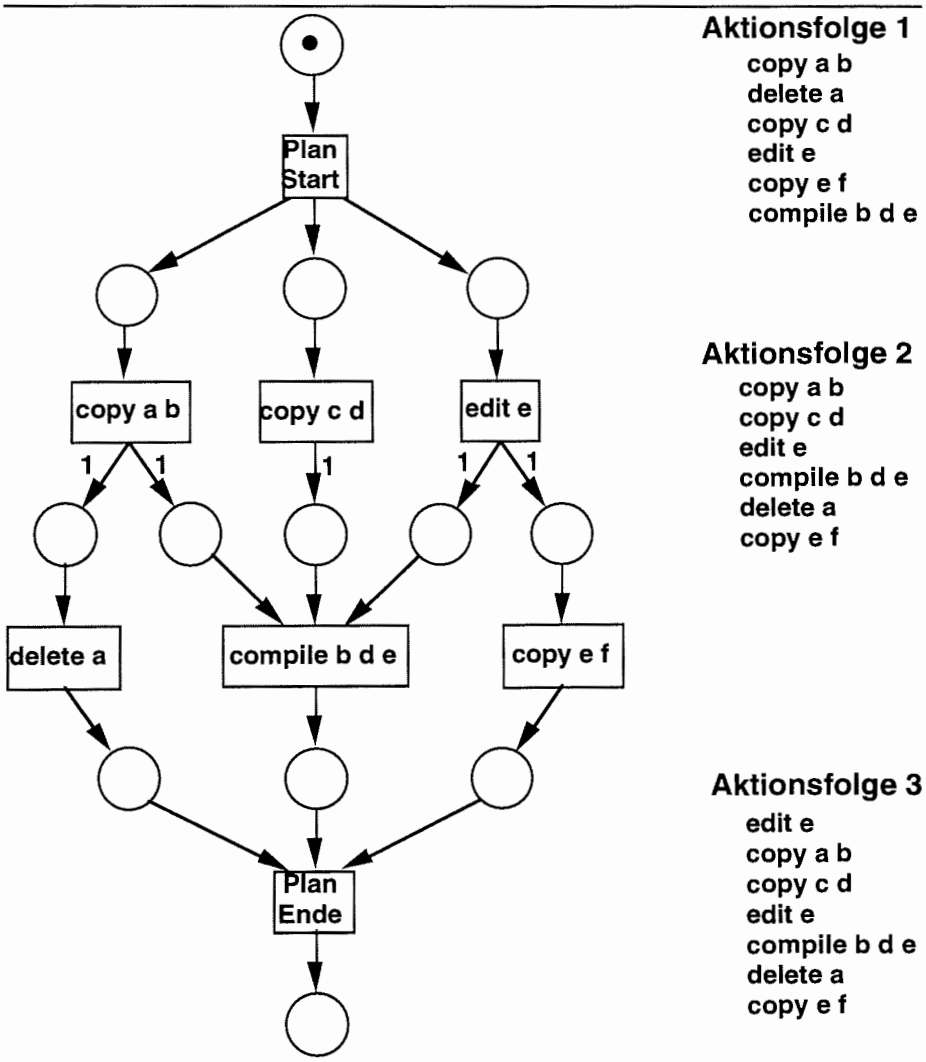


Abbildung 5.9: Die Abbildung zeigt den Graphen eines gespeicherten Plans als Transitionsnetz und daneben zwei mögliche Aktionsfolgen, die beide als Plan erkannt werden. Die Ebenen des gespeicherten Plans enthalten Aktionen, die voneinander unabhängig sind. Die Aktionsfolge 3 wird *nicht* erkannt.

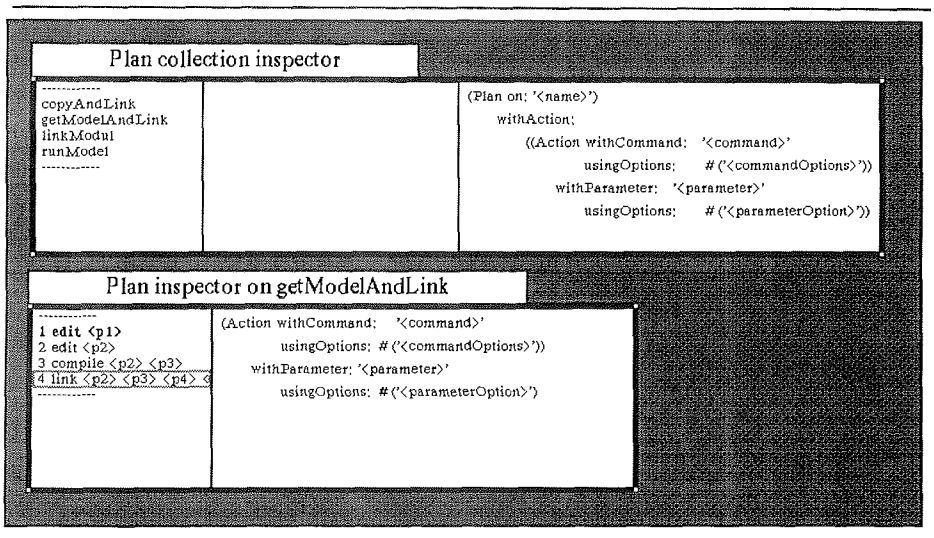


Abbildung 5.10: Die Bearbeitung eines Plans wird gezeigt. Das obere Fenster zeigt die verfügbaren Pläne im System. Es können Pläne hinzugefügt, verändert oder gelöscht werden. Ein Plan kann ausgewählt und in einem Editor bearbeitet werden. Dieser Editor ist im Fenster darunter zu erkennen. In den Textfenstern der *Browser* ist jeweils die Definition eines Plans (einer Aktion) gezeigt, wie sie vom Compiler verstanden wird. Im Planeditor können einzelne Aktionen hinzugefügt, modifiziert oder gelöscht werden.

... Der Hügel sinkt in die Ebene zurück, und Sie befinden sich wieder auf der Straße. Der nächste Schauplatz ist eine Wiese, auf der einige abstraktere Aspekte dieser Lerneinheit erörtert werden sollen. Dort haben sich schon ein halbes Dutzend exotischer Geschöpfe versammelt. Viele Menschen auf der ganzen Welt lesen dieses Buch, und dank der weltweiten Vernetzung können die Leser, die den Wunsch dazu verspüren, während der Lektüre miteinander in Verbindung treten. ...
Hans Moravec: Mind Children

Kapitel 6

Methodik der Realisierung

In den bisherigen Ausführungen haben wir ein Konzept zur Analyse und Erfassung eines wissenschaftlichen Arbeitsplatzes definiert. Ergänzend wurden Methoden zur Unterstützung des Benutzers eingeführt. Die Verfahren werden nun auf das reale Problem angewandt. Zunächst muß ein geeignetes System gefunden werden. Außerdem werden Werkzeuge zur Implementierung benötigt, die ebenfalls ausgewählt werden.

Aus den Erfahrungen, die bei der Betreuung der Wissenschaftler im AWI bei der Benutzung der Rechenanlagen gemacht wurden, ergaben sich für ein zu betrachtendes Zielsystem drei große Bereiche, die für den Einsatz der bisher entwickelten Konzepte sowie Methoden und Hilfsmittel geeignet erscheinen:

Erstellen einer Graphik: Der Benutzer möchte eine Graphik ausgeben. Dazu stehen ihm unterschiedliche Graphikterminals und verschiedene Drucker zur Verfügung. Außerdem soll die Möglichkeit bestehen, die Graphik aus mehreren Dateien aufzubauen und das Format der Graphik zu verändern. Der mittlerweile im Einsatz befindliche *MetaFile*–Handler unterstützt den Benutzer bei der Generierung von Graphiken auf diversen Ausgabegeräten.

Entwicklung eines numerischen Modells: Die Lösung eines Differentialgleichungssystems auf einem Rechner kann unter verschiedenen Prämissen erfol-

gen. Die Auswahl der unterschiedlichen Rechner geschieht nach Leistungsfähigkeit, Laufzeit, Anzahl der Variablen und der Größe des zu erwartenden Datensatzes. Bei der Bearbeitung von numerischen Modellen hilft sich der Benutzer bisher mit selbsterstellten Scripten.

Navigation im Netzwerk: Die Daten zu den numerischen Verfahren wie auch die Graphikdaten sind auf Dateiservern im Netz dezentral gespeichert. Der Benutzer benötigt effektive Verfahren, um auf die von ihm zu bearbeitenden Daten zugreifen zu können. Die Navigation im Netz erfolgt entweder über die Basisbefehle des Betriebssystems, mittels eigener Scripts, mittels spezieller Programme wie *FTP* oder, in besonderen Fällen, über netzweite Dateihierarchien z. B. *NFS*.

Neben der Auswahl der Anwendung ist auch eine Auswahl der Entwicklungsverfahren und Softwareanalysemethoden und der Entwicklungswerkzeuge zu treffen. Bei den Analyseverfahren kann u. a. mit strukturierter Analyse *(SA)* oder mit objektorientierter Analyse *(OOA)* vorgegangen werden. Die Verfahren sind unabhängig von den später zu wählenden Werkzeugen. Dabei sind die konzeptionellen Überlegungen aus Kapitel 4 zu berücksichtigen.

Die Werkzeuge, d. h. die eingesetzten Entwicklungsumgebungen, werden nach der Verfügbarkeit, der Problemorientierung und der voraussichtlichen Implementierungsdauer ausgesucht. Werkzeuge können die Analysemethodik unterstützen, wenn sie speziell dafür angepaßt sind[1].

6.1 Auswahl des zu realisierenden Zielsystems

In allen Arbeitsbereichen des AWI wird der Benutzer derzeit durch den Einsatz geeigneter Rechner (Apple MacIntosh) und anwendungsspezifischer Programme, die auf Datenbanksystemen basieren (SYBASE [Syb89], 4^{th}-Dimension [ACI89]), unterstützt. Insbesondere wird in einem Projekt im AWI durch konsequente Nutzung des Datenbanksystems und der dazu angebotenen Entwicklungswerkzeuge ein konsistentes Werkzeug für den Benutzer geschaffen (Nachfolger des *MetaFile*-Handler).

In den drei vorgestellten Bereichen modifiziert der Benutzer Objekte in seinem aktuellen Nutzungsbereich. Die Modifikationen werden mit den Betriebsmitteln des Rechners vorgenommen. Diese Betriebsmittel können in Ressourcen kategorisiert werden:

1. Betriebsmittel: z. B. Compiler, Editor,
2. Ausgabe: z. B. Bildschirm, Drucker,
3. Objekte: z. B. Datei, Directory.

[1] *Smalltalk-80* als objektorientierte Entwicklungsumgebung eignet sich besonders gut für die Umsetzung einer objektorientierten Analyse.

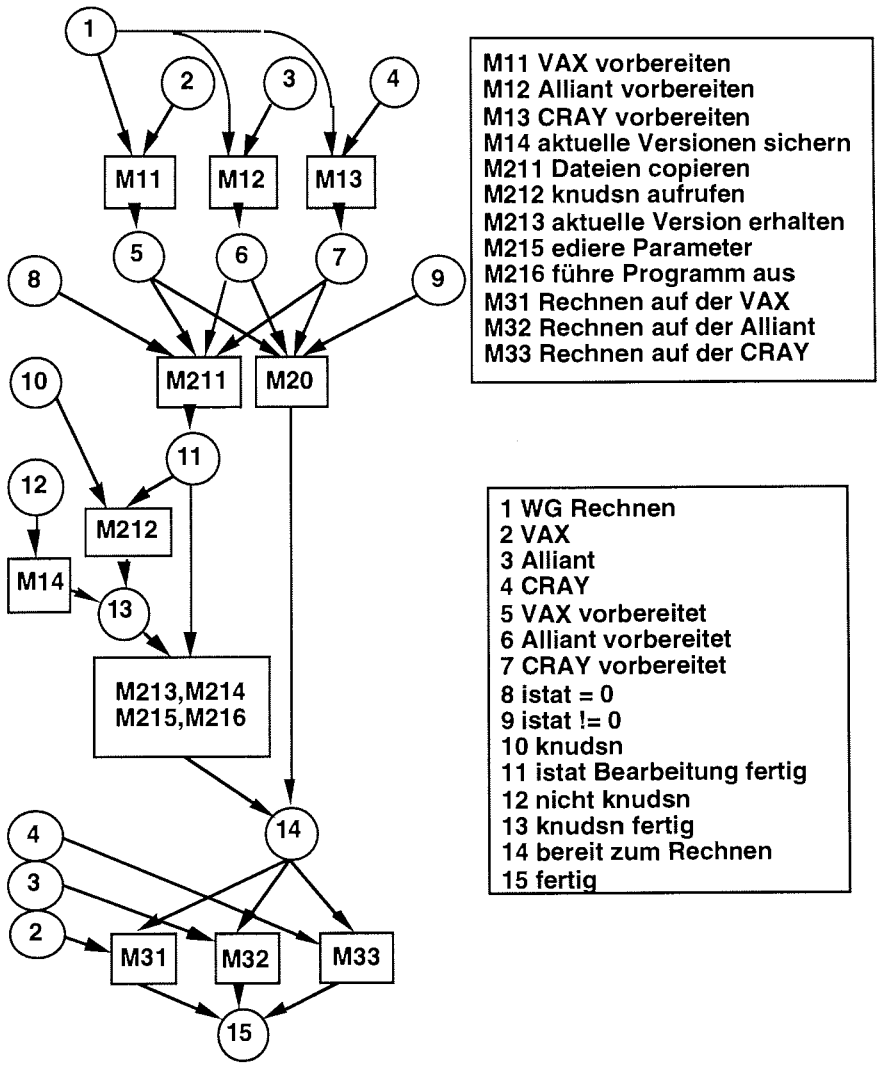


Abbildung 6.1: Die Abbildung zeigt den Ablauf einer Handlung eines Benutzers am Rechner bei der Durchführung eines Modellaufs als Zustandsdiagramm (Analyse einer `.COM`-Datei). Die Stellen 2, 3 und 4 sind aus Gründen der Übersichtlichkeit links unten wiederholt eingetragen. *istat* ist eine Variable, die vor dem Start des Programmlaufs gesetzt wird, *knudsn* ist ein spezielles Programm zur Datenvorbereitung.

Als Beispiel sei dazu der Zyklus für den Test eines numerischen Modells aufgeführt. In Abbildung 6.1 wird dargestellt, wie der Benutzer in diesem Fall am Rechner agiert. Er wählt sich eine Aufgabe aus, z. B. Rechnen auf der VAX (Diagrammpunkt 1, 2). Diese Aufgabe kann bedeuten, daß die Ressourcen frei gewählt werden können oder aber auch erzwungen sind[2]. Der *ResourcenManager* sollte daher in der Lage sein, anhand einer Aufgabe oder eines Faktes zu erkennen, welche Bedingungen eine Ressource erzwingen.

Mit den zunehmenden Anforderungen an die Rechenanlagen (Ressourcekategorie: *Objekte*) mit verteilten Rechnern und Ein–/Ausgabegeräten (Ressourcekategorie: *Ausgabe*) ist auch hier eine Unterstützung der Benutzer erforderlich. Produkte wie NFS und NCS (Ressourcekategorie: Betriebsmittel) stellen unter den im AWI genutzten Betriebssystemen VMS und UNIX Möglichkeiten zur einfachen Navigation im hierarchischen Filesystem der Rechner bereit. Die jeweilige benutzereigene Umgebung muß auf dessen Bedürfnisse eingestellt werden. Der Benutzer ist jedoch meist gefordert, diese Einstellungen selbst vorzunehmen. Dies erfordert wiederum Kenntnisse der jeweiligen Betriebssysteme. Einige, für alle Benutzer gleichermaßen gültige und sinnvolle, Einstellungen werden aber vom System vorgegeben.

Auch kann der Benutzer eine übersichtliche Darstellung seiner Daten verlangen. Diese Darstellung sollte unabhängig vom zugrundeliegenden Betriebssystem sein, um die Konsistenz der Oberfläche zu bewahren und ein ständiges Umdenken zu vermeiden. Die Folge wird eine vereinfachte Bedienung des gesamten Systems sein.

Um den Benutzer von diesen Anforderungen zu entlasten und ihm eine Unterstützung zu bieten, die sowohl dem Anfänger als auch dem Fortgeschrittenen gerecht wird, wird als Anwendung der bisher in der vorliegenden Arbeit dargestellten Konzepte ein *ResourcenManagementSystem* entwickelt.

Im folgenden werden nun die Anforderungen, gemäß den Ausführungen in Kapitel 4, an ein *ResourcenManagementSystem* im AWI in Form eines Pflichtenheftes dargestellt. Die Eingliederung einer wissensbasierten Benutzerschnittstelle erfolgt unter Berücksichtigung dieses Pflichtenheftes.

6.1.1 Anforderungen an das Zielsystem

Das Zielsystem muß zunächst in seiner *Ist*– und *Soll*–Konfiguration erfaßt werden. Dies geschieht nach Kapitel 4 mittels eines Pflichtenheftes. Dann werden die Ziel– und Kontrollgruppen bestimmt, die wesentlichen Anteil an der Entwicklung haben, indem sie helfen, das System zu verifizieren. Zusätzlich lassen sich aus den Analysen des Benutzerverhaltens aus diesen Gruppen die Stereotypen und Basisverhaltensregeln gewinnen.

[2]Wenn der Benutzer ein Programm (Ressourcekategorie: *Objekt*) schreiben möchte, hat er die Wahl zwischen verschiedenen Editoren (Ressourcekategorie: *Betriebsmittel*). Dann kann er auch die Programmiersprache (Ressourcekategorie: *Betriebsmittel*) wählen, sofern diese nicht bereits durch andere Bedingungen erzwungen wird. Ist das Programm aber erstellt, wird durch die Wahl der Programmiersprache die entsprechende Compiler–Ressource erzwungen, z. B. C–Code erzwingt die Anwendung des C–Compilers.

Das Pflichtenheft

Die Basisanforderungen an das Zielsystem werden gemäß den Betrachtungen aus Kapitel 4 in einem Pflichtenheft gesammelt. Die darin zusammengefaßten Angaben sind eine Sammlung der Benutzerwünsche, um deren Aufgabe optimal zu erfüllen. Hier werden nun auszugsweise wichtige Punkte aufgeführt:

Mußkriterien

- Die Benutzeroberfläche ist graphikorientiert.
- Der Benutzer wird durch ein Modell unterstützt. Dieses Modell enthält Informationen über grundlegende Handlungspläne, Zustandsvariablen und Regeln des Benutzers.
- Der Benutzer erhält passive und kontextbezogene Hilfe. Aktive Hilfe erscheint automatisch im Falle eines Bedienungsfehlers und bei vom System erkennbaren Fehlerzuständen auf dem Hintergrundrechner.
- Im Verlauf einer Sitzung am System erfolgt eine Protokollierung des Dialogs, um dem Benutzer einen Überblick über durchgeführte Aktionen zu geben (Voraussetzung für *UnDo/ReDo* und die Erstellung von Makros und zur Verfolgung von Handlungsplänen).
- *UnDo/ReDo* sind auf die ausgeführten Funktionen bezogen.

Wunschkriterien

- Die Einbindung eines tutoriellen Systems ist für die Einarbeitung in das System gedacht und im Bedarfsfalle als Erklärungkomponente einsetzbar.
- Die Aktionen des Anwenders werden durch das Konzept der Handlungsplanverfolgung mit automatischer Handlungsplanerstellung protokolliert und analysiert.
- Der Benutzer kann seine Dialogsprachen wählen.
- Es sind mehrere Fenster vorhanden, um auf verschiedenen Rechnern zu arbeiten, in denen der Zielrechner erkennbar ist.
- *cut/copy/paste* Funktionen für die Systemeditoren werden bereitgestellt.
- Die *UnDo/ReDo*–Funktionalität wird erweitert.
- Die Daten werden dezentral nach Inhalt und logischem Zusammenhang geordnet.

Die Zielgruppe(n)

Das *ResourcenManagementSystem* ist für alle Benutzer des AWI konzipiert. Wie bereits in Kapitel 4 erwähnt, konnten nicht alle Benutzer zur Mitarbeit herangezogen werden. Daher wurde der Personenkreis, der für den Test des Zielsystems in Frage kommt, eingeschränkt. Die Einschränkung erfolgte unter Berücksichtigung der Relevanz einer Einführung eines derartigen Werkzeuges, der Bereitschaft zur Mitarbeit und den Vorkenntnissen. Die Zielgruppen wie auch die Kontrollgruppen setzen sich aus den Wissenschaftlern der Fachbereiche Physik, Ozeanographie und Meteorologie mit einer offenen Architektur zur Einbindung der noch fehlenden Fachbereiche zusammen.

Stereotypen, Scripte und Handlungpläne

Stereotypen bei der Zielgruppe waren relativ einfach zu finden, da ein Modellierer einen festvorgegebenen Plan zur Bearbeitung und Prüfung eines numerischen Modells hat. Dieser Basisarbeitsplan ist festgelegt. Jedes numerische Modell ist jedoch einzigartig in seinem programmierten Aufbau. Hier gibt es ausschließlich individuelle Ausprägungen für die Arbeit mit einem realisierten Modell. Ein Stereotyp dafür ist in Abbildung 6.2 dargestellt. Dieses Stereotyp eignet sich jedoch nicht besonders für die Betrachtung der Arbeitsabläufe bei den verschiedenen Modellierern. Es wird daher darauf verzichtet, globale Stereotypen zu betrachten.

Im Gegensatz dazu stehen die lokalen Stereotypen der einzelnen Benutzer. Hier muß geprüft werden, ob tatsächlich ein Stereotyp vorliegt oder nur ein Handlungsplan[3]. Wegen der ständigen Modifikationen, die ein programmiertes Modell erfährt, müßte der Stereotyp ständig erweitert werden. Eine sich immer wiederholende Folge von Aktionen liegt also nicht vor. Daher wird auch darauf verzichtet, lokale Stereotypen zu betrachten.

Da globale und lokale Stereotypen nicht verwendbar sind, setzen wir hier Handlungspläne als Basis der Benutzeraktionen ein und können so die gerade erwähnten Schwierigkeiten überwinden. (Vergleiche hierzu Abbildung 2.2.)

Für Regeln gelten die gleichen Betrachtungen. Entweder es existieren sehr allgemeine Regeln, die dann aber die Benutzer in keiner Weise charakterisieren, oder sie sind so speziell, daß sie als Teil des individuellen Benutzerbildes im System abgelegt werden müssen. Zur Verdeutlichung sei hier auf die Abbildung 6.2 verwiesen, in der, abhängig von der Belegung einer Variablen im Script, verschiedene Bearbeitungspfade beschritten werden können.

6.1.2 Die Realisierungsidee

Aus den Anforderungen der Benutzer ergibt sich im ersten Ansatz für die Unterstützung bei der Navigation in einem Computernetzwerk ein System, das für den

[3]Handlungspläne unterliegen einer stärkeren Modifikation als Stereotypen.

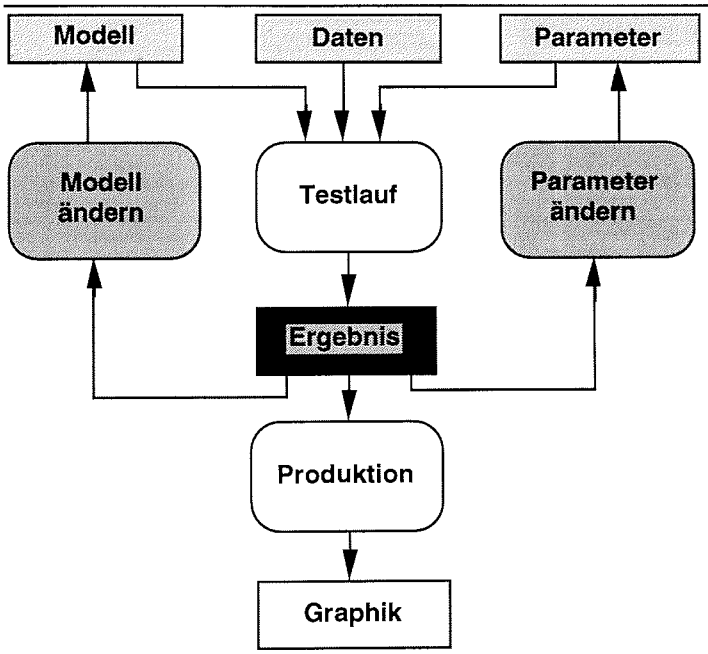


Abbildung 6.2: In der Abbildung wird ein Stereotyp eines Modellierers gezeigt. Dieses Stereotyp ist sehr allgemein gehalten und bedarf einiger genauerer Betrachtungen. (Siehe Text.) Entscheidend ist dabei, daß mit dieser allgemeinen Darstellung dieses Stereotyp nur von geringem Nutzen ist, da bei der Spezialisierung auf den einzelnen Nutzer spezifische Eigenheiten des Benutzers in so starkem Maße berücksichtigt werden müssen, daß ein stereotypes Verhalten der Benutzer nicht mehr angenommen werden kann.

Benutzer wie in Abbildung 6.3 erscheint. Die wesentlichen Verwaltungsobjekte werden gezeigt. Die Überlegungen führten zu der Forderung, die verschiedenen Funktionalitäten in verschiedenen Fenstern darzustellen. Die folgenden Funktionen werden aufgebaut:

- Netzwerknavigation
- Dateistrukturnavigation
- Ressourcennavigation
- Systemantwort
- Terminal (ähnlich DEC VT200)
- direkte Manipulation

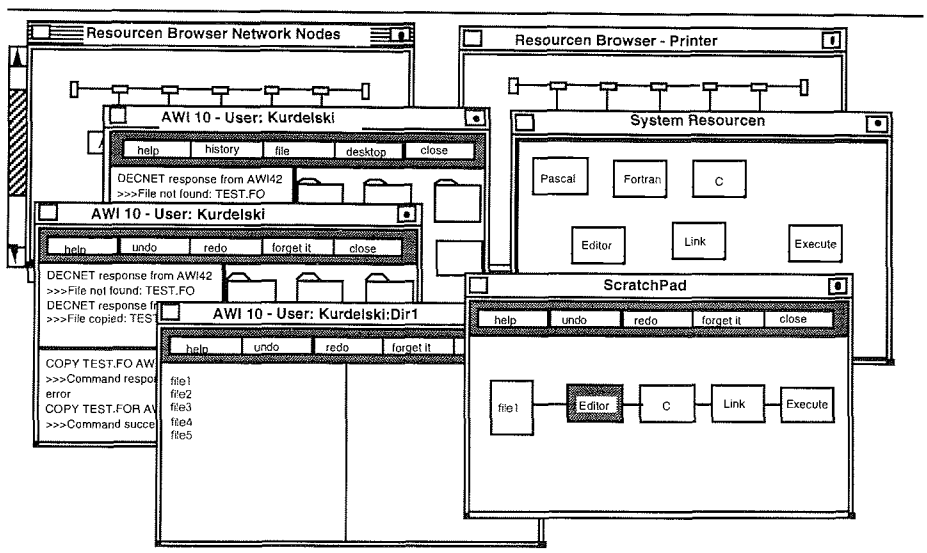


Abbildung 6.3: Die Abbildung zeigt, wie ein Dialog mit dem *ResourcenManager* aussehen könnte. Ein Fenster zeigt die Struktur des Netzwerkes (Fenster: `ResourcenBrowser Network Nodes`). In anderen Fenstern werden Benutzerdateihierarchien eines Rechnerknotens (Fenster: `AWI10 - User Kurdelski:Dir1`) und verfügbare Systemressourcen gezeigt. Der `ScratchPad` dient zum Aufbau komplexer Ablaufstrukturen, hier mit graphischer Unterstützung.

Der Wunsch sei, einen Rechnerknoten zu selektieren, ein Directory zu öffnen, eine Datei auszuwählen und Ressourcen mit der Datei zu verbinden. Zwei Ansätze sind hier zu unterscheiden:

1. externer Aufbau von Verbindungen (*Links*) und

2. Aufbau von Verbindungen auf einem Arbeitstableau.

Der erste Ansatz ist ressourceorientiert. Eine Ressource wird ausgewählt, eine Verbindung zu einer Datei wird aufgebaut und der Verarbeitungsvorgang gestartet. Die Handhabung ist vergleichbar der des *Xerox–Star* Systems [SIK+83], z. B. wird ein Dateiicon, das einen FORTRAN–Programmtext beschreibt, auf das Icon des FORTRAN–Compilers bewegt. Der Compilationsvorgang wird automatisch gestartet.

Der zweite Ansatz ist datenorientiert. Ausgehend von den ausgewählten Daten wird der Verarbeitungsvorgang auf einem Tableau spezifiziert. Der Benutzer erhält nun ein Datenflußdiagramm. Der Verarbeitungsvorgang wird auf Anforderung des

Benutzers gestartet. Dieses Verfahren entspricht dem *Fabrik*–Konzept, das von Ingalls et al. entwickelt wurde [IWC+88].

Ähnliche Verfahren sind auch in den Programmen, die einen interaktiven Aufbau einer Meßanordnung erlauben *LabView* [Nat] oder die eine spezielle Form des protypischen Programmierens vorsehen *ProGraph* [Pro], zu finden. Das Entwicklungspaket *aPe*, das zur Erstellung, Visualisierung und Aufbereitung von wissenschaftlichen, graphischen Darstellungen dient, ist in dieser Hinsicht sehr weit entwickelt, jedoch auf den einfachen Rechnern, wie z. B. SUN Sparcstation 1, zu langsam [Cen90].

6.2 Durchführung der Realisierung

Bei der Realisierung eines wissensbasierten Systems wird berücksichtigt, daß Komponenten zur intelligenten Unterstützung des Benutzers enthalten sind und diese auf zuvor erfaßten Regeln und Fakten basieren. Bedingt durch die Sichtweise des Entwicklers werden die Hilfsmittel ausgewählt, mit denen das System erstellt wird. Die Hilfsmittel umfassen im wesentlichen die Programmiersprache bzw. die Entwicklungsumgebung. Andere Hilfsmittel werden je nach Anwendungsgebiet hinzugenommen, z. B. Entwicklung in einer prozeduralen Programmiersprache mit Datenbankanbindung und Zugriff auf ein Expertensystem oder Entwicklung auf Basis einer Datenbank mit Zugriff auf Standardbibliotheken und ein Expertensystem. Im folgenden wird die Auswahl der Methodik und der Hilfsmittel erläutert.

6.2.1 Auswahl der Implementierungsmethode

In Kapitel 4 wurde ein Beschreibungskonzept entwickelt, das auf dem *GOMS***–Modell und *ATN/RFA*–Netzen basiert. Gemeinsam mit den Erkenntnissen aus der Literatur (z. B. [Her86a]) ist ein objektorientierter Implementierungsansatz vorzuziehen. Dieser Ansatz ist gekennzeichnet durch (Vergleiche auch [HA86].):

- Datenkapselung *(encapsulation)*,
- Nachrichten zwischen Objekten *(message passing)*,
- Vererbung *(inheritance)*,
- Polymorphismus *(polymorphism, operator overloading)*.

Die Auswahl einer reinen objektorientierten Entwicklungsmethode wird hier, im Vergleich zu anderen Methoden, begründet.

Prozeduraler Ansatz

Der prozedurale Ansatz (*GOMS**–Modell) läßt sich mit Programmiersprachen wie PASCAL oder C realisieren. Der Zugriff mit diesen Sprachen auf Datenbanken und

Expertensysteme ist gut realisierbar, da Bibliotheken verfügbar sind, mit denen der Zugriff ermöglicht wird.

Dieser Ansatz mußte jedoch verworfen werden, da sich die in Kapitel 4.3 definierten abstrakten Datentypen in diesen Sprachen nicht oder nur sehr schwer realisieren lassen. Das Versenden von Nachrichten, Vererbung und Polymorphismus ist nicht implementiert.

Funktionaler Ansatz

Der funktionale Ansatz läßt sich z. B. in `LISP` realisieren. Das *GTN*–Modell wurde bereits in seiner ursprünglichen Form in `LISP` implementiert [KP85]. Auch bietet `LISP` mit Erweiterungen gute Unterstützung des objektorientierten Ansatzes.

Die Entscheidung gegen `LISP` fiel einzig aus Gründen, die Verfügbarkeit, graphische Oberfläche und Wartung beinhalten. So ist z. B. *ObjTalk* der Universität Stuttgart ein Forschungssystem, das keiner Wartung unterliegt. Zudem stellen `LISP`–Maschinen, wie z. B. *Symbolics*, Insellösungen im Gesamtkonzept des AWI dar.

Logik–Ansatz

Die Entscheidung gegen `Prolog`, den Vertreter des logischen Ansatzes, fiel aufgrund der mangelhaften Unterstützung graphischer Benutzerschnittstellen und der fehlenden objektorientierten Erweiterungen.

Mittels dieses Ansatzes ließen sich sehr gut Faktendatenbanken und regelbasierte Entscheidungseinheiten realisieren.

Objektorientierter Ansatz

Aus den Analysen der Beschreibungsmittel und Modellansätze kann eindeutig ein Trend zur objektorientierten Beschreibung abgelesen werden. (Vergleiche [Her86a], [Obe88], [SH86].) Der objektorientierte Ansatz bietet Vorteile bei der Analyse (Betrachtung der agierenden Objekte) und Synthese (Kapselung von Daten und Methoden). Die Beschreibungen von systembezogenen Datentypen mittels abstrakter Datentypen lassen hier eine direkte Implementierung zu. Der Nachteil ist die bei den meisten einsetzbaren Werkzeugen fehlende komfortable Darstellung von Fakten und Regeln.

Entscheidung

Die Ablehnung der verschiedenen Ansätze bedeutet nicht, daß diese für das Gesamtsystem wertlos sind. Eine Wissensbasis, die auf der logischen Entscheidung von Fakten beruht, ist sicherlich flexibler in `Prolog` zu implementieren. Die Entscheidung für eine objektorientierte Implementierung wurde aus den Anforderungen des Benutzerschnittstellendesigns (Manipulation von Objekten) und der Analysemethode (*RFA*–Netze) abgeleitet. Dabei wurde die Arbeit von konzeptionellen Ansätzen im Da-

tenbankbereich [BCG+87], bei der Erstellung von Benutzungsoberflächen [BBG+89] und im Software–Engeneering [PW89, Coa89, SM88] beeinflußt.

Von Coad und Shlaer werden allgemeine Hinweise zum objektorientierten Programmentwurf vorgestellt [Coa89, SM88]. Mit diesen Konzepten sind Systeme im Hinblick auf verwendete Daten, Manipulationsmethoden und Zusammenhänge analysierbar.

6.2.2 Unterstützende Systeme im Vergleich

Die Implementierung des Zielsystems nach dem im AWI gesetzten *Haus*–Standard erfordert einen Vergleich zwischen existierenden *UIMS*'s und entsprechenden Softwarewerkzeugen. Viele dieser Systeme bauen auf demselben Betriebssytem (UNIX) auf oder benutzen einen virtuellen Prozessor *(Smalltalk-80)*: Die Ideen und Konzepte sind portierbar. In diesem Vergleich soll daher ein Leistungsstand bestehender Systeme aufgezeigt werden und die Einsatzfähigkeit des Systems für das zu entwickelnde *UIMS* geprüft werden. Die Systeme verfolgen alle grundsätzlich den objektorientierten Ansatz zur Erstellung der *Benutzungsoberfläche*.

Herczeg und Hoppe führten zwei ähnliche Vergleiche durch [Her86a, Hop88b], ebenso Balzert [BHJ87], deren Vergleichskriterien hier mit berücksichtigt werden. Herczeg betrachtet die ausgewählten Systeme unter dem Gesichtspunkt der softwareergonomischen und kognitionspsychologischen Zielvorstellungen und der, für den Benutzer sichtbaren, Systemfunktionalität. Hoppe untersucht die Eignung der Systeme auf ihre Anwendbarkeit in einer vom Benutzer geschaffenen Situation, z. B. Programmentwicklung in kleinerem und größerem Maßstab, Textverarbeitung, größere und kleinere Projekte, bei denen die Leistungsfähigkeit der Systeme (z. B. in *MIPS* gemessen) entscheidend ist. Einige der in den beiden Veröffentlichungen untersuchten Systeme sind für den Anwendungsfall im AWI von vornherein ausgeschlossen, da sie ein festes Aufgabengebiet umfassen (z. B. *Xerox–Star*). Die Kriterien und einige Ergebnisse können jedoch übernommen werden. Eine Ausweitung dieser Vergleichskriterien auf neuere Systeme, deren Auswahl sich auch an Normen und Verfügbarkeit orientiert und die die Anwendbarkeit auf die Situation im AWI gesondert berücksichtigen, wird vorgenommen. Weiter werden zusätzliche Auswahlpunkte eingeführt.

6.2.3 Vergleichsmethodik

Die zum Vergleich herangezogenen Systeme sind kommerzielle oder rein akademische Ansätze zur Lösung des Problems der benutzerfreundlichen, angepaßten Mensch–Maschine–Schnittstelle. Es finden sich in der Liste Systeme, die verwendbar und einsetzbar sind, die aber mit Einschränkungen zu sehen sind, z. B. sind sie nicht auf allen Rechnern lauffähig, oder ihre zukünftige Entwicklung ist ungewiß. Bei den gewählten Systemen wurde die Einsetzbarkeit als Entwicklungswerkzeug in den Vordergrund gestellt. Daher sind Systeme, wie die, die in der Arbeit von Herczeg

Tabelle 6.1: Fakten zur Auswahl einiger Benutzeroberflächen

- Ist das System verfügbar ?
- Wird Unterstützung geboten (Wartung, neue Versionen) ?
- Ist das System portierbar ?
- Ist es ein Standard oder nutzt es Standards?
- Werden Maschinen im AWI bei Investitionen in Erwägung gezogen, auf denen das System lauffähig ist ?
- Können die Konzepte portiert werden ?
- Ist der Einsatz von Expertensystemen mit dem System möglich ?

erwähnt wurden, nicht in die Auswahl genommen worden. Die Systeme wurden zusätzlich zu den von Herczeg und Hoppe aufgeführten Kriterien unter den in Tabelle 6.1 dargestellten Gesichtspunkten betrachtet. Die beschriebenen Systeme sind einerseits vollständige Entwicklungsumgebungen, teils Werkzeuge zur Entwicklung graphischer Oberflächen, die weitere Werkzeuge benötigen, um den Anforderungen einer adaptiven Schnittstelle gerecht zu werden. Wenn jedoch der Einsatz des Systems in den Vordergrund gestellt wird, dann ist es vernünftig, Systeme auszuwählen, mit denen eine einsetzbare Anwendung erstellt werden kann. Die Entscheidung muß nicht notwendigerweise für eine integrierte Entwicklungsumgebung fallen.

Bei der geplanten Realisierung mußte insbesondere die Verfügbarkeit der Systeme überprüft werden, die diese Kriterien erfüllen konnten, d. h., entweder die vorhandenen Ressourcen nutzen oder ein neues System wählen, das in die Umgebung eingepaßt werden kann. Eine Entwicklung auf einem nicht verträglichen System ist nicht anzustreben[4].

Die Tabelle 6.2 (Seite 135) zeigt eine Übersicht über die eingesetzten Systeme. Es ist zu erkennen, daß die meisten Systeme verfügbar sind. Ein System (*X-Aid*) ist noch in der Experimentalphase. Der *Presentation Manager* ist verfügbar[5]. Das einzige standardisierte System (*X-Windows*) wird im AWI zukünftig als Grundlage für die Erstellung von graphischen Oberflächen eingesetzt werden. In Tabelle 6.3 (Seite 136) werden weitere Fakten beschrieben, die die Auswahl unter dem Gesichtspunkt der Softwareentwicklung einordnen.

[4]Die Entwicklung ist heute einfacher, da viele Ansätze (X-Windows, NFS, NCS o. ä.) auch auf kleineren Arbeitsplatzrechnern verfügbar sind. Die Einbindung eines Systems fällt dann leichter.

[5]Der *Presentation Manager* wird als Teil des Betriebssystems OS/2 angeboten und ist kein eigenständiges Produkt mehr.

Tabelle 6.2: Übersicht über die Verfügbarkeit der Systeme

	Verf.	Unter.	port.	Stand.	Com.	Konz.	KI
Smalltalk-80	x	x	x	–	x	x	ie
Apple Macintosh II	x	x	–	–	x	x	e
X-Aid	–	–	–	–	–	x	ie
X–Windows	x	x	x	x	x	x	e
SUN NeWS	x	x	?	–	x	x	e
IBM/Microsoft:							
Presentation Manager	–	–	?	–	–	x	e
Apollo Domain Dialog	x	x	?	–	–	x	e
DEC Windows	x	x	–	?	x	x	e
New Wave	x	x	?	?	–	x	e
OSF Motif	x	x	x	x	–	x	e
Open Look	x	x	?	x	x	x	e

Erklärungen der Spaltenüberschriften

Verf.:	Das System ist zum Zeitpunkt der Analyse auf dem Markt erhältlich.
Unter.:	Das Produkt wird von Softwareherstellern unterstützt.
port.:	Das Produkt ist auf andere Rechner portierbar.
Stand.:	Das Produkt ist internationaler Standard (ISO etc.).
Com.:	Der entsprechende Rechner wird im AWI eingesetzt.
Konz.:	Die Konzepte sind übertragbar.
KI:	DIe Wissensbasis ist im System integriert (i), extern verfügbar (e).

Erklärungen der Spalteneinträge

x	Vorhanden.
–	Nicht vorhanden.
?	Unbekannt.
i e	Siehe KI.

In der Tabelle 6.4 (Seite 137) werden die Systeme im Hinblick auf ihre Einsatzmöglichkeit bei der Softwareentwicklung gegenübergestellt. Ein wesentliches Kriterium bei der Auswahl war auch die Einarbeitungszeit und die Zeit bis zum ersten Prototypen sowie Durchführung von Änderungen.

Im folgenden werden einige Systeme kurz beschrieben.

6.2.4 Beschreibung der Systeme

Smalltalk–80 und The Analyst

Smalltalk-80 ist eine objektorientierte Programmiersprache. Unter dem Begriff *Smalltalk* wird jedoch nicht nur die Sprache, sondern auch eine Programmierumgebung verstanden, die mit der Integration von Benutzerschnittstelle, Bibliotheksverwaltung und *source–level–debugger* andere Systeme maßgeblich beeinflußt hat.

Smalltalk-80 besticht durch die einfachen Methoden, eine graphische Benutzer-

Tabelle 6.3: Fakten zur Auswahl der einzusetzenden Entwicklungswerkzeuge im Hinblick auf Wartung, Verträglichkeit, Entwicklungszyklus und Unterstützung.

Entwicklungsumgebung: Die Entwicklung des Systems sollte innerhalb einer Entwicklungsumgebung erfolgen, um den Programmierzyklus klein zu halten, Übersichten zu erhalten und einfache Versionsverwaltung zu ermöglichen.

Wiederverwendbarer Programmtext: Die Bereitstellung gleichen Programmtextes für ähnliche Anwendungsfälle, wie z. B. bei der Vererbung in objektorientierten Sprachen, sollte unterstützt werden.

Graphikorientiert: Direkte Manipulation und visuelle, deutlichere Darstellung sind möglich.

Bibliotheken: Existiert bereits eine Bibliothek für Basisaufgaben wie Aufbau von Fenster, Dialogboxen, Betriebssystemdurchgriff, Schnittstellenanbindung?

Verträglichkeit: Ist das entwickelte System mit am Arbeitsplatz vorhandenen Systemen verträglich?

schnittstelle zu entwerfen. Das *MVC*–Konzept[6] ist eine leicht verständliche Vorgehensweise, um Daten zu manipulieren.

Smalltalk-80 unterstützt einfache Konzepte der KI wie *pattern matching*, Vererbung *(is–a, has–a)*, *Frames*. Baumstrukturen sind leicht realisierbar. Der Aufwand zur Generierung einer neuen Datenstruktur ist relativ gering, da durch den Vererbungsmechanismus meist wenige Änderungen erforderlich sind, um die gewünschten Effekte zu erzielen. (Siehe z. B. Abbildung 6.4, Seite 138.)

Neben *Smalltalk-80* müssen hier zwei Applikationen erwähnt werden, die besondere Eigenschaften zum *Smalltalk-80* bereitstellen. Die erste Anwendung, *The Analyst*, ist ein integriertes System mit Dokumentenbearbeitung, Tabellenkalkulation *(Spreadsheet)*, Dateiverwaltung, Landkartendarstellung und Landkartenbearbeitung und einer einfachen Datenbank.

Die Dateiverwaltung erfolgt über eine zentrale logische Einheit, das *Information Center*. In dieser Einheit werden die Informationen über Dokumente, Dateien, Graphiken etc. verwaltet. Jeder Benutzer erhält sein eigenes *Information Center*. Die Einordnung der Daten und die Suche nach den Daten wird durch die Vergabe von Schlüsseln erleichtert (Siehe Abbildung 6.5).

In der Tabellenkalkulation können beliebige Datenobjekte in die Zellen eingetra-

[6] *MVC–Konzept: (Model–View–Controller)*, *Model* = Datenstruktur, *View* = Sicht auf das *Model*, *Controller* = Steuerung der Benutzeraktionen (des Datenflusses). Für weitere Erläuterungen wird auf die Literatur verwiesen [GR83, Gol84, Xer89, Pie87, Dig86b, Dig88].

Tabelle 6.4: Übersicht über die Unterstützung des Entwicklungszyklusses

	Ent.	Wie.	Graph.	Stand.	Ver.
Smalltalk-80	x	x	x	x	x
Apple Macintosh II	x	2	x	x	x
X-Aid	?	x	x	–	–
X-Windows	–	2	x	3	x
SUN NeWS	–	2	x	3	?
IBM/Microsoft:	–	2	x	3	?
Presentation Manager	–	2	x	3	?
Apollo Domain Dialog	?	2	x	3	?
DEC Windows	x	2	x	3	x
New Wave	?	2	x	3	?
OSF Motif	1	2	x	3	?
Open Look	x	2	x	3	?

Erklärungen der Spaltenüberschriften

Ent.:	Das System besitzt eine komfortable Entwicklungsumgebung.
Wie.:	Das System läßt einen wiederverwendbaren Code zu.
Graph.:	Das System unterstützt Graphik.
Stand.:	Das System verfügt über komfortable, standardisierte Bibliotheken.
Ver.:	Das System ist mit der bestehenden Umgebung verträglich.

Erklärungen der Spalteneinträge

–	Nicht vorhanden.
?	Konnte nicht festgestellt werden bzw. ist von der eingesetzten Software abhängig.
x	Vorhanden.
1	Herstellerspezifische Entwicklungswerkzeuge.
2	Bedingt durch den Einsatz der Bibliotheken.
3	Umfangreiche Basisbibliotheken.

gen werden. Diese Objekte können manipuliert werden, und es können Berechnungen vorgenommen werden. Bei den Objekten sind beliebige, in *Smalltalk* repräsentierbare Objekte möglich, z. B. Graphiken, Datenbanken oder einfache Objekte wie z. B. Zahlen.

Das Dokumentensystem enthält einen DTP-Editor, mit dem eine Seite wahlfrei gestaltet werden kann. Dieses Dokument kann als Grundlage zum Suchen von Begriffen verwendet werden (Online-Hilfe) oder zu einem druckreifen Dokument umgesetzt werden.

Die zweite Anwendung, *Humble*, ist ein Expertensystem, das in weiten Teilen wie das bekannte Expertensystem MYCIN aufgebaut ist. Das System ist sehr gut in die *Smalltalk-80* Umgebung integriert, d. h., Wissensbasen können aus *Smalltalk* Methoden abgefragt und manipuliert werden.

Smalltalk-80 ist für alle gängigen Arbeitsplatzrechner erhältlich. Die Kompatibilität der Systeme auf Quelltextebene ist gut, da eine systemunabhängige Pro-

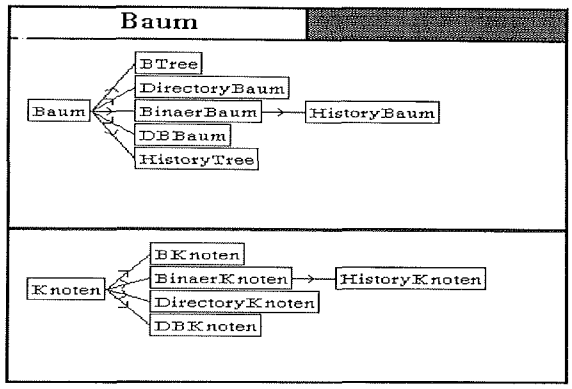


Abbildung 6.4: Die Abbildung zeigt die Hierarchie der Klassen *Baum* und *Knoten*, wie sie in *Smalltalk-80* realisiert werden können (Implementierung des Verfassers). Das Bild wurde im *The Analyst* mit der Klasse `ConnectedBoxView` erzeugt.

grammierung durch besondere Klassen und Methoden ermöglicht wird. Die binäre Kompatibilität der *image*–Dateien konnte im Rahmen der vorliegenden Arbeit nicht verifiziert werden. Nachteilig ist der immense Speicherplatzbedarf auf den lokalen Rechnern und das Laufzeitverhalten[7].

X–Aid, Babylon

X–Aid ist ein Projekt der GMD Birlinghofen zur Gewinnung eines adaptiven *UIMS*. Hierzu werden Wissensbasen benötigt, die vom System automatisch angepaßt werden können. Das System ist mit *UnDo/ReDo* ausgestattet und enthält einen Planerkenner zur Erfassung der Handlungspläne des Benutzers. Ebenfalls ist ein kontext–sensitives Hilfe–System integriert.

Das *UIMS* soll sich dem Benutzer individuell anpassen. Dies geschieht hier nicht durch Eintragen von Vorgaben z. B. eines Standardbenutzers. Das System soll mit dem Benutzer gemeinsam lernen. Weiterhin ist das System unabhängig von der zugrundeliegenden Applikation.

In Abbildung 6.6 ist der prinzipielle Aufbau des Systems abgebildet.

Im Vergleich zu den übrigen betrachteten Systemen ist *X–Aid* die am weitesten fortgeschrittene Arbeit und nicht nur ein Werkzeug. Der Nachteil von *X–Aid* ist, daß es bisher nicht verfügbar ist.

[7]Diese Einschränkung galt nur für die verwendete Version 2.3. Bei allen Nachfolgeversionen wurden sowohl das Laufzeitverhalten (durch zweistufige Compilierung) als auch die Einschränkungen im Objektadreßraum behoben.

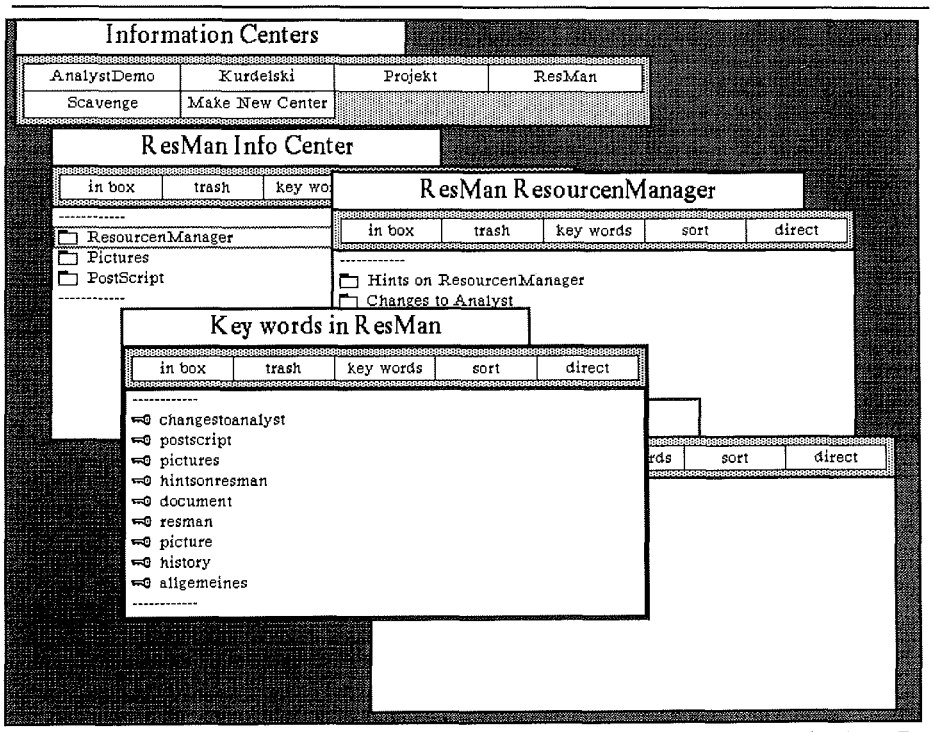


Abbildung 6.5: Die Abbildung zeigt ein *Information Center* des *The Analyst*. Der Anwender hat einige Ordner geöffnet. In dem mit der Überschrift *Key Words* gekennzeichneten Fenster sind die zur logischen Einordnung genutzten Schlüssel zu finden.

Ein weiteres System, *BABYLON*, ist bereits kommerziell erhältlich [CdV89]. Im Gegensatz zu *X-Aid* ist *BABYLON* als Expertensystem-Shell konzipiert, die auch in *X-Aid* eingesetzt wird.

Apple Macintosh

Der Apple Macintosh wird hier wegen der guten, standardisierten Benutzungsoberfläche erwähnt. Die Oberfläche wird durch die *Human Interface Guidelines* [App86] beschrieben. Der Anwendungsprogrammierer muß sich an diese Richtlinien halten, um ein funktionsfähiges Programm für den Apple Macintosh zu erhalten.

Die *Human Interface Guidelines* beschreiben jedoch nur die Hilfsmittel der direkten Kommunikation mit dem Benutzer [App86]. Dazu gehören Dialogboxen, *Property-Sheets*, *Prompter* für Text oder binäre Antworten. Durch die Anwendung der *Guidelines* ist der Umstieg von einem Programm auf ein weiteres in relativ kurzer Zeit möglich, da sich die Grundfunktionen (z. B. Datei öffnen, sichern, Objekt

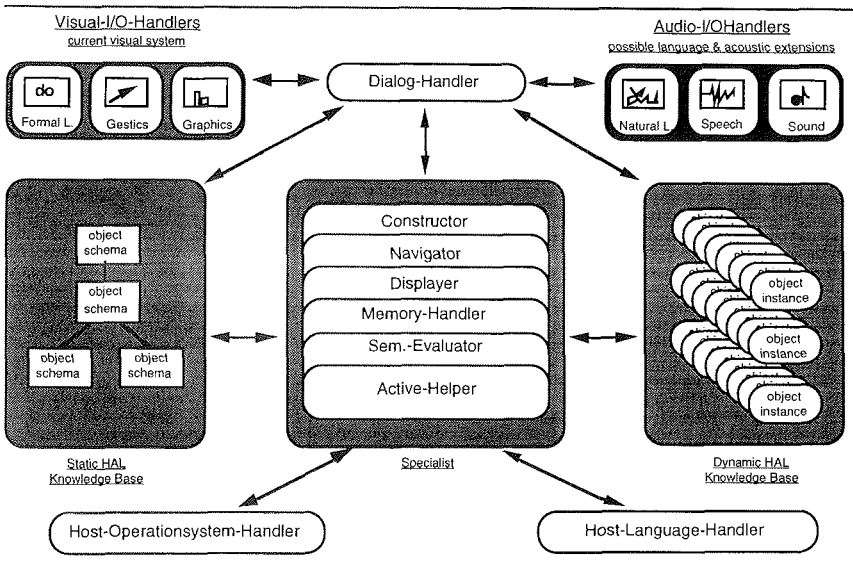


Abbildung 6.6: Das *UIMS X-Aid* mit seiner Struktur (aus [TKH86]

cut und *paste*) bei allen Programmen wiederholen.

Mit den Hilfsmitteln, die der Apple Macintosh bietet, ließe sich eine leicht erlernbare, intuitiv handhabbare Benutzungsoberfläche erstellen. Die Verbindung einer Benutzungsoberfläche mit den Applikationen des Apple Macintosh wird mit der *MacApp*-Bibliothek hergestellt.

Ein auf dem Apple Macintosh erstelltes *UIMS* benötigt ein Expertensystem und eine Datenbank. Die Schnittstellen zu diesen Systemen sind meist definiert und leicht zugänglich. Insbesondere ist der Zugriff auf Großrechner mit den verfügbaren Programmen einfach auszuführen.

Die wesentlichen Kritikpunkte an dieser Oberfläche sind die Verfügbarkeit des Systems allein auf dem Apple Macintosh und die fehlende Anpassung an andere Standards.

In jüngster Zeit sind im neuen System 7.x von Apple viele interessante Aspekte verwirklicht, wie z. B. dezentrale Speicherung von Daten und logische Ordnung der Daten durch Verbindungen *(links)*.

6.3 Analyse und Implementierung

Die Auswahl eines nach den bisher dargestellten Vorgaben realisierbaren Systems wurde sorgfältig nach anwendungsorientierten Gesichtspunkten durchgeführt. (Siehe

Kapitel 6.) Die *Modellierer*[8] benutzen das komplexe Netzwerk des Institutes, um ihre Daten immer auf dem für den Anwendungsfall notwendigen Rechner zur Verfügung zu halten. D. h., der Anwender muß bei der Navigation und den allgemeinen Aufgaben auf dem Rechner unterstützt werden. Das Zielsystem wird ein Navigationssystem für die im AWI vorhandene Rechnerkonfiguration sein, das insbesondere die Arbeit mit den Ressourcen erleichtern soll.

Die Realisierung eines komplexen Systems dieser Art erfolgt in mehreren Schritten, die auch durch das Verfahren des *rapid–prototyping* gestützt werden:

- Entwicklung des Systems innerhalb kleiner Grenzen in einer Sprache
- Überprüfung, welche Systemteile in der Sprache nicht implementiert werden sollen oder können
- Entwicklung von Systemschnittstellen
- Entwicklung der ausgelagerten Systemteile

Die Implementierung wurde mit dem *The Analyst V 3.0* unter *Smalltalk-80 V 2.3 Rel. 4* durchgeführt [Xer89, Xer87]. *Smalltalk-80* unterstützt eine *recovery/commit*–Funktion nach Archer, Conway und Schneider. Der Benutzer kann beim Verlassen von *Smalltalk-80* zwischen Sichern und Verlassen des Systems *(save&quit)*, Sichern *(save)* und nur Verlassen ohne Sicherung *(quit only)* wählen. Das Sichern entspricht dem *commit* nach Archer und das Verlassen ohne Sichern ist ein komplettes Wiederherstellen eines vorhergehenden internen Zustands von *Smalltalk-80 (recovery)*.

6.3.1 Objektorientierter Ansatz

In den bisherigen Betrachtungen der Modelle (Kap. 3.1) und Beschreibungsmethoden (Kap. 3.2) sowie den konzeptionellen Ansätzen, die zu Rahmensystemen führen [Her86a, Sch89a], wurde die Auswahl des objektorientierten Ansatzes (siehe auch Kap. 6.2) begründet. Die hier gewählte Programmierumgebung unterstützt diesen Ansatz. Daher sind im folgenden die benötigten Objekte, ihre Attribute, ihre Eigenschaften und die Beziehungen untereinander darzustellen. Die Analyse folgt im wesentlichen der Methodik von Coad [Coa89]. Dieser Ansatz hilft bei der Analyse des Systems und läßt genügend Spielraum für Techniken wie *rapid prototyping*. Die gewonnene Systemstruktur läßt sich direkt in den Programmtext übertragen. Die wesentlichen Beziehungen zwischen den Objekten werden dargestellt. Einzelne spezielle Beziehungen zwischen Objekten, wie z. B. die *MVC*–Beziehung, sind jedoch von der verwendeten Programmierumgebung abhängig und können daher nur konzeptuell erwähnt werden.

Der Benutzer arbeitet mit Programmen, Daten, Graphiken und Geräten. Aus diesem kurzen Satz können bereits wesentliche Objekte für das System abgeleitet

[8] Physiker, Ozeanographen, Meteorologen, die numerische Modelle zur Analyse von Vorgängen entwickeln.

Tabelle 6.5: Die Tabelle zeigt Objekte, mit denen der Benutzer arbeitet und wie sie implementiert werden können.

Objekt	Beschreibung	Smalltalk
Datei	Datenspeicherung	ja
Ordner	Ordungshilfsmittel	ja
Text	Programme, Texte	ja
Programm	Programmdatei	ja : FileStream
	ausführbar	ja : via SystemCall
Rechner	Darstellung	ja : Icon
	Darstellung des Inhalts	ja : SampleFileSystem

werden: (1) der Benutzer, (2) Programme, (3) Daten, (4) graphische Darstellungen und (5) Geräte. Es gibt einen Akteur *(actor)* und abhängige Objekte. Bei der Realisierung des Systems wurde davon ausgegangen, wie der Benutzer die Objekte einsetzt, d. h. welche Dinge (Objekte) er benutzt und wie er sie benutzt. Das Ergebnis dieser Betrachtung ist in Tabelle 6.5 gezeigt.

Einige Dinge sind dem Benutzer nicht zugänglich. Diese Objekte dienen der Strukturierung des Programms und verbergen systemspezifische Details. In Tabelle 6.6 sind die Objekte zusammengefaßt, die im System realisiert wurden.

Attribute

Attribute kennzeichnen die Eigenschaften eines Objekts. Zu diesen Eigenschaften zählen interne, extern zugängliche und systemspezifische Eigenschaften. Interne Eigenschaften (i) werden vom Objekt genutzt, um z. B. einen Zustand zu speichern. Extern zugängliche Eigenschaften (e) sind auch für den Anwender verfügbar, z. B. der Filename in einem *DirectoryBrowser*. Systemspezifische Eigenschaften (s) sind für die Koordination der Objekte untereinander relevant, z. B. der Verweis eines *TextEditors* auf seinen *DirectoryBrowser*, um Kommandos korrekt abzusetzen.

In Tabelle 6.7 werden einige Eigenschaften der obengenannten Objekte erläutert.

Methoden

Methoden geben dem Anwender die Möglichkeit, Eigenschaften der Objekte zu benutzen. Dem Objekt wird eine Nachricht gesendet, die die gewünschte Eigenschaft abfragt, als Ergebnis liefert oder ausführt. Der Anwender wird diese Eigenschaften über ein Menü selektieren. Das System wird die Aktion umsetzen.

In Tabelle 6.8 (Seite 147) werden einige Methoden gezeigt, die aktiviert werden können.

Tabelle 6.6: Die Tabelle zeigt die Objekte, die realisiert wurden, um die geforderte Funktionalität sicherzustellen.

Objekt	Beschreibung	Abkürzung
`ResourcenManager`	System zum Verwalten von Ressourcen	`RM`
`NetworkIconHBInternalState`	Netzwerkansicht der Rechnerkonfiguration	`NIHBIS`
`SampleFileSystem`	Bereitstellung einer Directory-Ansicht	`SFS`
`NoActionDocumentView`	spezielle Darstellung der Hilfstexte (*read-only*)	`NADV`
`HistorieBaum`	Repräsentant der Dialoghistorie	`HB`
`Papierkorb`	Retten gelöschter Daten nach UnDo oder *delete*	`PK`
`FabrikModel`	Unterstützung der interaktiven Planerstellung Netzwerkübersicht	`FM`
`Macro`	Scripterstellung/–parsing	`M`

Relationen

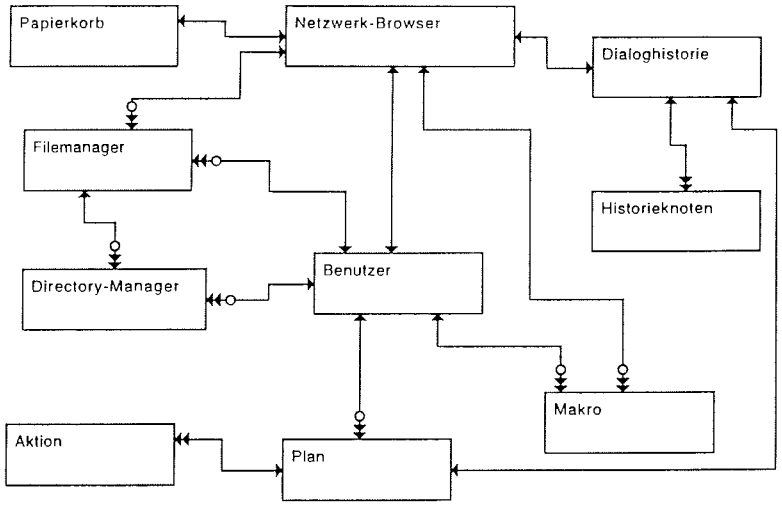
Relationen sind hier Objekte (Relationsobjekte), die Beziehungen zwischen anderen Objekten herstellen. Einfache Relationen zwischen zwei Objekten können ohne zusätzliche Relationsobjekte implementiert werden. Komplexe Relationen werden übersichtlicher unter Verwendung von Relationsobjekten. Nachrichten zur Veränderung werden nicht mehr an jedes einzelne Objekt abgesandt, sondern nur noch an die verbindende Relation.

Einfache Relationen können durch einen Verweis innerhalb des Objekts auf das abhängige Objekt realisiert werden. Bei komplexeren Relationen, insbesondere, wenn die Relation zusätzliche Attribute erhält, werden reine Relationsobjekte benötigt *(assoziative Objekte)*.

Eine wichtige Gruppe von Relationsobjekten sind die *dependents* im *model* einer *Smalltalk-80*–Implementierung. Das *MVC*–Konzept von *Smalltalk-80* benötigt eine Verbindung zwischen *model* (zu betrachtende Datenstruktur) und dem *view–controller*–Paar (*VC*–Paar). Es gibt zu jedem *VC*–Paar genau ein *model* aber zu einem Modell können mehrere *VC*–Paare gehören. In den *dependents* werden alle zu einem Modell gehörenden *VC*–Paare zusammengefaßt. Die *dependents* repräsentieren die *1:n*–Beziehung zwischen dem *model* und den *VC*–Paaren.

Die Beziehungen der Objekte sind in Abbildung 6.7, der Datenfluß innerhalb des *ResourcenManagementSystems* ist in Abbildung 6.8 dargestellt.

Objektstruktur des *ResourcenManagementSystems*



Name	Realisiertes Objekt	Funktion
Netzwerk-Browser	NetworkIconHBInternalState	Zugriff auf einzelne Rechner im Netzwerk
Filemanager	IconHBInternalState	Datenhierarchie, Status, Terminal
DirectoryManager	HBSampleFileSystem	Dateibearbeitung, Benutzeraktionen
Dialoghistorie	HistoryBaum	Mitschrift der Benutzeraktionen
Historieknoten	HistoryKnoten	Repräsentant einer Benutzeraktion
Benutzer	User	Angaben über den Benutzer
Makro	Macro	Parametrisierte Handlungssequenz
Plan	Plan	Handlungsplan
Aktion	Action	Einzelaktion

Abbildung 6.7: Die Abbildung zeigt die Beziehungen der Objekte im *ResourcenManagementSystem*. Die Darstellung erfolgt nach Chen/Bachmann.

Realisierter Datenfluß im *ResourcenManagementSystem*

Systemobjekte

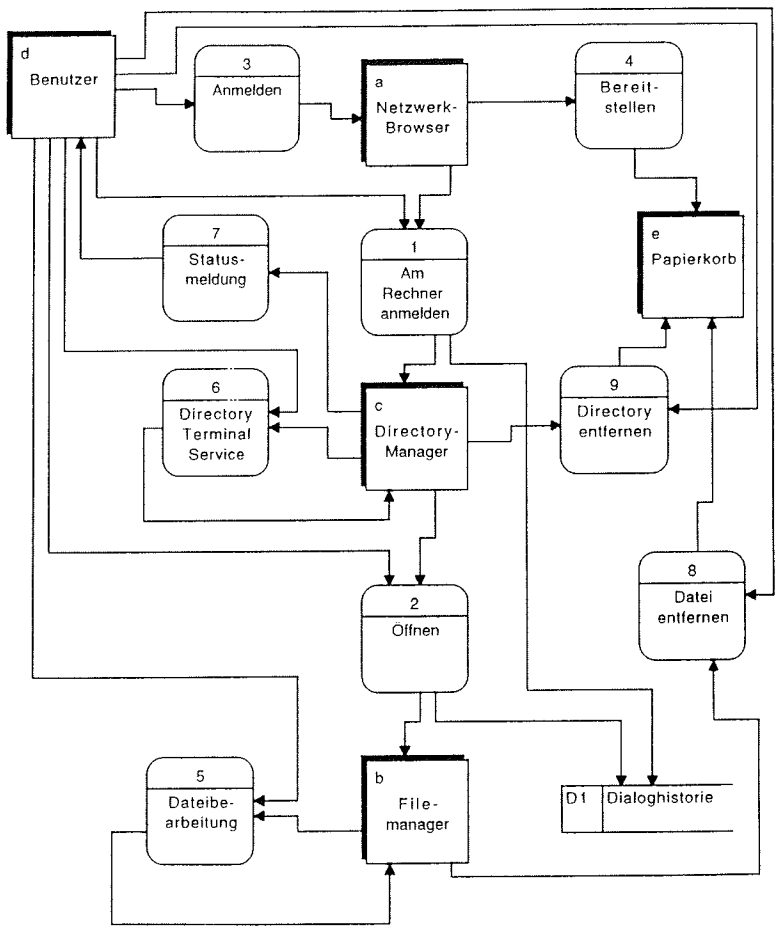


Abbildung 6.8: Die Abbildung zeigt den Datenfluß innerhalb des *ResourcenManagementSystem* (nach Gane/Sarson).

Tabelle 6.7: In der Tabelle werden einige realisierte Objekte und ihre Eigenschaften beschrieben.

Objekt	Attribut	Beschreibung	Eigenschaften
RM	ParticipantUsers	Nutzer des ResMan (name, password)	is
	HelpDictionary	Verwaltung der Hilfs–Texte	s
NIHBIS	historyTree	Zugeordnete Dialoghistorie	es
	deletedFileList	zugeordneter Papierkorb	es
	selection	ausgewählter Rechnerknoten	e
SFS	historyTree	zugeordnete Dialoghistorie	s
HB	datum	vom Benutzer ausgeführtes Kommando	e
PK	list	Liste der gelöschten Dateien	i
FM	icons	Liste der verfügbaren Auswahlmöglichkeiten	i
	selections	Liste der ausgewählten Objekte	es
M	ResManMacroDictionary	verfügbare Macros	es

s systemspezifische Eigenschaft
e externzugängliche Eigenschaft
i interne Eigenschaft

Tabelle 6.8: In der Tabelle werden einige Beispiele implementierter Methoden der realisierten Objekte gezeigt.

Objekt	Methode	Beschreibung
RM	open	Öffnen eines Netzwerk–Fensters
	help: aSymbol	Anzeigen der Hilfe zu einem Symbol
NIHBIS	openVolume	einen Browser auf einen Rechnerknoten öffnen
	history	Anzeigen der zugeordneten Dialoghistorie
	doThis: aSymbol	Aktion aus einem Macro ausführen
SFS	createFile	Datei erzeugen
	deleteFile	Datei löschen
HB	add: einKnoten	einen neuen Knoten anfügen
	pfadFrom: k1 to: k2	Pfad in der Historie auswählen
	again	Sequenz wiederholen
	undo	Sequenz zurücknehmen
PK	undelete	Löschen von Dateien zurücknehmen
M	macro: pfad	Macro aus einer Handlungssequenz erzeugen
	again: pfad	Handlungssequenz aus *pfad* zur Ausführung aufbereiten
	show	Macro anzeigen

6.3.2 Aufbau der Wissensbasen

Bereits in Kapitel 5 wurden die notwendigen Experten eines benutzerunterstützenden Systems beschrieben. Hier sollen nun die Realisierungen der Experten explizit dargestellt werden.

Benutzungsmodell

Das Benutzungsmodell ist die Zusammenfassung von Aktionsregeln und Handlungsplänen des Benutzers. Im Idealfall sind diese Wissensbasen durch explizites Eingreifen modifizierbar. Aber schon die Fähigkeit des Systems, sich bei der Hilfe auf den Benutzer anzupassen, erfordert dynamische Analysen des Benutzerverhaltens.

Handlungspläne werden zur Zeit nur über die Dialoghistorie erfaßt. Aktionsregeln der Benutzer, die mittels des *GOMS***–Modells gewonnen wurden, finden sich teilweise im Aufbau, teilweise in bestimmten Abläufen der Benutzerschnittstelle wieder.

Eine Sammlung von Makros kann dem Benutzer bereits über den Plan mitgegeben werden. Hier ist ein leistungsfähiges Hilfsmittel zur Spezifikation und Modifikation von Aktionen und Aktionsfolgen entstanden. Aktionen und/oder Aktionsfolgen können in einer systemunabhängigen Notation eingegeben werden. Den Aktionen liegen die Ausführungsregeln des Betriebssystems zugrunde, an das die Aktionsfolge gesandt wird.

Aus der Dialoghistorie werden Pläne aus programmtechnischen Gründen noch in einer Vorform der *ABS*–Beschreibung gewonnen. Die Pläne werden aber in der *ABS*–Darstellung von einem Planeditor dargestellt und können vom Benutzer modifiziert werden. Diesen Plänen wird bei der Aktionsanalyse die jeweils aktuelle Aktion geschickt. Jeder Plan prüft die aktive Aktion gegen die aktuelle Aktion und kann schrittweise fortschreiten. Die Planerkennung bricht bei Nicht–Übereinstimmung sofort ab. Die Pläne können mit *Triggern* versehen werden, die Aktivitäten auslösen, wenn der Plan erkannt oder die Erkennung abgebrochen wurde.

Diese Tatsache kann insbesondere dann ausgenutzt werden, wenn zu abgespeicherten suboptimalen Plänen optimierte Verfahren bekannt sind. Bei vollständiger Erkennung des suboptimalen Plans zeigt der *Trigger* dem Benutzer den optimalen Plan.

Vorausschau ist nur durch den Benutzer zu aktivieren. Der Benutzer kann sich die gerade aktiven Pläne anzeigen lassen und dann einen zur weiteren Bearbeitung auswählen. In Abbildung 6.9 ist der Datenfluß innerhalb der Planerkennung des *ResourcenManagementSystems* dargestellt.

Einige der vom Benutzer angewandten Regeln sind in der folgenden Übersicht zusammengestellt. Diese Angaben sind detailliert in der Analyse im Anhang zu finden (Anhang A.2).

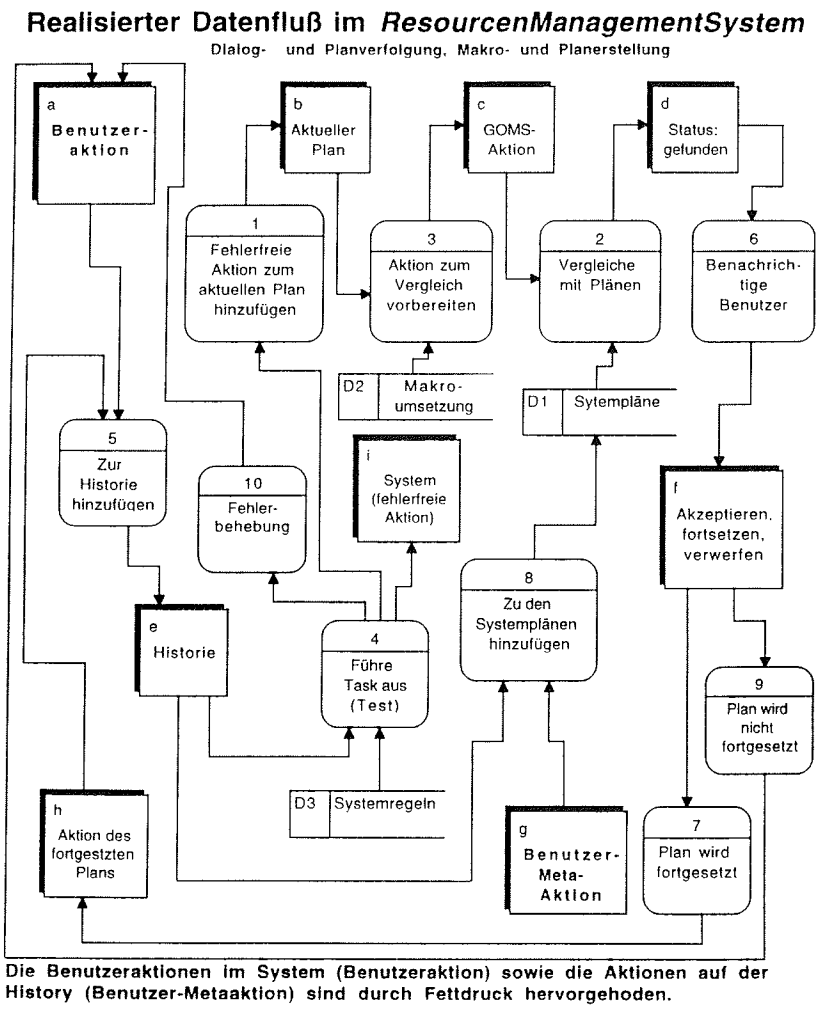


Abbildung 6.9: Die Abbildung zeigt den Datenfluß bei der Behandlung von Handlungsplänen. Drei Wege sind innerhalb der Darstellung zu erkennen. Zum ersten die Erfassung einer Aktionsfolge, zum zweiten die Erkennung einer Folge und zum dritten die Auswahl und Fortsetzung eines begonnenen und erkannten Plans.

- Auswahl eines Rechners bezogen auf ein Modell
  - Edieren auf einem langsameren Rechner
  - Compilieren auf dem langsameren Rechner zum Testlauf unter Berücksichtigung der Kompatibilität
  - Compilieren auf dem Zielrechner zum Modellauf
- Auswahl eines Rechners bezogen auf die graphischen Möglichkeiten
  - interaktives Betrachten der errechneten Daten
  - Darstellungsmöglichkeiten der Daten mittels Software
- Auswahl eines Verfahrens bezogen auf Modell und Rechner
  - Modelle kleiner Auflösung
  - Modelle großer Auflösung
  - Modelle mit großen realen Datenmengen
- Auswahl eines Ausgabegerätes
  - Graphik
    * Farbgraphik
    * Präsentation auf Folie
    * Auflösung
    * Größe der Darstellung (z. B. Plakat)
  - Text
    * Veröffentlichung
    * Programme
    * Test

Die Auswahl von Ressourcen für ein Modell bleibt über einen längeren Zeitraum konstant. Daher wird zu einem Modell eine Beschreibung gespeichert, die die Daten über die eingesetzten Ressourcen enthält. (Siehe Abbildung 6.10.) Diese und andere entsprechen einem *Repository* einer logischen Datenbank, in dem die Informationen über Objekte des Systems abgelegt werden.

Systemselbstbild

Das Systemselbstbild besteht aus Informationen des Systems über sich selbst. Aus diesem Wissen kann im jeweiligen Kontext Hilfe angeboten werden, die den Benutzer kontextabhängig unterstützt. Auch können Fehler erkannt und behoben werden. Der einfachere Fall ist eine statische, kontextbezogene Hilfe.

```
'D:\tex.dir\kurdelsk\prom'
    'prom.tex'
        'LATEX' 'GR'
        '-C=1' '-LA=OFF' '' ''
'd:\tex.dir\kurdelsk\ak'
    'ak.tex'
        'LATEX' 'GR'
        '-C=1' '-LA=OFF' '-S=1:2' ''
'D:\tex.dir\kurdelsk\ak'
    'ak.tex'
        'LATEX' 'GR'
        '-C=1' '-LA=OFF' '' ''
```

Abbildung 6.10: Ein Ausschnitt aus dem *File–Information–Dictionary*. Ein File wird durch Namen und Directory identifiziert. Die abgelegte Information ist in TEX–Daten und Druckerdaten aufgeteilt. Die hier abgelegte Informationen gehen weit über die Systeminformationen hinaus.

Zu einem Dialogzustand, z. B. ein aktives Fenster, kann der Benutzer Hilfe, bezogen auf den Zustand (das Fenster oder den Inhalt des Fensters), erhalten. Aus der Hilfsfunktion kann der Benutzer detailliertere oder allgemeinere Hilfe abfragen. Der Benutzer ist jedoch selbst zu einer Aktion aufgefordert.

Unter anderem zählt hierzu auch das Wissen über verfügbare Kommandos bzw. die Umsetzung der einzelnen Kommandos auf verschiedene Betriebssysteme.

Das Wissen ist in Form von Regeln innerhalb von Methoden und Wörterbüchern abgelegt. Veränderungen werden nicht vom Benutzer vorgenommen. Insbesondere sind hier die Compiler zu nennen, die eine Aktion von der Syntax des einen Betriebssystems in den korrekten Befehl des weiteren Betriebssystems überführen. Hier geht das statische Wissen über die Semantik der einzelnen Befehle ein.

In der Zusammenstellung ist noch einmal ausschnittsweise das benötigte Wissen gezeigt:

- Kennzeichnung der Fehlersituationen
  - Passive Hilfe aus dem Kontext
  - Hilfe im Fehlerfall
- Systemspezifisches Wissen
  - Syntax der Befehle im jeweiligen Zielsystem
  - Semantik der Befehle bei der Übersetzung
  - Gültige Operationen

Tabelle 6.9: Beispiele eines Kommandos mit Standardvorgabe. *Ja* in der Spalte *Vorgabe* bedeutet, daß die Option eine Standardvorgabe ist.

Aufruf	Beschreibung	Vorgabe
`fxc name.c`	Programm übersetzen und binden	ja
`fxc name.c -o`	Programm übersetzen, optimieren und binden	ja
`fxc name.c -l`	Programm übersetzen, *nicht* binden	nein
`fxc name.c -l -o`	Programm übersetzen, optimieren, *nicht* binden	ja

Anwendungswissen

Das Anwendungswissen beschreibt die Anwendung im einzelnen. Die Problematik soll am Beispiel des Anwendungsexperten für Betriebssysteme erläutert werden. Die Übersetzung eines Programmes erfolgt mit einem Compiler und einer Reihe von Optionen[9]. Die Optionen folgen bestimmten Regeln. Einige Optionen können sich gegenseitig ausschließen bzw. erfordern die Angabe einer weiteren Option. Für diese Anwendung muß ein Werkzeug bereitgestellt werden, das in der Lage ist, Optionen des Compilers zu erkennen, zu verändern und gegebenenfalls den Benutzer auf Fehler aufmerksam zu machen. Auch müssen Standardvorgaben möglich sein. Die Tabelle 6.9 gibt einen winzigen Auszug möglicher Optionen und Standardbelegungen bei einem C–Compiler wieder. Eine als Standardvorgabe eingesetzte Option ist mit *ja* gekennzeichnet.

Beim Aufruf des Übersetzers muß das System *wissen*, welche Optionen erforderlich sind, um das Programm wie gewünscht zu übersetzen. Auch *default*–Einstellungen der Applikation müssen bekannt sein. Der Anwendungsexperte nimmt dann die Überprüfung oder Generierung der Standardvorgabe vor.

Das Objekt, das Anwendungswissen verwaltet, kann als Parser der Benutzeraktion ausgeführt werden. Um bei dem Beispiel zu bleiben, wird der Befehl bei Eintrag in die Wissensbasis in seine semantischen Bestandteile aufgelöst.

`fxc`	name.c	-o
Programmname (Aktionstyp)	Datenspezifikation	Optionen

Der Benutzer wird innerhalb der Oberfläche nur noch mit semantischen Objekten arbeiten, d. h., jede ausgeführte Aktion muß in die Syntax der Applikation überführt werden. Die Benutzeraktion wird unter einer externen Bedingung (Rechner, Betriebssystem) in den syntaktisch korrekten Befehl überführt.

[9]Schalter, *switches*

Benutzeraktion			externe Bedingung	
Aktion	Datenspezifikation	Optionen		
Übersetze	*Programmtext*	*Option*	*Rechner*	*Betriebssystem*
Compiler	name.c	optimieren	AWI10	UNIX
fxc name.c -o				
Compiler	name.c	optimieren	AWI5	VMS
cc name.c				

Wie das Beispiel zeigt, muß die Optimierung bei einem Compiler (`fxc`) angegeben werden, bei dem zweiten Compiler (`cc`) ist dies voreingestellt.

Der Anwendungsexperte ist ein starres Regelwerk, das mit jeder Erweiterung der Applikation geändert werden muß. Hier sind Schnittstellen für den Betreiber der Software gefordert, die Änderungen des Regelwerkes erleichtern. Mit einer Expertensystemshell kann ein Expertensystem erstellt werden, das vom Parser befragt wird und die notwendigen Antworten liefert.

Interaktionsmethoden

Die Interaktion des Benutzers mit dem Rechner erfolgt mittels einer graphischen, mausorientierten Oberfläche. Der Benutzer muß jede Aktion[10] mit der Maus auswählen, die Tastatur ist derzeit noch nicht als Ersatz/Zusatz bereitgestellt.

Abhängig von der Art des Dialogs kann jedoch nicht jede Aktion mit der Maus allein durchgeführt werden (z. B. Eingabe eines Namens). Hier helfen Selektionsmethoden wie Menüs, wenn die Gesamtheit der Auswahl bekannt ist, *prompter*, wenn eine Zeichenkette eingegeben werden muß oder eine Ja/Nein–Antwort erwartet wird. Dialogboxen mit komplexem Aufbau helfen bei komplexen Abfragen.

Die Antwort des Rechners kann visuell und zusätzlich akustisch erfolgen. Jede Aktion des Rechners wird mit einer visuellen Antwort gekennzeichnet, die zumindest aussagt, daß einer der Rechner aktiv ist und der Benutzer warten muß.

Diese Tatsachen sind fest implementiert und gegen Veränderungen geperrt. Einfache Regeln werden jedoch beherrscht, z. B. die Auswahl eines Begriffs aus einer Liste durch Angabe eines Teilbegriffs mit Platzhaltern. Ist mehr als ein Begriff vorhanden, wird ein Menü angezeigt, ist nur ein Begriff vorhanden, so wird dieser automatisch gewählt. An einigen Stellen erfordert die Auswahl die Bestätigung durch den Benutzer (z. B. Löschen von Dateien).

Bei der Eingabe von Befehlen wird der Benutzer im System auch durch eine einfache Kommandooberfläche unterstützt. Diese Kommandooberfläche wird aktiviert, wenn der Benutzer eine Eingabe über die Tastatur wählt. Ein Dialogfenster erscheint, das die Eingabe eines Befehls entgegennimmt. Dieses Verfahren sichert dem Benutzer die Möglichkeit, einen Befehl direkt ohne den Umweg über mehrere Fenster (wie beim Kopieren von Dateien z. B. beim Apple Macintosh) auszuführen.

[10]Die Eingabe eines Zeichens über die Tastatur wird nicht als Aktion betrachtet, die Eingabe eines Funktionscodes über die Tastatur hingegen schon, z. B. *cursor*–Tasten, *enter*–Taste, *control*–Tastencode etc.

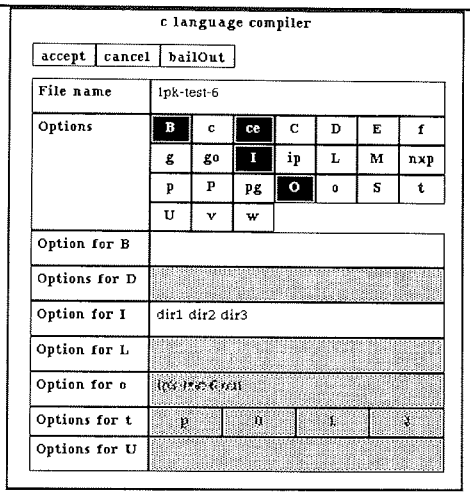


Abbildung 6.11: Die Abbildung zeigt ein *property sheet*, wie es bei der Eingabe der Optionen des Alliant FX80 C–Compilers benutzt wird. Es wird die Klasse `DBoxView` des *The Analyst* benutzt. Die grau unterlegten Felder sind für Eingaben und Selektionen gesperrt.

Zusätzlich verfügen zentrale Fenster, wie die `login`–Fenster, über ein Eingabe– und ein Antwortfenster. Hier erfolgt der Dialog nahezu wie auf einem gewohnten Terminal.

Zusammengefaßt wurden in dem System die Methoden Menü, Dialogbox, *Prompter*, Systemfenster und Terminal implementiert.

Je nach Übung und Kenntnis wird der Benutzer eine dieser Möglichkeiten wählen wollen. Da die Funktionalität des Systems durch die verschiedenen Interaktionsmethoden mehrfach verfügbar ist, besteht diese Möglichkeit der Auswahl.

In Abbildung 6.11 ist ein Beispiel eines Eigenschaftsfensters gezeigt, mit dem Einstellungen von Programmparametern, hier die Optionen eines C–Compilers, vorgenommen werden. Das Applikationswissen ist in den Regeln, welche Optionen andere ausschließen und welche Optionen weitere Parameter benötigen, verborgen.

Dialogwissen

Vom System her werden dem Benutzer festgelegte Dialogmethoden vorgegeben, die derzeit noch nicht individuell angepaßt werden können. Die Regeln, nach denen dieser Dialog implementiert ist, bilden eine Basis.

Eingabe einer Zeichenkette: Die Zeichenkette wird durch ein Dialogfenster, das nur diese eine Zeichenkette erwartet, abgefragt.

Auswahl aus einer Liste: Die Liste wird als Menü bereitgestellt, oder, wenn die Liste Teil eines Fensters ist, als Listenobjekt.

Komplexe Selektion: Komplexe Selektionen, z. B. die Druckmaske des Apple Macintosh, enthalten mehrere Basiselemente, aus denen die Dialogbox zusammengesetzt ist.

Der Benutzer muß für einen interaktiven Aufbau eines eigenen Dialogs die Basiselemente bzw. die Semantik kennen. Das System muß bereitstellen, welche Basiselemente für den Zweck geeignet sind, und gegebenenfalls Fehler in der Definition erkennen.

Im vorliegenden System wurde mit den verfügbaren Mitteln des *The Analyst* gearbeitet, daher sind zunächst alle Dialoghilfsmittel in ihrem Aufbau voreingestellt. Ein eigener Dialog muß derzeit noch in *Smalltalk-80* programmiert werden. Der Benutzer hat daher wenige Möglichkeiten einzugreifen. (Vergleiche auch Abbildung 6.11.)

6.3.3 Einige Details aus der Implementierung

In diesem Abschnitt werden einige Details vorgestellt, wie die vorgestellten Konzepte implementiert werden. Für eine ausführliche Betrachtung sei auf den Anhang A.6 verwiesen.

Betrachten wir zunächst die Übersetzung von Benutzeraktionen in Befehle verschiedener Betriebssysteme. Die Umwandlung der Benutzeraktionen in Betriebssystembefehle erfolgt über Parser und Codegeneratoren. Dabei ist für jedes Betriebssystem je ein Parser und ein Codegenerator vorgesehen. Mit einer abstrakten Klasse `OperatingSystemParser` werden die Grundoperationen zur Verfügung gestellt. (Siehe Abbildung 6.12.) Die spezielle Anpassung für das jeweilige Betriebssystem erfolgt über die einzelnen Unterklassen, z. B. für *VMS* mit der Klasse `VMSParser`. Jeder Parser überträgt eine Befehlssequenz des jeweilige Betriebssystems in die *GOMS***–Syntax und umgekehrt. Im folgenden Beispiel wird eine Befehlssequenz aus dem *UNIX*–System nach *VMS* übertragen. Das Ergebnis ist eine Liste der *VMS*–Befehle.

```
OperatingSystemParser
        gomsSequenceToSystem: (UNIXParser systemSequenceToGOMS:
            '           cp a b
                        mv a c
                        rm d')
        in: 'VMS'.

OrderedCollection
    ('copy a b'
     'rename a c'
     'delete d')
```

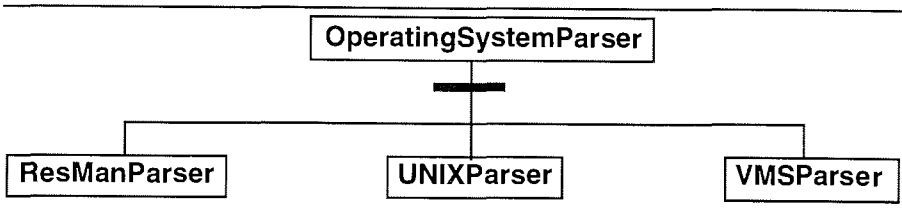


Abbildung 6.12: Die hier dargestellte Hierarchie der Parser erfolgt nach [SM88]. Erklärungen siehe Text.

Betrachten wir nun die Generierung eines Makros aus der Dialoghistorie. Die Makrogenerierung erfolgt durch die Analyse der eingegebenen Befehlssequenz. Dabei wird einem Objekt, hier der Klasse `Macro`, ein Pfad aus der Dialoghistorie übergeben. Das Objekt `Macro` analysiert diesen Pfad und erzeugt, mit Unterstützung des Benutzers, einen Makro. Die folgende Methode ist ein Auschnitt aus diesem Ablauf. Die Methode `macro:` wird bei der Makrogenerierung aufgerufen. Der Parameter ist ein Pfad, d. h. eine Liste von Historieknoten. Die Einzelheiten der Methode `macro:` sind im Anhang A.6 dargestellt.

```
Macro macro: aHistoryPath
```

Als nächstes betrachten wir die Erkennung von Aktionen und Plänen. Die Erkennung eines Plans durch *pattern-matching* wird in der folgenden Methode verdeutlicht. Die Methode wird in ihrer vollen Länge dargestellt. Die Pläne sind in einer Kollektion zusammengefaßt. Jedem einzelnen dieser Pläne wird die Nachricht `match:` mit der aktuellen Aktion als Parameter gesendet. Der Plan vergleicht die Aktion mit der intern aktuellen Aktion und setzt seinen internen Zeiger auf die folgende Aktion. Wenn keine Aktion mehr im Plan vorhanden ist, kann ein Aktionsblock ausgeführt werden. Wenn die Aktion nicht erkannt wurde, kann ein Fehlerblock ausgeführt werden.

```
'From The Analyst(tm) on Smalltalk-80, Version 2.3 of
13 June 1988 on 12 February 1993 at 6:41:31 pm'!

!Plan methodsFor: 'matching'!

match: anAction
  | f |
  "Ein matchen der Aktion findet nur statt, wenn die Aktion keine
  Dummy-Aktion ist. Da der Test auf Dummy-Aktion auch wahr ist, wenn
  die Dummy Aktion die letzte innerhalb eines Plans ist. Wird zusätzlich
  die Bedingung finalState überprüft. Diese Abfrage ist nur dann falsch,
  wenn der letzte Zustand erreicht ist. Dann wird in jedem Fall die Aktion
  gematched."
  (self finalState not and: [self actionIsDummy: anAction])
    ifFalse: [
      parameterCollection isNil
            ifTrue: [parameterCollection ←OrderedCollection new].
      anAction parameterOrdered do: [ :anAssoc |
        (parameterCollection includes: anAssoc key)
          ifFalse: [ parameterCollection add: anAssoc key]].
      f ← (actionCollection at: state) copy. "self halt."
      ((f replaceParameterFrom: parameterCollection)
          and: [f match: anAction])
        ifTrue: [
          state ←state + 1.
          state ←actionCollection size
              ifTrue: [
                self actionBlock value: state.
                self reset]]
         ifFalse: [
           self faultBlock value: state.
           self reset.
           anAction parameterOrdered do: [ :anAssoc |
             (parameterCollection includes: anAssoc key)
               ifFalse: [ parameterCollection add: anAssoc key]].
        f ← (actionCollection at: state) copy.
      ((f replaceParameterFrom: parameterCollection)
              and: [f match: anAction])
          ifTrue: [ state ←state + 1]
          ifFalse: [ self reset ]]]! !
```

6.4 Erfahrungen mit dem *ResourcenManager*

Der *ResourcenManager* wurde als Prototyp auf dem Apple Macintosh II entwickelt. Der Rechner besitzt keine entsprechend leistungsfähigen Schnittstellen zu anderen Rechnern wie UNIX–Arbeitsplatzrechner. Der Datentransfer zu anderen Rechnern, bzw. die Rechner selbst mußten daher auf dem Rechner simuliert werden.

An diesem simulierten System konnten alle Eigenschaften gezeigt werden:

- Direkte Manipulation von Objekten beim Kopieren, Löschen, Bewegen
- Über Eigenschaftsfelder gesteuerte Eingabe von Optionen beim Aufruf einer Ressource (z. B.: Compiler)
- Dialoghistorie
- Makro–Generierung

Bei einem vollständig ausgebauten System mit Zugriff auf die realen Rechner werden, wegen des Transfers vieler Daten beim Zugriff auf ein Directory, die Antwortzeiten des Systems steigen. Die Datenhaltung innerhalb des *Smalltalk*–Systems ist derzeit für Produktionsversionen nicht ratsam, da der verfügbare Objektspeicherbereich knapp ist[11]. Eine Speicherung der Objekte in einer Datenbank ist ein zukünftiges Projekt.

Der große Vorteil des Systems für den Benutzer liegt zunächst in der konsistenten Oberfläche für das Gesamtsystem und der zentralen Steuerung der Vorgänge. Zum zweiten wird der Benutzer von vielen Arbeitsgängen entlastet, z. B. durch die Anzeige und Vorauswahl von Optionen beim Aufruf des Compilers. An einer Auslagerung von Objekten wird zur Zeit gearbeitet [Rei92].

Bei der Implementierung wurden folgende Punkte besonders berücksichtigt (Vergleiche auch Abbildung 6.13):

Gesamtdarstellung der erreichbaren Rechner

Die Übersicht über das Netzwerk bzw. die erreichbaren Rechner erleichtern dem Benutzer das Navigieren und die Auswahl des Rechners. Zusätzlich können zu den einzelnen Rechnern Informationen über deren besondere Eigenschaften abgerufen werden (Hilfe). Über den Netzwerk–Browser können auch Daten transferiert werden. Das *Icon* eines Rechners entspricht dem Wurzelverzeichnis des Benutzers. Diese Regel muß im System vorhanden sein.

Gesamtdarstellung des Directory–Baumes

Die Übersicht über den Directory–Baum entbindet den Benutzer von der Aufgabe, die Kommandos zum Directory–Wechsel einzugeben. In einer Übersicht kann der Benutzer bequem navigieren. Diese Konzepte werden inzwischen auch von neueren Benutzungsoberflächen bereitgestellt. Besonders einfach gestaltet sich der Zugriff auf logische deklarierte Speichermedien *(mounten)*.

[11] Mit der neuen Version von *Smalltalk* ist diese Beschränkung nicht mehr gültig.

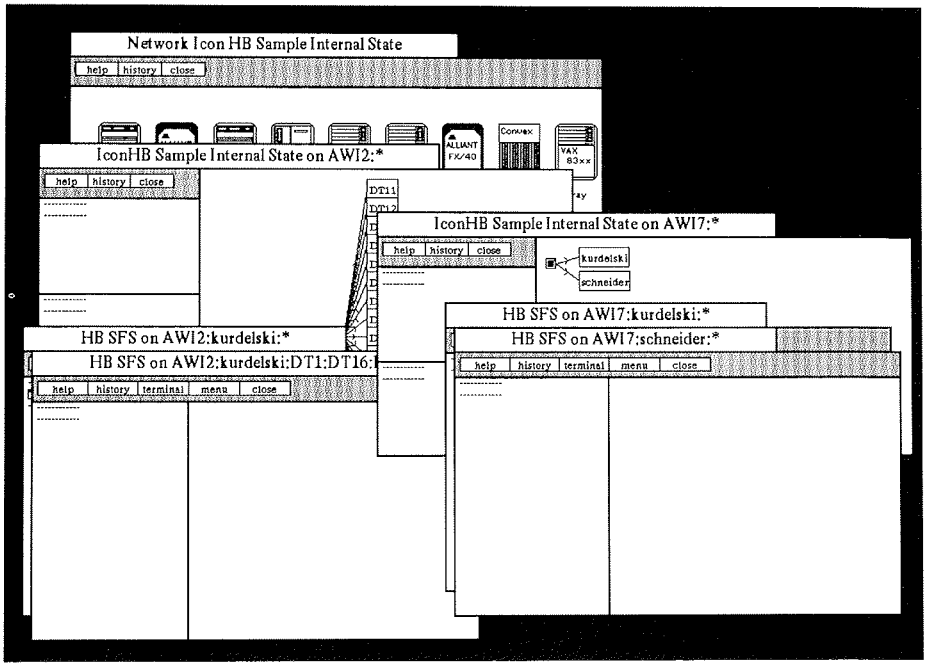


Abbildung 6.13: Die realisierten Fenster entsprechen dem geforderten Aufbau. (Vergleiche Abbildung 6.3, Seite 130, Erläuterungen siehe Text.)

Kompakte Anordung der geöffneten Directory–Fenster

Die geöffneten Directory–Fenster *eines* Rechners werden wie ein Kartenstapel übereinander angeordnet, so daß aber immer noch das Label sichtbar bleibt. Dieser Stapel kann nur als Ganzes verschoben werden. Wie aus einem Kartenstapel kann ein Directory–Fenster herausgezogen und auf den Stapel gelegt werden. Diese Darstellung vermeidet ein Durcheinander auf dem Desktop, insbesondere, wenn viele Fenster geöffnet sind. Diese Darstellung kann durch die Erfahrungen begründet werden, die im Umgang mit Fensteroberflächen gesammelt wurden. Die Betrachtung eines Fenster allein ist möglich. Das Auffinden einer Datei wird durch ein spezielles Programm ermöglicht. Zum Kopieren von Dateien muß jedoch ein Teil des Zielordners sichtbar sein. Dies wird durch den sichtbaren Label erreicht.

Direkte Eingabe von Befehlen

Die direkte Eingabe von Befehlen ist ein Zugeständnis an die *intermediate user*, die wissen, welchen Befehl sie eingeben müssen, und die wissen, wo die Daten in der Directory–Hierarchie stehen. In vielen Fällen ist es einfacher,

einen Befehl direkt einzugeben, als mühsam in der Directory–Hierarchie zu navigieren, bis die Datei gefunden wurde.

Historiebaum für *alle* Aktionen auf *allen* Ebenen

Die Dialoghistorie wird für die Sitzung des Benutzer erstellt. D. h., jede Aktion des Benutzers, die korrekt ausgeführt wird, erscheint in der Dialoghistorie. Anfängliche Überlegungen, die Dialoghistorie an ein Fenster, i. e. ein Directory, zu binden, wurden wieder aufgegeben, da (1) diese Information aus der Gesamthistorie extrahiert werden kann und (2) die Verwaltung der einzelnen Historiebäume vom Benutzer geleistet werden muß. Dies widerspricht der Einfachheit der Bedienung der Oberfläche. (3) Mit einer Historie werden die Änderungen des Gesamtzustands des Systems betrachtet. Bei einer Gesamthistorie ist zusätzlich der Verwaltungsaufwand geringer, da eine Aktion nur an eine Dialoghistorie angefügt werden muß. Es entfallen die Überprüfungen, welche weiteren Historien von der durchgeführten Aktion betroffen sind[12].

Hilfe

Das System verfügt über passive Hilfe. Hilfe ist durch Schlüsselwörter, die an ein Hilfe–Objekt gesendet werden, das daraufhin die entsprechende Hilfe anzeigt, kontextgesteuert. Auf automatische Hilfe wurde verzichtet, da diese störend wirken kann oder nicht genügend Platz in einem Fenster vorhanden ist, um ausführliche Hilfe zu geben. In jedem Hilfe–Fenster sind, wenn erforderlich, Begriffe hervorgehoben, über die dann weitere Hilfe angefordert werden kann. (Siehe auch Abbildung 6.14.)

Papierkorb lokal für den Rechner und das Netzwerk

Beim Papierkorb wurde jedoch lokal gedacht. Die Daten sind lokal auf einem Rechner verfügbar. Der Benutzer wird sie über diesen Rechner löschen. Wenn der Benutzer dazu übergeht, sämtliche Aktionen über das Netzwerk abzuwickeln, ist eine zentrale Verwaltung der gelöschten Daten erforderlich, um Inkonsistenzen zu vermeiden und den Verwaltungsaufwand gering zu halten.

Bei der Dialoghistorie sind insbesondere die Ergebnisse aus der theoretischen Betrachtung eingeflossen. Dabei wurden die Probleme wieder deutlich, die schon bei der Betrachtung des Befehls *delete* auftreten: inverse Funktion, Behandlung des Systemzustands. Bei der theoretischen Betrachtung ergab sich schließlich, daß es Fälle gibt, in denen kein *UnDo* oder *ReDo* möglich ist, da die Systemzustände nicht unabhängig sind. Es stellt sich dann die Frage, sollte die Dialoghistorie dennoch als Baum realisiert werden oder nicht, d. h., ein Verzicht auf die Ergebnisse aus der theoretischen Betrachtung in Kauf genommen werden. Die Entscheidung für den Baum fiel, nachdem eingesehen wurde, daß ein Baum logische Vorteile birgt. Zum ersten erscheint eine lineare Folge von Aktionen auch im Baum, so daß bequem ein Makro

[12]Beim Kopieren von einem *Directory* in ein anderes werden zwei Dialoghistories beeinflußt: die Historie des Quellverzeichnisses und die des Zielverzeichnisses.

erzeugt werden kann. Zum zweiten kann bequem erkannt werden, welche Wege an Verzweigungen eingeschlagen wurden, ohne daß diese in der Historie verloren gehen. Es konnte gezeigt werden, daß die linearen Modelle von Archer, Conway und Schneider und Herczeg sowie das Baummodell von Vitter ebenfalls berücksichtigt sind.

Bei der Implementierung der Sicht auf die Dialoghistorie wurden zwei Wege eingeschlagen:

1. Darstellung als Graphik in Form eines Baumes
2. Darstellung in textueller Form als Liste mit Einrückungen, die die Hierarchie verdeutlichen

Der Vorzug der graphischen Darstellung liegt unter anderem in der einfacheren Deutung. Bei der textuellen Darstellung kann mehr Information auf dem gleichen Raum dargestellt werden. Die textuelle Darstellung ist optional verfügbar. Als Einstellung kann sie der Oberfläche als *default* bekanntgegeben werden.

Anhand eines kurzen Scenarios soll die Arbeit mit dem *ResourcenManager* gezeigt werden. Die Darstellung des Scenarios zeigt *nicht* die wichtigen Funktionen wie Kopieren durch Bewegen eines Icons von einem Ordner in einen zweiten. Auch Menüs werden nicht gezeigt. Dieses Scenario zeigt Zustände des Systems nach den einzelnen Aktionen des Benutzer.

In Abbildung 6.15 (Seite 164) wird der Zustand der Oberfläche, wie sie sich dem Benutzer darstellt, gezeigt, nachdem der Benutzer die wichtigsten Funktionen angesprochen hat:

- Öffnen des *ResourcenManager*–Fensters
- Öffnen eines Zuganges zu einem Rechner
- Öffnen eines Ordners
- Öffnen der Dialoghistorie
- Öffnen des Papierkorbs
- Der Benutzer hat bereits zwei Dateien gelöscht

Im Fenster der Dialoghistorie (`History`) sind die bisher ausgeführten Befehle zu erkennen. Im Papierkorb sind die beiden Dateien zu sehen.

Abbildung 6.16 (Seite 165) zeigt das Scenario, nachdem der Benutzer das Löschen der beiden Dateien mittels *UnDo* zurückgenommen hat. Die Aktionen sind in der Dialoghistorie grau unterlegt, und die Dateien sind aus dem Papierkorb verschwunden.

In den Menüleisten ist der *help*–Button zu erkennen, der eine kontextsensitive Hilfe aufruft.

Das Scenario der Planbearbeitung wird in Abbildung 6.17 und 5.10 (Seite 166 und 122) gezeigt. Über diese Browser kann ein Plan eingegeben bzw. modifiziert werden. Falls nicht bereits bei der Generierung des Makros geschehen, kann hier die Festlegung der Parameter als Konstante, Pseudo–Konstante oder Variable erfolgen. Diese Festlegung kann menügeführt oder durch Angabe der korrekten Nachricht erfolgen.

Abschließend wird in Abbildung 6.18 (Seite167) das Planerkennungsverfahren gezeigt. Zur Verfügung stehen vier Pläne, denen die Aktionen 1–5 (im untersten Fenster) geschickt werden. Die Pläne zeigen in dieser Ansicht, daß die Aktionen zweifelsfrei erkannt werden. Die Belegung der Parameter innerhalb eines Plans erfolgt durch Ersetzen der Parameter durch den angegebenen Namen. Ein Plan wird abbrechen, wenn die Reihenfolge der Parameter nicht exakt eingehalten wird[13].

[13]Dieser Effekt kann auftreten, wenn innerhalb eines Plans die Parameter festgelegt sind und ein Kopierbefehl abgesetzt wird, der mehrere Dateien in ein Directory kopiert. Wird die Reihenfolge hier nicht exakt eingehalten, bricht der Plan ab. Dies kann jedoch durch die Beachtung der Semantik des Befehls umgangen werden, indem die Befehlsliste als *Set* aufgefaßt wird. Hier spielt die Anordnung keine Rolle.

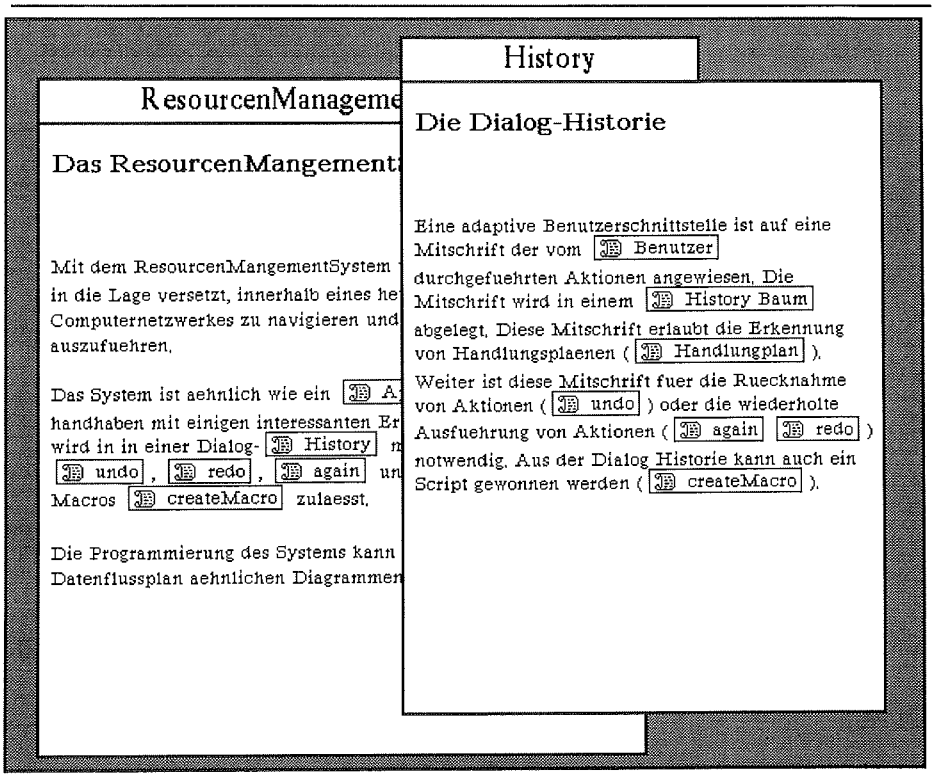


Abbildung 6.14: Die Abbildung zeigt eine Situation, wie sie bei der Anforderung von Hilfe entsteht. Kurze, prägnante Informationen geben dem Benutzer Hinweise. Über die eingerahmten Worte kann durch *doppelclick* ein weiteres Hilfe–Dokument geöffnet werden. Der Benutzer suchte zunächst nach der Erklärung des Begriffs *ResourenManagementSystem* und dann nach dem Begriff *History*.

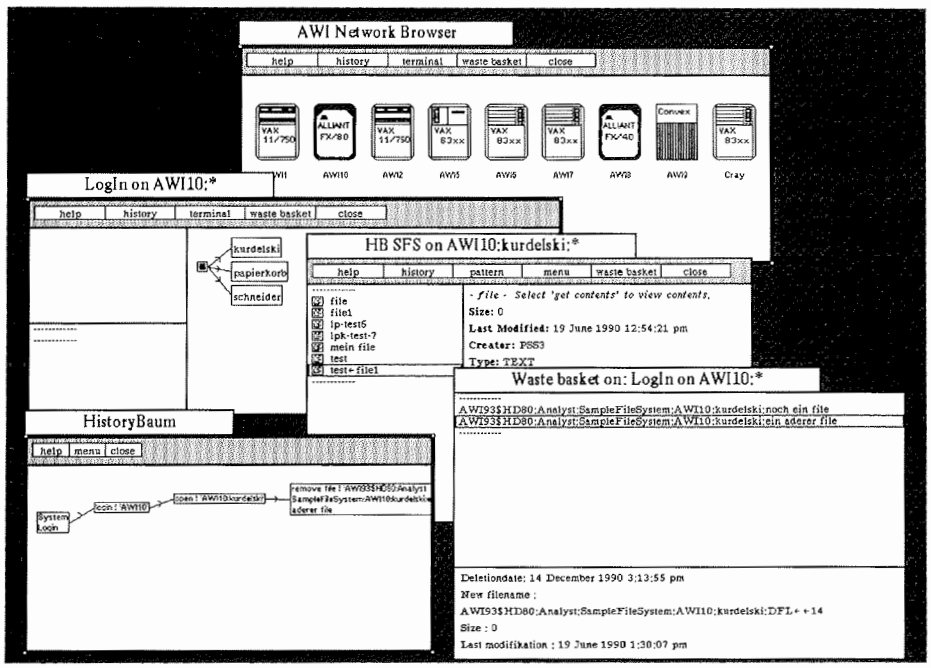


Abbildung 6.15: Der Anfang einer Sitzung eines Benutzers mit den wichtigsten Fenstern: Netzwerk, Rechner, Directory, Papierkorb und Historie

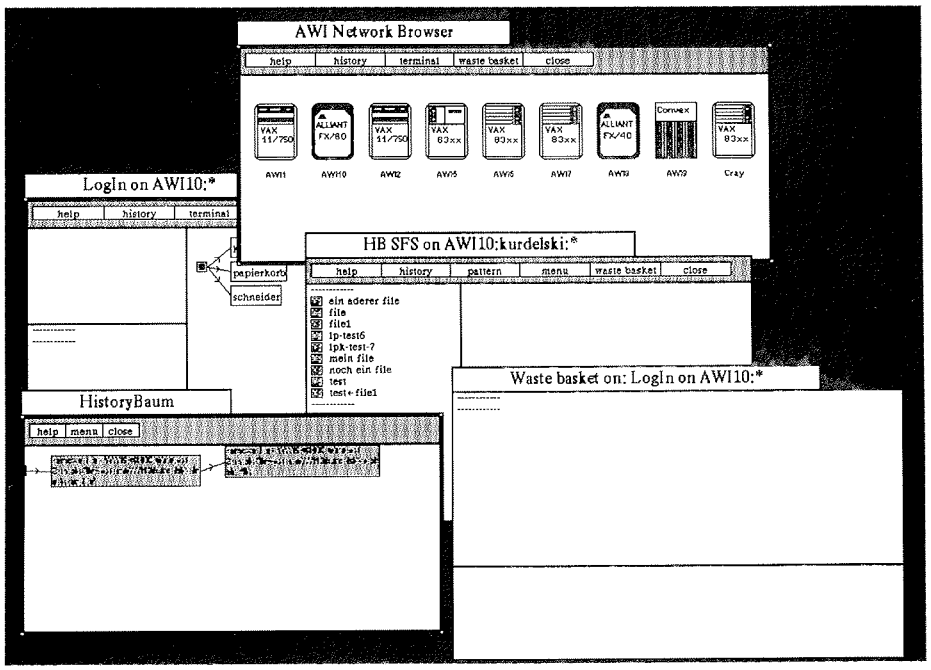


Abbildung 6.16: Das Scenario nach einem *UnDo*. Es ist zu erkennen, daß die zurückgenommenen Aktionen in der Historie grau unterlegt sind.

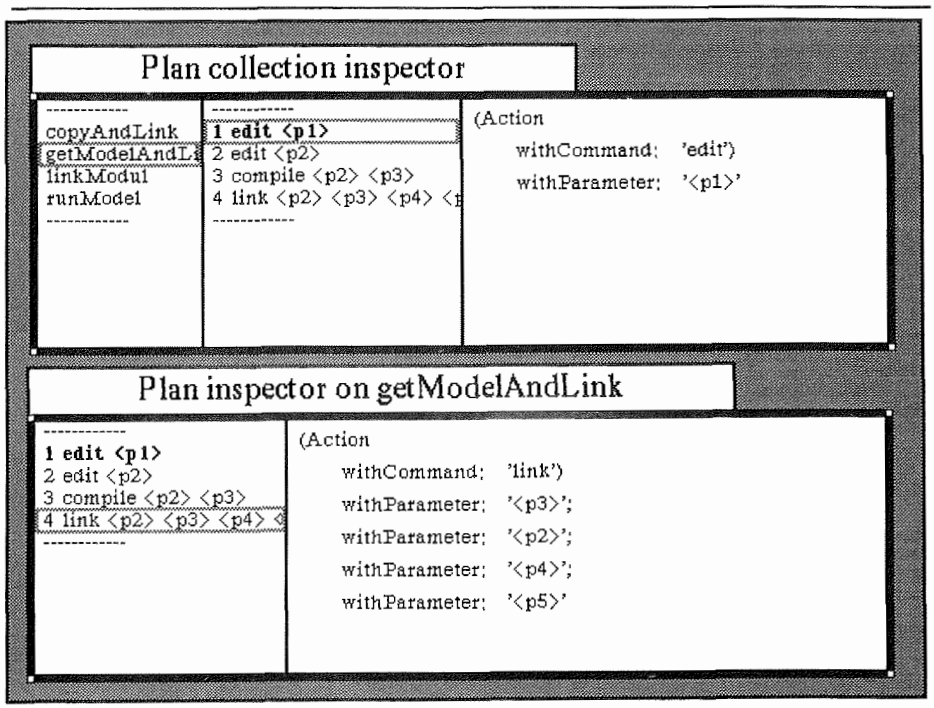


Abbildung 6.17: Die Plandarstellung mittels *Browser*. Die Pläne können gelöscht oder verändert werden. Ein Hinzufügen ist ebenfalls möglich.

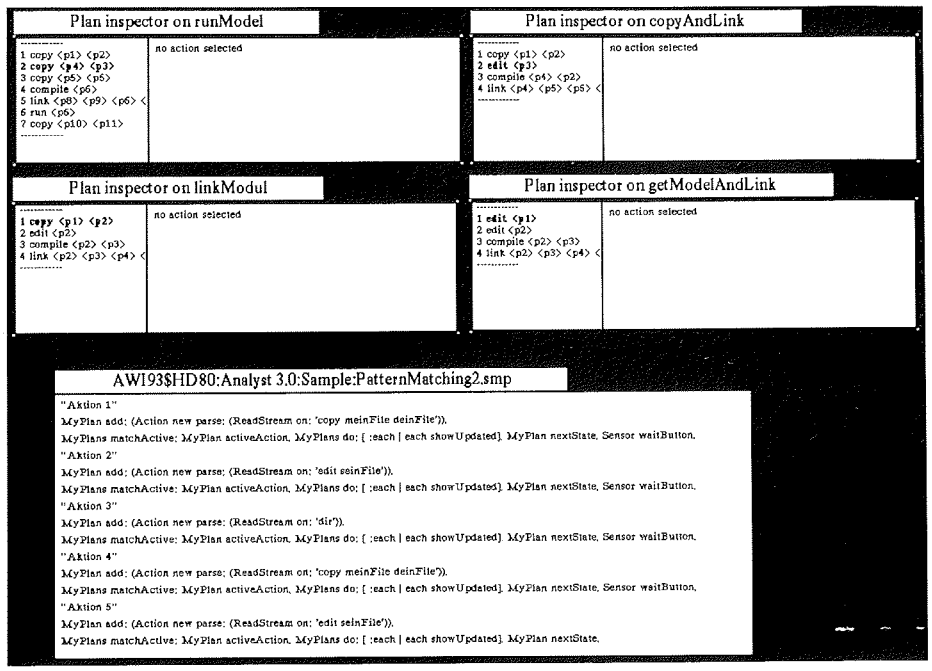


Abbildung 6.18: Hier wird der Ablauf der Planerkennung an einem fiktiven Beispiel gezeigt. Die vier oben gezeigten Pläne haben die Aktionen erkannt (oder sind zurückgesetzt worden). Der aktuelle Zustand eines Plans wird durch die hervorgehobenen Aktionen in der Aktionsliste angezeigt. Die verfügbaren Pläne sind *runModel, copyAndLink, linkModul* und *getModelAndLink*. Die erkannten Aktionen sind in der gleichen Reihenfolge die Aktionen 2 *(copy <p1 ><p2>)*, 2 *(edit <p2>)*, 1 *(copy <p1 ><p2>)* und 1 *(edit <p1>)*.

Ein Werkzeug ist immer zugleich ein Modell für seine eigene Reproduktion und eine Gebrauchsanweisung für die erneute Anwendung der Fähigkeiten, die es symbolisiert.

Joseph Weizenbaum

Kapitel 7

Ausblick

7.1 Zusammenfassung der Ergebnisse

In der vorliegenden Arbeit wurde, ausgehend von der Situation im Alfred–Wegener–Institut, ein *ResourcenManagementSystem* entwickelt, das die Aspekte

- direktmanipulierbare Schnittstelle,
- Analyse des Benutzerverhaltens,
- Speicherung der Benutzeraktionen in abstrahierter Darstellung,
- Bereitstellung von statischen Regeln und Fakten,
- Bereitstellung von dynamisch modifizierbaren Regel– und Faktenbasen,
- formale Beschreibung der Dialoghistorie und
- Integration von Makroerstellung, –verfolgung und –erkennung

berücksichtigt. Zu allen Punkten existieren in der Literatur mehrere Ansätze, die jeweils ein Teilgebiet beschreiben. In dieser Arbeit wurde nun die Integration der Ansätze, verbunden mit neuen Ideen, an einem realen System durchgeführt. An Hand des *ResourcenManagementSystems*, das nach diesen Gesichtspunkten entwickelt wurde, konnte verifiziert werden, daß dieser Ansatz erfolgreich war. An einigen Stellen hat sich gezeigt, daß weiterführende Arbeiten notwendig sind, um den Benutzer noch effektiver zu unterstützen. Im folgenden wird das Verfahren kurz diskutiert und ein Ausblick auf weiterführende Arbeiten gegeben.

7.2 Diskussion des Modellierungsverfahrens

Die Ergonomie einer Benutzungsoberfläche wird in zunehmendem Maße auch von der Fähigkeit des Systems bestimmt, auf die Aktivität des Benutzers zu reagieren und ihn gegebenenfalls zu unterstützen. Der Benutzer wählt Objekte aus, mit denen er seine Arbeit erledigen kann. Um diese Objekte und die damit verbundenen Aktivitäten des Benutzers erfolgreich zu erfassen und dem Benutzer geeignete Werkzeuge zur Manipulationen seiner Daten in die Hand zu geben, wurde im Rahmen der Arbeit der objektorientierte Ansatz gewählt. Der Benutzer mußte außerdem in die Lage versetzt werden, die Objekte zu erkennen und dann zu manipulieren. Im Rahmen der Arbeit wurden verschiedene objektorientierte Modelle auf ihre Eignung geprüft. Dabei stellte sich heraus, daß kein Ansatz das Problem vollständig erfaßt.

Aus diesem Grunde wurde ein Kombinationsansatz gewählt, bei dem die positiven Aspekte der in Kapitel 3 diskutierten Modelle genutzt wurden. Globale Zusammenhänge, Relationen und Datenflüsse zwischen Objekten konnten mit *RFA*–Netzen dargestellt werden. Die parallel durchgeführte objektorientierte Analyse lieferte einen statischen Überblick über die Zusammenhänge zwischen den Objekten. Mit ihrer Hilfe konnten die Attribute der Objekte und die benötigten Dienstleistungen einfach gefunden und dargestellt werden.

Um die Aktivitäten der Benutzer zu beschreiben, werden in der Literatur unterschiedliche Grammatiken verwendet. Mit diesen Grammatiken werden die Regeln, nach denen bestimmte Aktionen vom Benutzer ausgewählt werden, notiert. Das Ziel der vorliegenden Arbeit war, die Aktivitäten der Benutzer in ihrer Grundform erst einmal zu erfassen, d. h., auch der Anwender soll eine Methode erhalten, mit der er seine Aktionen planen kann. Diese Pläne finden dann als Eintrag in der Wissensbasis Verwendung.

Die individuelle Arbeitsgestaltung der Wissenschaftler am AWI kann nicht mit einfachen, für eine Großzahl von Wissenschaftlern gültigen Standardsequenzen erfolgreich unterstützt werden. Aus diesem Grunde wurde insbesondere eine Modifikation des *GOMS**–Modells erforderlich. Die Abstraktion von einer Aktionsfolge zu einer formalen Beschreibung war ein Beweggrund für die Formalisierung. Mit diesem neuen *GOMS***–Modell ist es nun möglich, Selektionsregeln für Planverzweigungen und Fakten sowie Vorbedingungen für Aktionen kompakt darzustellen. Die Syntax des *GOMS***–Modell ist in **EBNF** dargestellt. Eine erwünschte Folge dieser Darstellung ist die Möglichkeit, für diese Beschreibungssprache einen Übersetzer zu konstruieren, der eine *GOMS***–Sequenz in die Syntax der verschiedenen Betriebssysteme transformiert.

Um eine geschlossene Darstellung der Aktionspläne des Anwenders zu erhalten, ist eine Mitschrift seiner vergangenen Aktionen notwendig. Diese Mitschrift wurde als Baum implementiert. Die hier auftretenden Probleme wurden diskutiert. Die Baumstruktur wurde gewählt, um dem Benutzer jede Änderung in ihrem Kontext zu präsentieren. Parallele Zweige in dieser Historie können zu Widersprüchen im Datenbestand führen. In der vorliegenden Arbeit wurde eine Darstellung der

Handlungsalternativen als besonders wichtig erachtet. Mit geeigneten Funktionen (*combine*, *cut*) und einer dadurch bedingten Einschränkung der Baumstruktur konnten diese Widersprüche beseitigt werden. Die Historie wird insbesondere zur Generierung von Makros aus den Pfaden des Baumes genutzt. Diese Makros können als parametrisierte Pläne bereitgestellt werden. Außerdem können bekannte Ansätze mitverwendet werden, z. B. das *UnDo*–Rahmensystem von Herczeg.

Der Erkennungsmechanismus für Handlungspläne ist hier noch einfach gehalten. Jede gerade ausgeführte Aktion wird mit jedem verfügbaren Plan verglichen. Dadurch werden verschiedene Pläne aktiv, d. h. als fortsetzbar erachtet. Die Pläne besitzen einen Triggermechanismus, der bei vollständiger Erkennung, bei Nichterkennung oder bei einem Zwischenzustand schalten kann. Das Schalten eines Zwischenzustandes wird dann von Regeln und Fakten in den Wissensbasen beeinflußt.

Die Makroerkennung und –generierung berücksichtigt durchgängig das *GOMS***–Modell. Die Aktionen werden in einer leicht modifizierten, für die Erkennung vorbereiteten Form abgelegt. Makros werden aus der Dialoghistorie in dieser Form übernommen. Der Benutzer kann interaktiv die Makros und Pläne verändern. Dabei bewegt er sich sowohl in der Aktionsebene, wenn er die Parameter einer Aktion modifiziert, als auch auf einer Metaebene, wenn er die Struktur des Makros verändert oder Aktionen hinzufügt oder löscht.

Im folgenden wird diskutiert, wie die Arbeiten am *ResourcenManagementSystem* weitergeführt werden können.

7.3 Adaptive Erweiterungen

Am Anfang der Entwicklung stand der Wunsch, das System adaptiv zu gestalten. Ein adaptives System ist in der Lage, aus den Aktionen des Benutzers Verhaltensregeln abzuleiten, die in späteren ähnlichen Situationen als Script benutzt werden können. Die vorliegende Arbeit beschränkt sich zunächst auf ein adaptierbares System[1].

Ein Teil der Implementierung setzt sich mit der Planerkennung auseinander. In diesem Rahmen wurde ein einfaches *pattern–matching*–Verfahren gewählt, um aus einer Handlungssequenz einen Plan herauszufiltern. Dem Benutzer können auf diese Weise mehrere Pläne zur Auswahl vorgelegt werden. Aktionen werden gemäß der Darstellung in Kapitel 5.3.1 gefiltert. Eine Aktion wird nur dann erkannt, wenn sie exakt mit der gespeicherten parametrisierten Aktion übereinstimmt. Eine äquivalente Behandlung gilt für Aktionsfolgen. Der konzeptionelle Ansatz des Systems ist so gestaltet, daß zu einem späteren Zeitpunkt dieser Erkennungsmechanismus erweitert werden kann.

Ein adaptives System muß auch mit aktiver Hilfe aufwarten. Die Hilfe wird im vorliegenden System auf den aktuellen Zustand des Systems bezogen, ist jedoch nur

[1]Diese Beschränkung heißt jedoch nicht, daß das System nicht adaptiv sein kann. In einigen Situationen verhält es sich adaptiv, z. B. bei der Generierung von Makros. Wenn ein Makro generiert oder entfernt wird, werden alle Menüs, die auf die Liste der Makros Bezug nehmen, automatisch angepaßt. Der Benutzer sieht dann immer die aktuelle Liste von Makros.

auf Anforderung durch den Benutzer verfügbar. Durch Festlegung eines *Kompetenzfaktors*, auf den sich eine erneute Meldung bezieht und der sich ständig anpaßt, kann ein Lernen des Systems erreicht werden. Die Detaillierung der Hilfe erfolgt ebenfalls über diesen Kompetenzfaktor. Zusätzlich zur kontextbezogenen Hilfe benötigt der Benutzer eine Referenz über die Möglichkeiten des Systems. Diese Aufgabe kann ein *online*–Manual[2] übernehmen. Der Einstiegspunkt in dieses *online*–Manual erfolgt über den aktuellen Zustand der aktiven Hilfe.

Eine einfache Adaptierung des Systems kann auch durch das *dynamic–defaulting* erzielt werden. Dabei werden Auswahlangebote[3] so gesteuert, daß beim Aufruf des Auswahlangebotes der Zeiger auf eine Auswahl zeigt, die sich in sinnvoller Weise (Regeln) aus der vorhergehenden Aktion ergibt, z. B. ein *cut* oder *copy* erzwingt ein *paste* als Selektionsangebot. Das *dynamik defaulting* wird z. B. bei allen Menüs eingesetzt. (Siehe Abbildung 7.1.) Auch Eigenschaftsfelder werden unterstützt. Die zuletzt ausgewählte Aktion oder Einstellung wird beim *dynamic–defaulting* erneut zur Verfügung gestellt. Hierzu auch die folgende Tabelle:

Auswahl	Beschreibung	Beispiel
Menüs	Die zuletzt gewählte Aktion wird erneut angeboten.	Jedes Menü
Prompter	Frühere Eingaben des Benutzers	Druckereinstellung
	Getätigte Selektionen innerhalb eines Browsers	Filemanipulation
Property Sheet	Die letzte Einstellung	(Compiler Option)
	Eine Grundeinstellung	Erster Aufruf

Das System ist derzeit noch nicht im Sinne der Systeme von Hoppe und Kobsa lernfähig [HP89, Hop89, Kob84, Kob85]. Aber die Trennung in Aktionsobjekte[4] und Datenbankobjekte[5] macht eine Anpassung einfach, da nur die Eigenschaften der verwendeten Objekte modifiziert werden müssen.

7.4 Fazit und Vorausschau

Die Untersuchungen haben gezeigt, daß trotz des Einsatzes von graphischen Benutzungsoberflächen, wie *OSF–Motif*, die wirkungsvolle Unterstützung der Anwender mit leistungsfähigen Werkzeugen bisher nur in Ansätzen erreicht ist. Während der Entwicklung des *ResourcenManagementSystems* am AWI wurden die Betriebssysteme der verfügbaren Rechner ständig weiterentwickelt. Derzeit gibt es noch wenige

[2]Neuere Versionen von bekannten Programmen wie *Word* und *Excel* verfügen über ein *online*–Tutotrial, das dem Benutzer die ersten Schritte mit dem Programm erläutert, und *online*–Hilfe, die mit einem Thesaurus gekoppelt ist, um sinnverwandte Begriffe bei der Hilfe mitanzugeben.

[3]Menüs, Dialogfenster, *property sheets*

[4]Aktionsobjekte kennzeichnen die Objekte, auf denen der Benutzer operiert und mit denen er agiert.

[5]Zu Datenbankobjekten werden hier auch die Wissensbasen hinzugezählt.

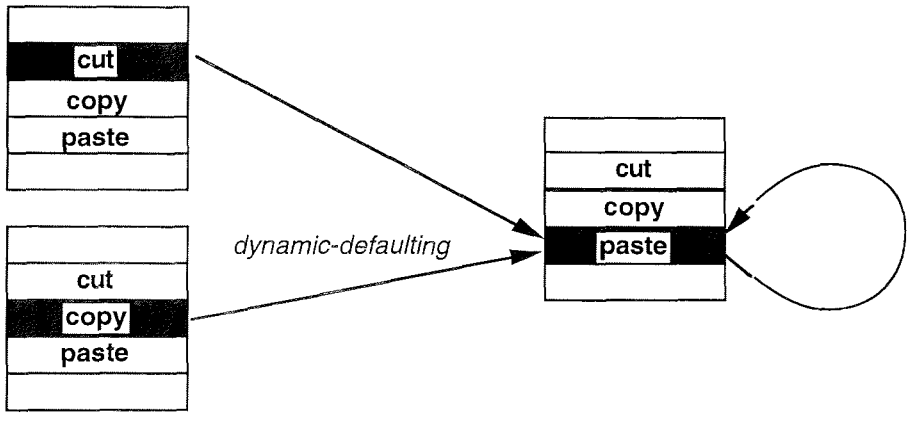


Abbildung 7.1: Ein Menü, das mehrfach aufgerufen wird, bietet eine Vorauswahl an, die aus der vorhergehenden Aktion sinnvoll folgt.

Spezialrechner, die eine einfache Oberfläche anbieten[6]. Dennoch wird meist nur ein Rahmen geboten, der dem Anwender die Arbeit erleichtert, der jedoch keineswegs benutzerorientiert ausgelegt ist. Ansätze unter Verwendung *künstlicher Intelligenz* fehlen ganz. In der vorliegenden Arbeit wurde ein Weg gezeigt, die Wünsche und Forderungen der Benutzer mit benutzerorientierten Methoden aufzunehmen und sie in ein reales System zu überführen. Zusätzlich wurde ein Ansatz für den Einsatz von Methoden und Verfahren aus der *KI* gezeigt.

Die Manipulation von Objekten mit einem Zeigeinstrument ist unbestritten einfach, aber das Zeigeinstrument muß nicht unbedingt eine Maus sein. Die neuen, mit einem Zeichenstift zu bedienenden transportablen Rechner[7] demonstrieren dies eindrucksvoll. Auch die neuen Möglichkeiten, die von Multi–Media und Hypertext–Systemen angeboten werden, werden dem Benutzer eine große Hilfe sein. Das Erkennen einer handschriftlichen Eingabe macht es zudem einfacher, mit dem Rechner zu kommunizieren. Die Eingabe in natürlicher Sprache ist jedoch noch ungelöst. Akustische Spracheingabe bei Rechnern ist noch nicht verbreitet. Hier gibt es Probleme bei der reproduzierbaren Erkennung einer Person und der Verarbeitung natürlicher Sprache.

Die ungelöste Frage der kommerziellen Systeme bleibt aber die individuelle, auto-

[6]Auch Superrechner wie die CRAY besitzen inzwischen komfortable Oberflächen zur Administration.

[7]Es gibt hier mehrere Begriffe, die auch gleichzeitig Produktbezeichnungnen sind: *Notebook, PowerBook, Pen–Computer*. Der letztgenannte ist ein Rechner, der nicht mehr über eine Tastatur sondern allein mit einem Schreibstift bedient wird. Daher der Name. Dabei ist die Erkennung von Handschriften ein wichtiger Aspekt.

matische oder halbautomatische Anpassung an die Bedürfnisse des Menschen. Nach wie vor muß sich der Mensch den, wenn auch wesentlich leichter zu bedienenden Maschinen anpassen. Die Entwickungen in der *KI*–Forschung zeigen zwar Fortschritte, aber die Maschinen, mit denen die geforderten Resultate geliefert werden, sind noch nicht am Arbeitsplatz einsetzbar. Daher wird weiterhin auf gute Ansätze gehofft, die die Unterstützung des Entwicklers von Applikationen und des Benutzers vor Ort ermöglichen.

Unter diesen Voraussetzungen ist keiner der Gedanken, die in dieser Arbeit gesagt wurden, überholt. Statt dessen sollte eine Weiterentwicklung des *ResourcenManagementSystems* unter den Aspekten

- Weiterentwicklung des Planverfolgungsmechanismus,
- erweitertes Lernen von Handlungsplänen,
- Weiterführung des *Wissenschaftler*–Benutzermodells,
- Repräsentation des Wissens,
- Manipulation von Objekten und
- Interaktionsmethoden

erfolgen. Nicht vergessen werden darf dabei der Feldtest im Stile des *KeyStroke–Level*–Modells, der dann quantitative Daten über die Aktionen des Benutzers liefert. Denn nur eine Überprüfung der Akzeptanz wird über den Erfolg einer Benutzungsoberfläche entscheiden.

Anhang A

Fachspezifische Anhänge

Die folgenden Anhänge wurden nicht mitübernommen und sind der Originalarbeit des Verfassers zu entnehmen.

A.1 EBNF für *GOMS***

A.2 *GOMS* Analysen, Vorbereitung, Veränderung

A.3 Abstrakte Datentypen und der Historiebaum

A.4 Analyse der VMS– und UNIX–Befehle bezogen auf die Dialoghistorie

A.5 Konzeptionelle Objekte des *RM*

A.6 Ausgewählte Beispiele des Quellcodes

A.7 TEX-Desktop

Anhang B

Glossar

Adaptive Benutzerschnittstelle Benutzerschnittstelle, die sich hinsichtlich der Informationsdarbietung, des Eingabemodus, ggf. auch hinsichtlich des Funktionsangebotes an die individuellen Eigenschaften des jeweiligen Benutzers oder an den Aufgabenkontext anpaßt. Im Unterschied dazu heißt eine Schnittstelle *adaptierbar*, wenn sie durch explizite Voreinstellungen individuell konfiguriert werden kann (nach [Bal88]).

Akzeptanz Oberflächliche Annahme eines Systems durch die Benutzer. Die bloße Beschränkung auf eine vordergründige und kurzfristige Betrachtung der Akzeptanz verhindert oft einzig sinnvolle inhaltlich tiefergehende Änderungen z. B. des Arbeitsinhaltes (nach [Bal88]).

Aufgabe Die Zielsetzung, einen gegebenen Ausgangszustand in einen gewünschten Zielzustand zu überführen. Dabei kann es sich um eine organisatorisch vorgegebene Aufgabe, um die subjektive Interpretation der Aufgabenstellung oder um eine selbstgestellte Aufgabe handeln. In der Software–Ergonomie ist der Begriff *Aufgabe* nicht auf Handlungen im Rahmen von Erwerbstätigkeiten beschränkt (nach [Bal88]).

Aufgabenanalyse Methodisches Vorgehen zur Bestimmung von Aufgaben sowie deren Struktur und Merkmale. Je nach Kontext wird die Aufgabenanalyse durchgeführt, um aus einem nicht rechnergestützen Arbeitskontext Anforderungen an eine Systemgestaltung abzuleiten oder für ein bestehendes System Verbesserungen zu spezifizieren. Die Aufgabenanalyse liefert Informationen für den Systementwurf (nach [Bal88]).

Aufgabenanalyse, kognitive Die kognitive Aufgabenanalyse wird zur Erfassung einer Aufgabe durch den Benutzer herangezogen. Aus einer Modellierung der kognitiven Aufgabenrepräsentation können Aussagen über Erlernbarkeit, Handlungseffizienz, Fehlerwahrscheinlichkeit, Konsistenz u. a. von Benutzerschnittstellen gewonnen werden (nach [Bal88]).

Auskunftssysteme Sie geben dem Benutzer Auskunft über alle den Einsatz der Mensch–Computer–Schnittstelle und der Anwendung betreffenden Fragen (nach [Bal88]).

Benutzungsmodell Ein Modell des Systems über den Benutzer. Dieses beinhaltet Fakten oder Annahmen über einen Benutzer, d. h., systemrelevante Fähigkeiten und Eigenschaften des Benutzers. Benutzungsmodelle sind die Voraussetzung für eine individuelle Behandlung von Benutzern und/oder Benutzergruppen. Dynamische Benutzungsmodelle können im Lauf des Dialogs aufgebaut und modifiziert werden (nach [Bal88]).

Benutzerschnittstelle Unter einer Benutzerschnittstelle werden häufig die für eine Computeranwendung gewählten Methoden des Mensch–Maschine–Dialogs verstanden. Technisch gesehen ist dies die Menge der Programme, die der Abwicklung des Mensch–Maschine–Dialogs dient.

Beratungssysteme beraten den Benutzer über den Einsatz der Mensch–Computer–Schnittstelle und die Anwendungssysteme betreffenden Fragen. Für die Beratung wird ein Benutzungsmodell benötigt (nach [Bal88]).

BitBlt *bit boundary block transfer.* (Siehe *RasterOp.*)

Bitmap Ein Bild, das aus einem Pixel–Array aufgebaut ist.

Bravo Ein bildschirmorientiertes WYSIWYG–Textbearbeitungsprogramm der Firma Xerox auf der Alto Workstation.

Browser Ein Instrument zur strukturierten Betrachtung der Zusammenhänge von Daten (*engl. to browse* stöbern). Ein Browser wird meist als Fenster einer Benutzerschnittstelle ausgeführt, in dem oft in verschiedenen Teilfenstern die Informationen über das untersuchte Objekt angezeigt werden (nach [Bal88]).

Button Ein Icon, das einer Taste nachgebildet ist. Durch Aktivieren des Buttons wird die dazugehörige Aktion ähnlich wie beim Betätigen einer Taste sofort ausgeführt.

Click Das kurzzeitige Drücken einer Taste innerhalb eines bestimmten Zeitintervalls, um ein Ereignis auszulösen.

Client Ein Applikationsprogramm, das den Window–Manager auffordert, bestimmte Aktionen auszuführen.

Cursor Eine sichtbare Darstellung der Eingabeposition auf einem Bildschirm, der den Bewegungen oder Befehlen des Eingabegerätes folgt.

Cut and Paste Die Fähigkeit, einen Bereich in einem Fenster einer Applikation zu selektieren und in einer anderen in einem weiteren Fenster aktiven Applikation einzusetzen. Bei Editoren entspricht dies dem Bewegen von Textbereichen innerhalb des aktuellen Dokuments oder zwischen zwei Dokumenten (z. B. Apple MacIntosh II: Zwischenablage).

Default Voreinstellung einer Datenstruktur, wenn kein Wert explizit angegeben ist.

Desktop Nachbildung (Metapher) einer Schreibtischoberfläche auf dem Bildschirm. Es werden meist Büroobjekte wie Dokumente, Ordner und Postkörbe mittels Text und Graphik dargestellt. Mit Hilfe eines Zeigeinstrumentes und der Tastatur bewegt, füllt, kopiert, bearbeitet und entfernt der Benutzer diese Objekte. Oft wird der *Schreibtisch* zum *Büro* erweitert, und es werden auch Objekte wie Papierkorb, Schränke, Kopierer und sogar Personen oder Personengruppen dargestellt (nach [GB88]).

Dialogverhalten Das Dialogverhalten ist die vom Benutzer subjektive empfundene Reaktion auf eine Aktion des Benutzers. Ein System sollte die gewohnte Arbeitsumgebung des Benutzers wiederspiegeln.

Direkte Manipulation Bearbeitung von Objekten ohne Einsatz einer Kommandosprache, z. B. Verschieben einer Datei in ein anderes Verzeichnis durch Bewegen eines Icons auf dem Bildschirm. Das Gefühl einer direkten Einflußnahme wird vermittelt (nach [Bal88]).

Expertensystem Programmsystem, das menschliches Expertenwissen, das durch Ausbildung und Erfahrung erworben wurde, verwendet, um in erklärungsfähiger Form Probleme zu lösen, die normalerweise menschliche Intelligenz erfordern (nach [Bal88]).

Fenster In einem konventionellen graphischen System der Teil des Bildes, der auf dem Bildschirm dargestellt wird. In einer Benutzeroberfläche ein virtueller Bereich, der einer Applikation zugeordnet wird und die mit dem Window–Manager in einem Bereich des Bildschirmes abgebildet wird.

Hilfesystem Teil eines Programmes, das zu vom Benutzer ausgewählten Aspekten Erklärungen bereitstellt. Dieses Programm kann passiv (auf Anforderung durch den Benutzer) oder aktiv (auf Grund der Erkenntnis, daß der Benutzer eine Erläuterung benötigt) reagieren. Statische Hilfe liegt vor, wenn die Antwort auf dieselbe Frage immer gleich ist. Dynamische Hilfe bezieht in die Beantwortung den aktuellen Kontext des Systems ein (nach [Bal88]).

Icon Ein kleines Piktogramm, das zur visuellen Darstellung von Fensterinhalten, Operationen, Operanden, Prozessen und Prozeßstati eingesetzt wird.

Kognitive Modellbildung Die typische Vorgehensweise einer sich konstruktiv verstehenden *Kognitionswissenschaft* (cognitive science), die sich aus der Überlappung von Informatik (speziell KI), kognitiver Psychologie und Linguistik ergibt. Das allgemeine Ziel ist, Denkprozesse durch Computermodelle zu simulieren. Dabei kommt es nicht allein darauf an, daß das Modell ein bestimmtes *intelligentes* Verhalten zeigt, sondern auch darauf, daß die im Modell verwirklichten internen Prozesse und Strukturen den in der Realität festgestellten oder vermuteten entsprechen, also *psychologisch valide* sind (nach [Bal88]).

Konsistenz Grad der strukturellen und inhaltlichen Übereinstimmung zwischen verschiedenen Teilen einer Benutzerschnittstelle, zwischen verschiedenen Anwendungsprogrammen oder zwischen verschiedenen Systemen. Hohe Konsistenz fördert das Erlernen und die Durchschaubarkeit von Benutzerschnittstellen (nach [Bal88]).

Look–Ahead–Mechanismus Bezeichnet einen Vorausschau–Mechanismus bei Erkennungsvorgängen. Für den Erkennungszweck kann es notwendig sein, daß eine gewisse Anzahl von Zeichen vorausgelesen wird (nach [GB88]).

Makro/Macro Unter einem Makro wird eine Zusammenfassung einer parametriesierten Befehlssequenz verstanden, die unter einem neuen Namen aufgerufen werden kann. Makros dienen der Verkürzung der Eingabezeit und der Betriebssicherheit.

Maus Eingabegerät für graphische interaktive Terminals. Die Bewegung der Maus wird in die Bewegung des Cursors auf dem Bildschirm übersetzt.

Mausklick siehe *Click*

Menü Ein Fenster, in dem eine Auswahl von Texten oder Graphiken angeboten wird. Durch Bewegen des Zeigers (Maus, Cursors) wählt der Benutzer eine Auswahl aus und aktiviert damit eine Aktion oder gibt den Parameter einer Aktion an.

Metafile Sequentielles File, das Segmente enthält (100 Segmente bei *UNIRAS*).

MetaFile–Handler Programm, das die Auswahl, Transformation und Ausgabe von bestimmten Segmenten eines bestimmten Metafiles auf diverse Ausgabegeräte ermöglicht.

Modell, mentales Ein mentales Modell ist die Vorstellung des Benutzers, die sich der Benutzer von seiner Arbeit mit dem Computer und von den Datenflüssen im Rechner macht. Kobsa zitiert drei Arten von Modellen: individuelles Modell, verallgemeinertes Modell zur Unterstützung des Softwareentwicklers und das Modell, das das Programm vom Benutzer generiert [Kob89]. Mentale Modelle werden insbesondere zur Verbesserung der Lernleistung eingesetzt.

Modell, operatives Operative Modelle repräsentieren die Vorstellung des Rechners vom Benutzer. Diese Vorstellung wird durch die Arbeit des Benutzers mit dem Rechner ständig verfeinert. Nach Kobsa wird ein Benutzungsmodell in ein Dialogsystem eingeführt, um dem Computersystem eine *natürliche* Verhaltensweise zu ermöglichen [Kob89]. Die Analyse der natürlichen Sprache bei der Eingabe von Fragen und Antworten steht hier im Vordergrund. Es werden Annahmen über Verhalten und Ziele des Benutzers formuliert und, aufgrund bereits vorhandenen Wissens, Inferenzen gebildet *(belief systems)*.

MVC–Konzept Das MVC–Konzept ist ein unter *Smalltalk-80* entwickeltes Konzept, Daten zu manipulieren.

Navigation Eine Form der interaktiven Informationssuche, bei der sich der Suchende schrittweise durch eine hierarchisch oder netzartig aufgebaute (in aller Regel komplexe) Informationsstruktur hindurchbewegt. Werkzeuge, die diesen Vorgang unterstützen, werden als Navigationswerkzeuge oder Browser bezeichnet (nach [GB88]).

Objekt Ein Objekt ist ein Modell einer realen oder abstrakten Sache, das durch Attribute bestimmt wird und das auf Nachrichten ein der Sache angemessene Verhalten, das durch Methoden bestimmt wird, zeigt.

Plotfile Datei mit geräteabhängigen Informationen zur Ansteuerung eines bestimmten Ausgabegerätes.

Pop–Up Menü Ein Menü, das auf Anforderung an der Cursor–Position erscheint und wieder gelöscht wird, wenn eine Auswahl selektiert wurde.

Prompter Ein Fenster, in dem der Benutzer gezielt (durch eine dargestellte Frage) zu einer speziellen Eingabe aufgefordert wird. Es werden Texteingabe (z. B. Dateiname beim Kreieren einer Datei) und Auswahl (z. B. *Ja/Nein*–Abfrage beim Löschen einer Datei) unterschieden.

Property Sheet Fenster, in dem Eigenschaften mittels verschiedener Auswahlmechanismen eingestellt werden können. Die Einstellungen erfolgen mit Auswahllisten, Schaltern, Texteingabe. Ein Property Sheet wird z. B. bei der Einstellung eines Fonts und der Fontgröße benutzt (nach [Bal88]).

Pull–Down Menü Eine Variante des Pop–Up Menüs, die erscheint, wenn in einem Fixed Menü die Maustaste an einem Menüpunkt gedrückt wurde. Ein Pull–Down Menü wird wie ein Rollo in den Bildschirm gezogen.

Rapid–Prototyping Bezeichnet eine bestimmte Methode der Software–Erstellung, die im Bereich der Künstlichen Intelligenz entstanden ist. Ausgangspunkt dieser Methode ist die Erkenntnis, daß es sich bei der Erstellung von Software um einen Entwurfsprozeß handelt, der am besten dadurch unterstützt wird, daß schon zu einem möglichst frühen Zeitpunkt Prototypen des Anwendungssystems existieren. An diesen Prototypen läßt sich die Spezifikation der Anwendung fortentwickeln (nach [GB88]).

RasterOp Eine Operation mit der rechteckige *BitMaps* überlagert werden. Jedes Pixel im Ziel–*BitMap* ist eine logische Kombination der korrespondierenden Pixel des Quell– und Ziel–*BitMaps*.

ReDo Bezeichnet eine Operation zur Wiederholung einer früher ausgeführten Operation, mit der ein mittels UnDo verlassener Systemzustand wiederhergestellt wird.

Scollbar Ein Bereich eines Fensters, mit dem das Verschieben des Fensterinhaltes kontrolliert wird.

Script Siehe auch *Macro*.

Segment Enthält eine Graphik in geräteunabhängiger Form. Ein Segment setzt sich aus graphischen Primitiven zusammen.

Segmentliste Liste von Segmenten, die zur sequentiellen Ausgabe erstellt wurde.

Selbsterklärungsfähigkeit Die Selbsterklärungsfähigkeit des Systems basiert auf grundlegenden Wissensbasen einer Benutzerschnittstelle wie Wissen über Visualisierung, Wissen über den Arbeitsablauf sowie die Arbeitsmittel des Benutzers und Wissen über den Kenntnisstand des Benutzers, bezogen auf das System. Da sich der Kenntnisstand des Benutzers ständig ändert und jeder Benutzer andere Verfahren zum Erreichen eines Zieles verwendet,

ist eine Anpassung des Systems an die Erfordernisse des Benutzers vorzusehen. Im Falle eines adaptiven Systems sollte der Benutzer vor der Änderung des Systemverhaltens um Zustimmung gebeten werden oder mindestens über die Änderung informiert werden, damit ein unkontrolliertes Verhalten mit nicht vorhersagbaren Ergebnissen vermieden wird (Verlust der Konsistenz).

Star Ein Büroautomationssystem der Xerox–Corporation, das als eines der ersten Systeme eine graphische Benutzeroberfläche integrierte.

Stereotyp Stereotypen stellen eine Sammlung von Attributen zur Verfügung, aus denen das System plausible Inferenzen auf der Basis von wenigen Beobachtungen generieren kann [Ric89]. Ein Benutzungsmodell enthält weiter eine Sammlung von Inferenzregeln, die aus den gegebenen Fakten, z. B. den Werten eines Stereotyps, Folgerungen ableiten. Stereotypen, die im Verlauf der Sitzung instanziiert und angepaßt werden, spiegeln Präferenzen des Benutzers wieder. Abbildung 1.1 zeigt zwei einfache Stereotypen für den Umgang mit dem Computer. Für spezielle Anwendungsfälle sind bereits Systeme im Einsatz, die mit Stereotypen arbeiten, so z. B. der *UC (UNIX Consultant)*, ein Hilfe–System für das Betriebssystem UNIX [Chi89].

Titlebar Feld am Kopf eines Fensters, in das die Applikation Informationen zur Kennzeichnung des Fenster einschreiben kann.

Transitionsnetz Ein Übergangsdiagramm. Es besteht aus Zuständen und markierten Zustandsübergängen, die zusätzlich noch mit Aktionen versehen sein können. Mit Transitionsnetzen lassen sich Grammatiken beschreiben. Transitionsnetze werden meist für Erkennungszwecke verwendet und dann von einem Automaten abgearbeitet, der je nach Eingabezeichen das Transitionsnetz durchläuft (nach [GB88]).

Tutorielle Systeme Unterstützen den Benutzer beim Erlernen der Mensch–Computer–Schnittstelle oder eines Anwendungssystems oder von Komponenten davon (nach [Bal88]).

UIMS Abkürzung für *User Interface Management System*. Ein *UIMS* ist die Software–Realisierung einer anwendungsneutralen Benutzerschnittstelle, die an mehrere auch unterschiedliche Anwendungsprogramme angekoppelt werden kann und damit die Wiederverwendung von Benutzerschnittstellen–Software ermöglicht. Die Konsistenz von Benutzerschnittstellen über mehrere Anwendungen läßt sich dadurch erhöhen. Der Begriff des *UIMS* ist an den Begriff *DBMS* (*Data Base Management System*) angelehnt, der für Datenbanken steht, die ebenfalls anwendungsneutrale Software–Komponenten darstellen (nach [GB88]).

UnDo Bezeichnet eine Operation, die Effekte einer früher ausgeführten Operation rückgängig macht. Bei einem zustandsorientierten UnDo kehrt das System in einen früheren Zustand zurück, beim funktionsorientierten UnDo wird eine inverse Operation ausgeführt, sofern eine inverse Operation existiert.

Viewport Ausschnitt einer Graphik im Gerätekoordinatensystem.

Window Ausschnitt einer Graphik im Weltkoordinatensystem. (Siehe auch Fenster.)

Wissensbasierte Systeme Programmsysteme, die menschliches Allgemeinwissen verwenden, um in erklärungsfähiger Form Probleme zu lösen, die normalerweise menschliche Intelligenz erfordern (nach [Bal88]).

Wissensbasis Die Funktionalität und das Verhalten von wissensbasierten Systemen ist durch die verwendeten Wissensbasen bestimmt. Sie modellieren jeweils einen abgegrenzten Bereich: Expertenwissen, Systemfunktionalität, Benutzungsmodelle etc. Aus technischer Sicht besteht eine Wissensbasis aus einer Kollektion von Wissenseinheiten, die gemeinsam verwaltet werden (nach [Bal88]).

Anhang C

Verzeichnisse

Zeichenerklärungen

$\top$	logisch wahr
$\bot$	logisch falsch
$\emptyset$	leere Menge
$\mathcal{P}(\mathcal{A})$	Potenzmenge von $\mathcal{A}$
$\forall$	für alle
$\exists$	es existiert ein
$\exists!$	es existiert genau ein
$\diamond$	Mengendifferenz
$\bigtimes_{a \in \mathcal{A}} \mathcal{A}$	Kreuzprodukt über die in $\mathcal{A}$ enthaltenen Elemente
$\bigcirc_{1 \leq i \leq n} f_i$	Hintereinanderausführung der Abbidungen $f_1 \ldots f_n$
$\longmapsto$	wird abgebildet auf
$\overset{\delta}{\longmapsto}$	wird durch δ abgebildet auf
$\hookleftarrow$	Zuweisung
$\bullet$	Zeichenkettenkonkatenation
fl	Vereinigung von Bäumen
$\circ$	Abbildungsverknüpfung
ι	Kennzeichnungsoperator
ε	leere Zeichenkette
κ	undefinierter Knoten
χ	undefinierter Zustand
δ	Zustandsänderung
τ	leerer Baum
id	identische Abbildung
$\prec$	zeitliches Aufeinanderfolgen in einem Pfad
$\succ\!\!\succ$	zeitliches Aufeinanderfolgen

Abkürzungsverzeichnis

ABS	AktionsBeschreibungsSprache
AI	Artificial Intelligence
ATN	Augmented Transition Network
AWI	Alfred-Wegener-Institut für Polar- und Meeresforschung
CCT	Cognitive Complexity Theory
CGM	Computer Graphics Metafile
DHB	Dialog History Baum
DTP	Desktop Top Publishing
FM	FabrikModel
FTP	File Transfer Program
GKS	Graphics Kernel System
GMD	Gesellschaft für Mathematik und Datenverarbeitung
GOMS	Goals, Operators, Methods, Selectors
GUMS	General User Modelling Shell
GWUIMS	Georg Washington User Interface Management System
HBE	History Baum Element
HCI	Human Computer Interaction
HUFIT	HUman Factor Laboratories Information Technologie
IMSL	Mathematik and Statistic Library
IPSI	Institut für Integrierte Publikations und Informationssysteme Integrated Publication and Information Systems Institute
KI	Künstliche Intelligenz
LAN	Local Area Network
LSE	Language Sensitive Editor
NAG	Numerical Arithmetic Group
NCS	Network Computing System
NeWS	Network extensible Window System
NIHBIS	NetworkIconHBInternalState
NFS	Network File System
OOPS	Object Oriented Programming System
PARC	Palo Alto Research Centre of Xerox Corporation
PHIGS	Programmers HIerarchical Graphic System
PK	PapierKorb
RFA	Rolle Funktion Aktion
RMS	ResourcenManagementSystem
SFS	SampleFileSystem
TAG	Task Aktion Grammar
TECO	Text Editor and COrrector
TOP	Task Oriented Parsing

UC	UNIX Consultant
UCM	User Conceptual Model
UIMS	User Interface Management System
UNIX	Betriebssystem
VMS	Betriebssystem der VAX-Rechner von Digital Equipment
WAN	Wide Area Network
WAP	Wissenschaftlicher Arbeitsplatz
WYSIWYG	What You See Is What You Get

Literaturverzeichnis

[ACI89] ACI, Paris, *4^{th}–Dimension User Manual*, 1989

[ACS84] J. E. Jr. Archer, R. Conway, F. B. Schneider, User Recovery and Reversal in Interactive Systems, *ACM Transactions: Programming Languages and Systems*, 6(1):1–19, 1984

[ANS84] ANSI, An American National Standard: IEEE Guide to Software Requirements Specifications, Technischer Bericht, American National Standards Institute ANSI/IEEE, 1984

[AW86] C. M. Allwood, T. Wikström, Learning Complex Computer Programs, *Behaviour and Information Technology*, 5(3):217–225, 1986

[Ack84] D. Ackermann, *Untersuchungen zum individualisierten Computerdialog: Einfluß des operativen Abbildsystems auf Handlungs– und Gestaltungsspielraum und die Arbeitseffizienz*, Kognitive Aspekte der Mensch-Computer-Interaktion, 1984

[Ack88] D. Ackermann, Empirie des Softwareentwurfs: Richtlinien und Methoden, In H. Balzert, H. U. Hoppe, R. Oppermann, H. Peschke, G. Rohr, N. A. Streitz (Hrsg.), *Einführung in die Software–Ergonomie*, Seiten 257–260, de Gruyter, 1988

[All89] Alliant Inc., Corp, Littleton, Mass., USA, *UNIX Reference*, 1989

[App86] Apple Computer Inc., *Human Interface Guidelines: The Apple Desktop Interface*, Addison–Wesley, Reading, Mass., 1986

[Arb87] ArborText, Inc., *The Publisher User Manual*, 1987

[Are89a] U. Arend, Analysing complex tasks with an extended *GOMS*–Model, In D. Ackermann, M. Tauber (Hrsg.), *Mental Models and Human–Computer Interaction*, North Holland Publishing Company, Amsterdam, 1989

[Are89b] U. Arend, Einfluß von Visualisierung und Kommandostruktur auf das Problemlösen an einer Prototypendatenbank, In S. Maaß, H. Oberquelle (Hrsg.), *Software–Ergonomie '89: Aufgabenorientierte Systemgestaltung und Funktionalität*, Seiten 355–364, BG Teubner, Stuttgart, 1989

[BBG+89] D. Bauer, H. D. Böcker, R. Gunzenhäuser, H. v. d. Herberg, D. Maier, M. Rathke, M. Ressel, T. Schwab, Einsatz einer anwendungsneutralen Benutzerschnittstelle in einer Büroanwendung als Beispiel für wissensbasierte Mensch–Computer–Kommunikation, *Angewandte Informatik*, (7):294–300, 1989

[BCG+87] J. Banerjee, H.-T. Chou, J. F. Garza, W. Kim, D. Woelk, N. Ballou, H.-J. Kim, Data Model Issues for Object–Oriented Applications, *ACM Transactions on Office Information Systems*, 5(1), 1987

[BG82] F. L. Bauer, G. Goos, *Informatik, Erster Teil*, Heidelberger Taschenbücher 80, Springer–Verlag, Heidelberg, 1982

[BHJ87] A. Balzert, H.U. Hoppe, Ziegler J., *Fenstersysteme im Vergleich – Architektur, Leistungsfähigkeit und Eignung für die Anwendungsentwicklung*, 1987

[BMH85] M. L. (Hrsg.) Brodie, J. (Hrsg.) Mylopoulos, Schmitt (Hrsg.), *On Conceptual modelling*, Springer, New York, 1985

[BS87] J. Bauer, T. Schwab, Propositions in Help–Systems, *Angewandte Informatik*, (1):23–29, 1987

[BSN86] D. P. Browne, B. D. Sharratt, M. A. Norman, The Formal Specification of Adaptive User Interfaces Using Command Language Grammar, In *CHI'86 Proceedings*, Seiten 256–260, 1986

[Bal88] H. Balzert, Trends und Perspektiven der Software–Ergonomie, In H. Balzert, H. U. Hoppe, R. Oppermann, H. Peschke, G. Rohr, N. A. Streitz (Hrsg.), *Einführung in die Software–Ergonomie*, de Gruyter, Berlin, 1988

[Bec90] G. Becker, Weitblick, SiATEX – eine Arbeitsumgebung für TEX, *iX – Multiuser – Multitasking – Magazin*, (1), 1990

[Blu89] Blue Sky Research, *TEXtures User's Guide*, 1989

[Bö84] T. Bösser, Lernanforderungen als Gestaltungsgrundlage für die Mensch–Maschine–Schnittstelle eines Rechners, In *Kognitive Aspekte der Mensch-Computer-Interaktion*, Seiten 69–75, Springer Verlag, Heidelberg, 1984

[Bö87] T. Bösser, *Learning in Man–Computer Interaction*, Springer Verlag, Heidelberg, 1987

[Bro86] M. L. Brodie, On the Development of Data Models, In M. L. Brodie, J. Mylopoulos, J. W Schmidt (Hrsg.), *On Conceptual Modelling*, Springer, New York, 1986

[CM80] S. T. Card, T. P. Moran, The KeyStroke–Level Model for User Performances Time with Interactive Systems, *Communications of the ACM*, 23(7):396–411, 1980

[CMN83] S. T. Card, T. P. Moran, A. Newell, *The Psychology of Human Computer Interaction*, Lawrence Erlbaum Association, London, 1983

[CR85] J. M. Carrol, M. B. Rosson, Usability Specifications as Tool in Iterative Developement, In R. Hartson (Hrsg.), *Advances in Human–Computer–Interaction*, Ablex Publishing Corporation, New Jersey, USA, 1985

[Car88] S. Carberry, Modelling The User's Plans And Goals, *Computational Linguistics*, 14(3):23–37, 1988

[Car89] S. Carberry, Plan Recognition and Its Use in Understanding Dialog, In A. Kobsa, W. Wahlster (Hrsg.), *User Models in Dialog Systems*, Seiten 131–162, Springer Verlag, Heidelberg, 1989

[CdV89] T. Christaller, F. di Primo, A. (Hrsg) Voss, *Die KI–Werkbank Babylon*, Addision–Wesley, 1989

[Cen90] The Ohio Supercomputer Center, *The aPe Reference Guide*, The Ohio State University, Columbus, Ohio, USA, 2.0 Auflage, 1990

[Chi85] U. Chi, H., Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches, *IEEE Transactions on Software Engeneering*, SE–11:8–20, 1985

[Chi89] D. N. Chin, KNOME: Modelling What the User Knows in UC, In A. Kobsa, W. Wahlster (Hrsg.), *User Models in Dialog Systems*, Seiten 75–107, Springer Verlag, Heidelberg, 1989

[Coa89] P. Coad, *OOA: Object–Oriented Analysis, Tutorium der OOPSLA '89*, ACM, New Orleans, 1989

[DFG89] DFG, DFG–Kriterien zum Arbeitsplatzrechner für Wissenschaftler, Technischer Bericht, Deutsche Forschungsgemeinschaft, 1989, Anlage zum WKMS Nr. V/6 – 5a/24 486

[Dat88] DataEase International, Trumbull, USA, *DataEase Reference Manual*, 1988

[Dig86a] Digitalk Inc., *Smalltalk/V 286:* Object–Oriented Programming System (OOPS), *Tutorial and Programming Handbook*, 1986

[Dig86b] Digitalk Inc., *Smalltalk/V: Object–Oriented Programming System (OOPS), Tutorial and Programming Handbook*, 1986

[Dig88] Digitalk Inc., *Smalltalk/V Mac:* Object–Oriented Programming System (OOPS), *Tutorial and Programming Handbook*, 1988

[Dig90] Digital Equipment Corporation, *VMS User Manual*, 1990

[Dre72] H. L Dreyfus, *Was Computer nicht können: Grenzen der künstlichen Intelligenz*, Atheneum Verlag, Frankfurt, 1972, 1989

[DvBF+84] A. Dirlich, H. v. Benda, C. Freska, D. Furbach, E. Müller, F Wimmer, Computerunterstützte Planung von Ferienreisen: ein fiktives Beispiel, In *Kognitive Aspekte der Mensch-Computer-Interaktion*, Seiten 22–30, Springer Verlag, 1984

[FG86] G. Fischer, R. Gunzenhäuser, *Methoden und Werkzeuge zur Gestaltung benutzergerechter Computersysteme*, de Gruyter, Berlin, 1986

[FGV93] P. Forbrig, P. Gorny, A. Viereck, Unterstützung des Software–Design–Prozesses durch EXPOSE, 1993

[FM86] W. L. Fuerst, M. P. Merle, Using Computer Knowledge in the Design of Interactive Systems, *JMMS*, (26):333–342, 1986

[FW89] H. Fleischhack, A. Weber, Rule Based Programming, Predicate Transition Nets and the Modelling of Office Procedures and Flexible Manufacturing Systems, In *10th Int. Conf. on Application and Theory of Petri–Nets, Bonn*, 1989

[Fro87] M. Fromme, Werkzeuge für eine einheitliche Benutzerschnittstelle von Programmsystemen in der wissenschaftlichen Datenverarbeitung, Instituts–Bericht: Bereich Datenverarbeitung und Elektronik, Hahn–Meitner–Instituts, Berlin, 1987

[FvD82] J. D. Foley, A. van Dam, *Fundamentals of Interactive Computer Graphics*, awv, 1982

[GB88] R. Gunzenhäuser, H.-D. Böcker, *Prototypen benutzergerechter Computersysteme*, de Gruyter, Berlin, 1988

[GJST88] R. A. Gargan Jr., J. W. Sullivan, S. W. Tyler, Multimodal Response Planning: an Adaptive Rule Based Approach, In *Proceedings to CHI'88*, Seiten 229–234, ACM Press, 1988

[GR83] A. Goldberg, D. Robson, *Smalltalk–80: The Language and its Implementation*, Addison–Wesley, 1983

[GW88] S. Greenberg, H. Witten, How Users Repeat Their Actions on Computers: Principles for Design of History Mechanisms, In *Proceedings to CHI'88*, Seiten 171–178, ACM Press, 1988

[Gol84] A. Goldberg, *Smalltalk–80: The Interactive Programming Environment*, Addison–Wesley, 1984

[H+87] W. Hiller et al., Rechnerkonzept des Alfred–Wegener–Instituts (AWI), Interner Bericht, Alfred–Wegener–Institut, 1987

[HA86] J. Hendler, P. A., Wegner, Viewing Object–oriented Programming as an Enhancement of Data Abstraction Methodology, In *Proceedings of the Nineteenth Annual Hawaii International Conference on System Science, 1986*, 1986

[HKST87] H. W. Hein, G. M Kellermann, S. R. Smith, C. G. Thomas, A Shell for Building Highly Interactive and Adaptive User Interfaces, interne Veröffentlichung, Gesellschaft für Mathematik und Datenverarbeitung, Birlinghoven, 1987

[HP89] H. U. Hoppe, R. Ploetzner, Inductive Knowledge Acquisition for a UNIX Coach, In D. Ackermann, M. Tauber (Hrsg.), *Mental Models and Human–Computer–Interaction II*, 1989, (in Vorbereitung)

[HS88] L. Hertweck, W. Stöhr, Graphik am Wissenschaftlerarbeitsplatz: Dokumenterstellung und Dokumentaustausch, In *Graphik im Bürobereich GI–Fachgespräch, Bad Honnef*, 1988

[HTZ86] H. U. Hoppe, M. Tauber, J. E. Ziegler, A Survey of Models and Formal Description Methods in HCI with Example Applications, Report B3.2a, 1986, ESPRIT PROJECT HUFIT

[Hee88] F. J. Heeg, *Empirische Software Ergonomie*, Springer Verlag, 1988

[Her86a] M. Herczeg, *Eine objektorientierte Architektur für wissensbasierte Benutzerschnittstellen*, Dissertation, Universität Stuttgart, 1986

[Her86b] T. Herrmann, *Zur Gestaltung der Mensch–Computer–Interaktion: Systemerklärung als kommunikatives Problem*, Max Niemeyer Verlag, Tübingen, 1986

[Her87] T. Herrmann, Beiträge des Bereiches Informatik und Gesellschaft 1986/87, Themen: Softwareergonomie, psychologische Auswirkungen, Forschungsbericht Nr. 243, 1987, Universität Dortmund, 1987

[Hop88a] H. U. Hoppe, Task–Oriented–Parsing: A Diagnostic Method to be Used by adaptive Systems, In *Proceedings to CHI'88*, Seiten 241–247, ACM Press, 1988

[Hop88b] H. U. Hoppe, Werkzeuge für die Prototypen–Entwicklung von Benutzerschnittstellen, In H. Balzert, H. U. Hoppe, R. Oppermann, H. Peschke, G. Rohr, N. A. Streitz (Hrsg.), *Einführung in die Software–Ergonomie*, de Gruyter, Berlin, 1988

[Hop89] H. U. Hoppe, Automatic Acquisition of Procedural Task Knowledge for an Intelligent Help System, In 2^{nd} *International Symposium on Artificial Intelligence (ITESM), Monterry, Mexico*, 1989

[IWC+88] D. Ingalls, S. Wallace, Y.Y. Chow, F. Ludolph, D. Doyle, A Visual Programming Environment, In *Proceedings OOPSLA'88*, ACM Press, 1988

[KF88a] R. Kass, T. Finin, A General User Modelling Facility, In *Proceedings to CHI'88*, Seiten 145–150, ACM Press, 1988

[KF88b] R. Kass, T. Finin, Modelling the User in Natural Language Systems, *Computational Linguistics*, 14(3):5–22, 1988

[KHTF87a] G. M. Kellermann, H. W. Hein, C. G. Thomas, E. Finke, Semantische Protokolle und Handlungspläne in einer universellen Mensch–Computer–Schnittstelle, *KI*, (4), 1987

[KHTF87b] G. M. Kellermann, H. W. Hein, C. G. Thomas, E. Finke, X–Aid: Eine wissensbasierte, anwendungsunabhängige Mensch–Computer–Schnittstelle, *GMD Spiegel*, 87(1):23–30, 1987

[KLS85] F. M. Kühn, W. Laurig, K.-H. Schmidt, Ein Modell zur Erklärung von Leistungen und Stategien bei Eingaben über Tastaturen, In *Softwareergonomie '85*, Stuttgart, 1985, BG Teubner

[KMP93] L.-P. Kurdelski, S. Makedanz, G. Praetsch, TEXDesktop, Eine Benutzungsoberfläche für TEX, Addison–Wesley, 1993

[KP85] D. Kieras, P. G. Polson, An approach to the Formal Analysis of User Complexity, *Int. Journal of Man–Machine Studies*, (22):365–394, 1985

[KS89] R. Keil-Slawik, Systemgestaltung mit Aufgabennetzen, In S. Maaß, H. Oberquelle (Hrsg.), *Software–Ergonomie '89: Aufgabenorientierte Systemgestaltung und Funktionalität*, Seiten 123–133, BG Teubner, Stuttgart, 1989

[Knu84] D. E. Knuth, *The TEXbook*, Addison–Wesley, Reading, Mass., 1984

[Kob84] A. Kobsa, Ergänzung und Korrektur des mentalen Modells durch Rechnerunterstützung, In *Kognitive Aspekte der Mensch-Computer-Interaktion*, Seiten 154–160, Springer Verlag, Heidelberg, 1984

[Kob85] A. Kobsa, *Benutzermodellierung in Dialogsystemen, Informatik Fachberichte 115, KI*, Springer Verlag, Heidelberg, 1985

[Kob89] A. Kobsa, A Taxonomy of Beliefs and Goals for User Models in Dialog Systems, In A. Kobsa, W. Wahlster (Hrsg.), *User Models in Dialog Systems*, Seiten 52–68, Springer Verlag, Heidelberg, 1989

[Kun90] K. Kunkel, Visualization an Direct Manipulation in User Interfaces: Are we Overdoing It?, In P. Gorny, M. Tauber (Hrsg.), *Visualization in Human–Computer–Interaktion*, Seiten 183–193, Springer Verlag, Heidelberg, 1990

[Lam85] L. Lamport, LATEX, *A Document Preparation System*, Addison Wesley, Reading Mass., 1985

[MaA85] M. Mac an Airchinnigh, Report on the User's Conceptual Model, In G. E. Pfaff (Hrsg.), *User Interface Management Systems*, Seiten 31–40, Springer, New York, 1985

[Mak90] S. Makedanz, Programmieren in Smalltalk – Portierung einer Benutzungsoberfläche für TEX, Diplomarbeit, Hochschule Bremerhaven, 1990

[Mic87] Microsoft Inc., *Microsoft Word User Manual*, 1987

[Mor81] T. P. Moran, The Command Language Grammar: A representation for the user interface of interactive computer systems, *Int. Journal of Man–Machine Studies*, (15):3–20, 1981

[Nat] National Instruments, *LabView User Manual*

[Neh87] J. Nehmer, *Konstruktionsprinzipien für konkurrente verteilte Systeme*, Fernuniversität Hagen, Fernuniversität – Gesamthochschule, Hagen, 1987

[New72] A. Newell, Production Systems: Models of Control Structures, In W. G. Chase (Hrsg.), *Visual Information Processing*, Seiten 463–526, Academic Press, 1972

[Obe88] H. Oberquelle, *Sprachkonzepte für benutzergerechte Systeme, Informatik Fachbericht 144*, Springer Verlag, Hamburg, 1988

[Opi89] G. Opitz, Konzipierung einer graphischen Benutzeroberfläche für ein geographisches Informationssystem und Entwicklung eines Prototyps, Diplomarbeit, Hochschule Bremerhaven, 1989

[PS89] M. Plaschke, R. Schnars, Metafile–Interpreter, Diplomarbeit, Hochschule Bremerhaven, 1989

[PW89] L. J. Pinson, R. S. Wiener, *Object–oriented Programming and Smalltalk*, Addison–Wesley, Reading, MA, 1989

[Pfa85] G. Pfaff (Hrsg.), *User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim, FRG*, Heidelberg, 1985, Springer Verlag

[Pie87] K. W. Piersol, *HUMBLE V. 2.0 Reference Manual*, XEROX Special Information Systems, Pasadena, 1987

[Pra90] G. Praetsch, Entwicklung einer interaktiven, sensitiven Benutzungoberfläche zum Edieren von TEX–Dokumenten, Diplomarbeit, Hochschule Bremerhaven, 1990

[Pro] Prograph, *Prograph User Manual*

[RM83] A. Roberts, T.P. Moran, The Evaluation of Texteditors: Methodology and empirical Results, *Communications of the ACM*, 26(4,):265–283, 1983

[Rat86] M. Rathke, WYSIBIB: Einsatz von Undo/Redo–Mechanismen bei der Verwaltung einer Literaturdatenbank, In R. Gunzenhäuser, H.-D. Böcker (Hrsg.), *Prototypen benutzergerechter Systeme*, de Gruyter, 1986

[Rat89] M. Rathke, Erweiterung interaktiver Anwendungen um Undo–Mechanismen, In S. Maaß, H. Oberquelle (Hrsg.), *Software–Ergonomie '89: Aufgabenorientierte Systemgestaltung und Funktionalität*, Seiten 274–283, BG Teubner, Stuttgart, 1989

[Rei85] W. Reisig, *Systementwurf mit Netzen*, Springer Verlag, 1985

[Rei92] M. Reinke, Ein persistenter Speicher für Smalltalk–80 Objekte auf der Basis eines relationalen Datenbanksystems, Diplomarbeit, Fernuniversität Hagen, 1992

[Ric84] L. Richter, *Betriebssysteme*, BG Teubner, Stuttgart, 1984

[Ric89] E. Rich, Stereotypes and User Modelling, In A. Kobsa, W. Wahlster (Hrsg.), *User Models in Dialog Systems*, Seiten 35–51, Springer Verlag, Heidelberg, 1989

[Roc89] B. Rockel, *ROTEX, Bedienungshandbuch und Makrosammlung*, Institut für Geophysik und Meteorologie, Universität zu Köln, W–Germany, 1989

[S+86] J. L. Silbert et al., An Object–Oriented User Interface Management System, *Computer Graphics*, 20(4):259–267, 1986

[SA77] R. C. Schank, R. P. Abelson, *Scripts, Plans, Goals, and Understanding*, Erlbaum, Hillsdale, New York, 1977

[SH86] M. Schneider-Hufschmidt, *Software–Entwicklung in einer objektorientierten Programmierumgebung*, Dissertation, Universität Stuttgart, 1986

[SIK+83] D. C. Smith, C. Irby, R. Kimball, B. Verplank, E. Harslem, Designing the Star User Interface, *Integrated Interactive Computersystems*, 1983

[SM77] B. Shneiderman, R. Mayer, Syntactic/Semantic Interactions in Programmers Behaviour: A Model and Experimantal Results, *Int. Journal of Man–Machine Studies*, 8(3):219–238, 1977

[SM88] S. Shlaer, S. M. Mellor, *Object–Oriented Systems Analysis*, Yourdon Press, Englewood Cliffs, N.J., 1988

[SS89] G. Schmidt, T. Ströhlein, *Relationen und Graphen*, Springer Verlag, Heidelberg, 1989

[SSNB84] H. Sugaya, J. Stelovsky, J. Nievergelt, E.S. Biagoni, XS–2: An Integrated Interactive System, Forschungsbericht, Brown Boveri Forschungszentrum, Baden, Schweiz, 1984, KLR 84–73C

[SSS88] S. Sippu, E. Soisalon-Soininen, *Parsing Theory, Vol. 1: Languages and Parsing*, Springer, New York, 1988

[SW88] F. Schmalhofer, T. Wetter, Kognitive Modellierung: Menschliche Wissensrepräsentationen und Verarbeitungsstrategien, In Th. Christaller, H.-W. Hein, M. M. Richter (Hrsg.), *Künstlicher Intelligenz Informatikfachbericht 159*, Springer Verlag, Heidelberg, 1988

[Sch89a] T. Schwab, Dialogmodellierung in interaktiven Systemen, In S. Maaß, H. Oberquelle (Hrsg.), *Software-Ergonomie '89: Aufgabenorientierte Systemgestaltung und Funktionalität*, Seiten 174–183, BG Teubner, Stuttgart, 1989

[Sch89b] T. Schwab, *Methoden zur Dialog- und Benutzermodellierung in adaptiven Computersystemen*, Dissertation, Universität Stuttgart, 1989

[Sha86] M. Shaw, The Impact of Modelling and Abstraction Concerns on Modern Programming Languages, In M. L. Brodie, J. Mylopoulos, J. W Schmidt (Hrsg.), *On Conceptual Modelling*, Springer, New York, 1986

[Sha87] S. C. (Hrsg.) Shapiro, *Encyclopedia of Artificial Intelligence Vol. 1 und 2*, Wiley & Sons, New York, 1987

[Shn83] B. Shneiderman, Direct Manipulation: A Step Beyond Programming, *Computer*, 16(8):57–69, 1983

[Shn85] B. Shneiderman, *Designing the User Interface*, Addison–Wesley, 1985

[Sta89] C. Stary, Das Interaktions–Management–Netz: Ein Beitrag zur konzeptionellen Modellierung der Mensch–Computer–Interaktion, *Informatik Forschung und Entwicklung*, (4):193–204, 1989

[Ste84] J. Stelovsky, *XS-2: The User Interface of an Interactive System*, Dissertation, ETH Zürich, 1984

[Str89] A. Streitz, N., Fragestellungen und Forschungsstrategien der Software–Ergonomie, In H. Balzert, H. U. Hoppe, R. Oppermann, H. Peschke, G. Rohr, N. A. Streitz (Hrsg.), *Einführung in die Software-Ergonomie*, de Gruyter, Berlin, 1989

[Syb89] Sybase Inc, Emeryville, Califonia, USA, *Sybase User Manual*, 1989

[TFK87] C. G. Thomas, E. B. Finke, G. W. Kellermann, Aid: Ein wissensbasierter Ansatz für die adaptive Mensch–Computer–Schnittstelle, In *Informatikfachberichte 156*, Springer Verlag, Heidelberg, 1987

[TKH86] C. G. Thomas, G.M. Kellermann, H.W. Hein, X–Aid: An adaptive and knowledgebased Human Computer Interface, In *Human–Computer–Interaction – INTERACT '87*, Seiten 1075–1080, 1986

[TO90] R. Tuck, D. R. Olsen, Help by Guided Tasks: Utilizing UIMS Knowledge, In *CHI'90 Proceedings*, 1990

[Thi90] H. Thimbleby, *User Interface Design*, ACM Press, New York, 1990

[VS89] G. Vouros, C. D. Spyropoulos, A Methodology for Conceptual Representation of Knowledge Using Attribute Grammars, *Angewandte Informatik*, (7), 1989

[Vet90] M. Vetter, *Aufbau betrieblicher Informationssysteme mittels konzeptioneller Datenmodellierung*, BG Teubner, Stuttgart, 1990

[Vie93] A. Viereck, Design von Benutzungsoberflächen als ingenieursmäßiger Prozeß, 1993

[Vit84] J. S. Vitter, *US&R:* A new Framework for Redoing, *IEEE Software*, 1(4):39–52, 1984

[WB84] H. Willmer, H. Balzert, *Fallstudie einer industriellen Software-Entwicklung: Definition, Entwurf, Implementierung, Abnahme, Qualitätssicherung*, Bibliographisches Institut, Mannheim, 1984

[Wal89] K. Waldhör, Wissensbasiertes Generieren von Benutzerschnittstellen, In S. Maaß, H. Oberquelle (Hrsg.), *Software-Ergonomie '89: Aufgabenorientierte Systemgestaltung und Funktionalität*, Seiten 294–303, BG Teubner, Stuttgart, 1989

[Wei87] J. Weiß, Konzeption einer Benutzerschnittstelle für ein Informationssystem mit relationaler Datenbank und rechnerunerfahrenem Benutzerkreis, Diplomarbeit, Universität Paderborn, 1987

[Wei90] J. Weizenbaum, *Die Macht der Computer und die Ohnmacht der Vernunft*, Suhrkamp Verlag, Frankfurt, 8. Auflage, 1990

[Wes87] S. Westerwick, Einsatz eines Datenstrukturwörterbuches für relationale Datenbanken, Diplomarbeit, Universität Paderborn, 1987

[Wir75] N. Wirth, *Systematisches Programmieren*, Technische Studienbücher Informatik, BG Teubner, 2. Auflage, 1975

[Xer87] Xerox XSIS, *The Analyst V2.1 Programmers Guide*, 1987

[Xer89] Xerox XSIS, *The Analyst V3.0 User Guide*, 1989

[YM88] R. M. Young, A. MacLean, Choosing Between Methods: Analysing the Users's Decision Space in Terms of Schemas and Linear Models, In *Proceeding to CHI'88*, Seiten 139–143, ACM Press, 1988

[Yan88a] Y. Yang, A New Conceptual Model for Interactive User Recovery and Command Reuse Facilities, In *Proceedings to CHI'88*, Seiten 165–170, ACM Press, 1988

[Yan88b] Y. Yang, Undo support models, *Int. Journal of Man-Machine Studies*, (28):457–481, 1988

[Zie88] J. Ziegler, Aufgabenanalyse und Funktionsentwurf, In H. Balzert, H. U. Hoppe, R. Oppermann, H. Peschke, G. Rohr, N. A. Streitz (Hrsg.), *Einführung in die Software–Ergonomie*, de Gruyter, Berlin, 1988

[iD84a] Normenausschuß Informationsverarbeitungssysteme im DIN, *Bildschirmarbeitsplätze: Codierung von Information*, Deutsches Institut für Normung e.V., DIN 66234 Teil 5 Auflage, 1984

[iD84b] Normenausschuß Informationsverarbeitungssysteme im DIN, *Bildschirmarbeitsplätze: Grundsätze ergonomischer Dialoggestaltung*, Deutsches Institut für Normung e.V., DIN 66234 Teil 8 Auflage, 1984

[vB84] H. v. Benda, Sachprobleme, Interaktionsprobleme und die Rolle des Benutzers, In *Kognitive Aspekte der Mensch-Computer-Interaktion*, Seiten 37–40, Springer Verlag, Heidelberg, 1984