In: Bullinger, H.-J. (Hrsg.): Software-Ergonomie '85: Mensch-Computer-Interaktion.
Stuttgart : B.G. Teubner. S. 108-118

Software-Ergonomie durch wissensbasierte Systeme

Joachim Bauer und Michael Herczeg

Forschungsgruppe INFORM[1]
Institut für Informatik
Universität Stuttgart
Herdweg 51
7000 Stuttgart 1

In diesem Beitrag soll dargestellt werden, welche Eigenschaften Benutzerschnittstellen haben sollten, die das Prädikat "ergonomisch" verdienen. Die uns besonders wichtig erscheinenden Eigenschaften sind:

* **Konsistenz**
* **Erklärungsfähigkeit**
* **Gestaltbarkeit**
* **flexible Fehlerbehandlung**

Untersucht man diese Eigenschaften genauer, so zeigt sich, daß das System zur Verwirklichung dieser Eigenschaften auf umfangreichem, verschiedenartigem Wissen aufbauen muß.

1. Konsistenz der Interaktion

Arbeitet ein Benutzer mit einem System, so erwartet er daß gleichartige Vorgänge durch gleichartige Interaktionen bewirkt werden können. Die Verläßlichkeit des Dialogverhaltens ist eine Voraussetzung für unbelastetes Arbeiten. Die Verläßlichkeit von Systemen, man könnte sie als **innere Konsistenz** bezeichnen, ist leider nicht immer gewährleistet, da das Verhalten des Systems häufig an mehreren Stellen des Programms festgelegt wird, die sich nur mühevoll konsistent halten lassen. Bei der Konzeption wissensbasierter Systeme wird darauf geachtet, daß das Wissen (und damit auch das Verhalten) des Systems in klar isolierbaren Teilen festgehalten ist, auf die von verschiedenen Stellen Bezug genommen werden kann. Die Erhaltung der Konsistenz wird dadurch garantiert. Bei der Benutzung verschiedener Anwendungssysteme ist es durchaus nicht die Ausnahme, daß sich der Benutzer verschiedene Bedienungsmethoden aneignen muß: er lernt jedes dieser Systeme wieder nahezu von Grund auf. Die Wahrung der **Konsistenz zwischen mehreren**

[1]*Die Forschungsgruppe INFORM wird im Rahmen des Verbundprojekts WISDOM vom Bundesminister für Forschung und Technologie gefördert.*

Anwendungssystemen ist ein großes Problem. Was benötigt wird, sind **integrierte Systeme**, die sich derselben Interaktionstechnik, unabhängig von der Anwendung, bedienen. Eine Lösung sind **anwendungsneutrale Benutzerschnittstellen**, die den Zugang zu allen benutzten Anwendungssystemen in derselben Art und Weise ermöglichen [Dzida 83; Hayes, Szekely, Lerner 85]. Die Benutzerschnittstelle wird zu einer weitgehend eigenständigen Systemkomponente, in der anwendungsneutrales Wissen über Methoden und Regeln der Kommunikation repräsentiert wird. Die Isolation dieses Wissens über die Kommunikation macht die Benutzerschnittstelle nicht nur für den Systemdesigner überschaubarer und leichter änderbar, sondern erlaubt auch die Modifikation der Benutzerschnittstelle durch den Benutzer selbst (siehe Abschnitt 3).

Konsistenz mit der bekannten manuellen Arbeitsumgebung bieten sogenannte **metaphorische Systeme**. Bei dieser Art von Systemen wird auf dem Bildschirm die Arbeitsumgebung nachgebildet. Dazu werden auf dem Bildschirm die gewohnten Arbeitsobjekte (z.B. Dokumente, Ordner, Schränke, Papierkorb) als **Icons** (Piktogramme) dargestellt. Die Bildschirmfläche stellt den Schreibtisch dar. Mit Hilfe eines Zeigeinstruments können diese Objekte dann ausgewählt, bewegt und mit einem Editor bearbeitet werden [Smith et al. 82]. Die Wirkungen sind sofort sichtbar und damit kontrollierbar. Man spricht deshalb auch von **direkter Manipulation** [Shneiderman 83] oder dem **WYSIWYG-Prinzip** (What You See Is What You Get). Häufig werden für die unterschiedlichen Bildschirmdarstellungen und Arbeitskontexte verschiedene Fenster verwendet, die sich auf dem Bildschirm gegenseitig überlappen können (siehe Abbildung 1). Die Konstruktion als wissensbasierte Systeme verhindert, daß metaphorische Systeme mehr scheinen als sie wirklich sind. Die auf dem Bildschirm dargestellten Objekte müssen für den Benutzer verständlich sein und aktuelle Bezüge zur Anwendung herstellen. Werden sie manipuliert, müssen die durch sie repräsentierten Arbeitsobjekte mitverändert werden. Das Kopieren eines Dokuments auf dem Bildschirm, muß also auch das Kopieren der entsprechenden internen Informationsstruktur bewirken.

2. Hilfeleistungen

Hilfeleistungen durch das System sollten nicht dazu dienen, Mängel des Systems auszugleichen, die auf einer Fehlkonzeption beruhen. Diese Mängel sollten durch konstruktive Maßnahmen vermieden werden, wie sie in den anderen Abschnitten dieses Beitrags beschrieben werden. Allerdings lassen sich komplizierte Anwendungen, die sich durch hohe Funktionalität oder Funktionen mit vielen verschiedenen Parametern auszeichnen, nicht immer durch leicht verständliche Systeme realisieren. Beispielsweise können metaphorische Systeme den Dialog zwar durchsichtiger machen; da man aber aus einem Namen

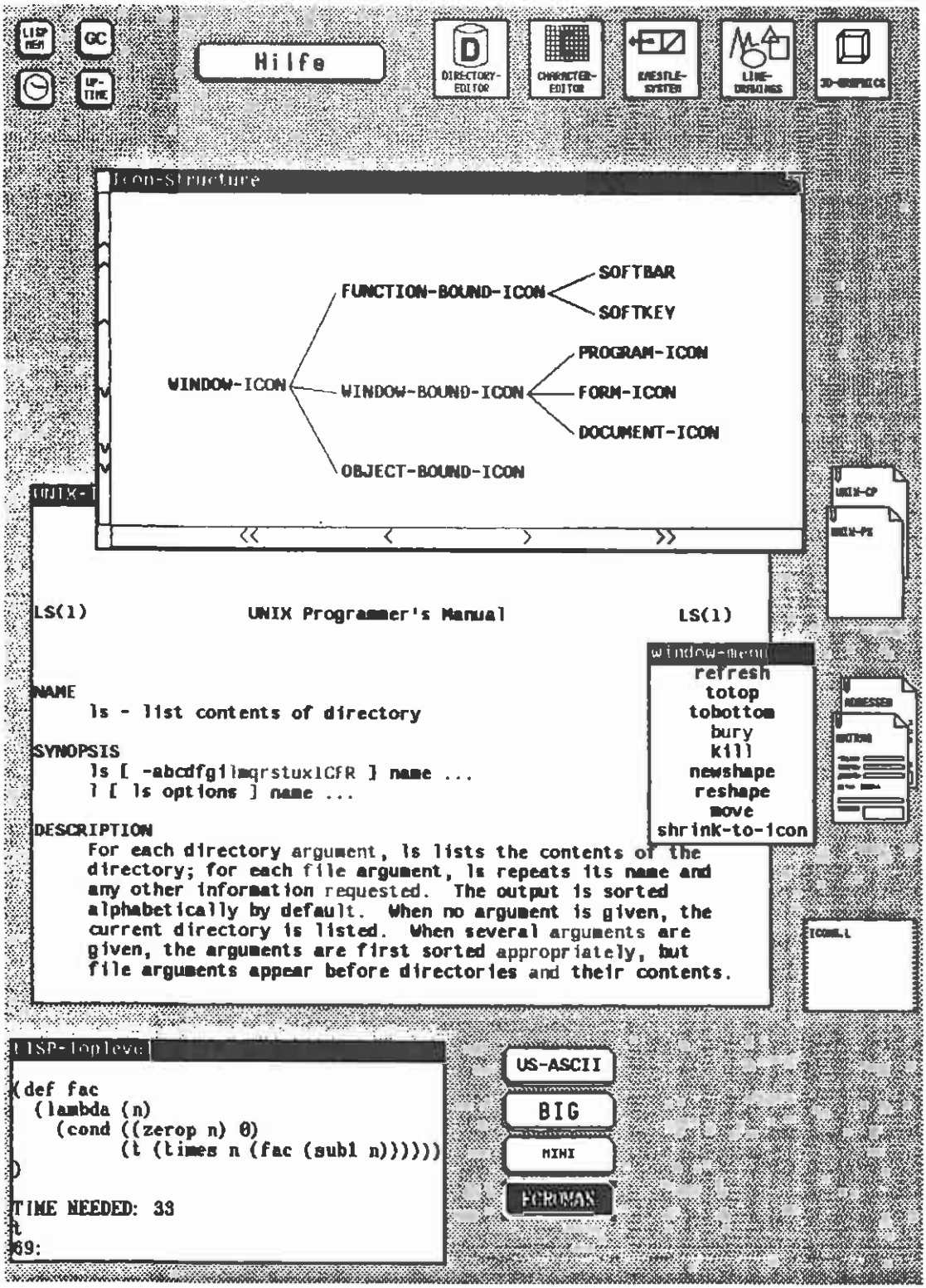

Abbildung 1: Verschiedene Fenster auf dem Bildschirm

oder einem Icon die ihm zugeordnete Aktion oft nur erahnen kann und sich nicht alle Details einer Aktion in ihrer Bezeichnung widerspiegeln können, sind weitere **Hilfeleistungen durch das System** unumgänglich.

Zwei Klassifizierungsmerkmale für Hilfesysteme sind die Unterscheidung zwischen aktiver und passiver Hilfe und die zwischen statischer und dynamischer Hilfe.

Bei **passiven Hilfekomponenten** muß der Benutzer die Hilfe explizit anfordern. Sie müssen dem Benutzer die Formulierung von Anfragen erlauben, mit denen er schnell zu gezielter Information kommt. Gute Darbietung dieser Information ist von großer Wichtigkeit. Auch Fragen, die nicht auf ein bestimmtes Kommando zurückgeführt werden können, sollte das Hilfesystem beantworten können, indem es die notwendige Kommandofolge herausfindet und vorschlägt. Ein Beispiel für ein passives Hilfesystem, das Anfragen in Teilprobleme zerlegt und dem Benutzer schrittweise vorführt, ist das Hilfesystem PASSIVIST [Lemke 84] (siehe Abb. 2).

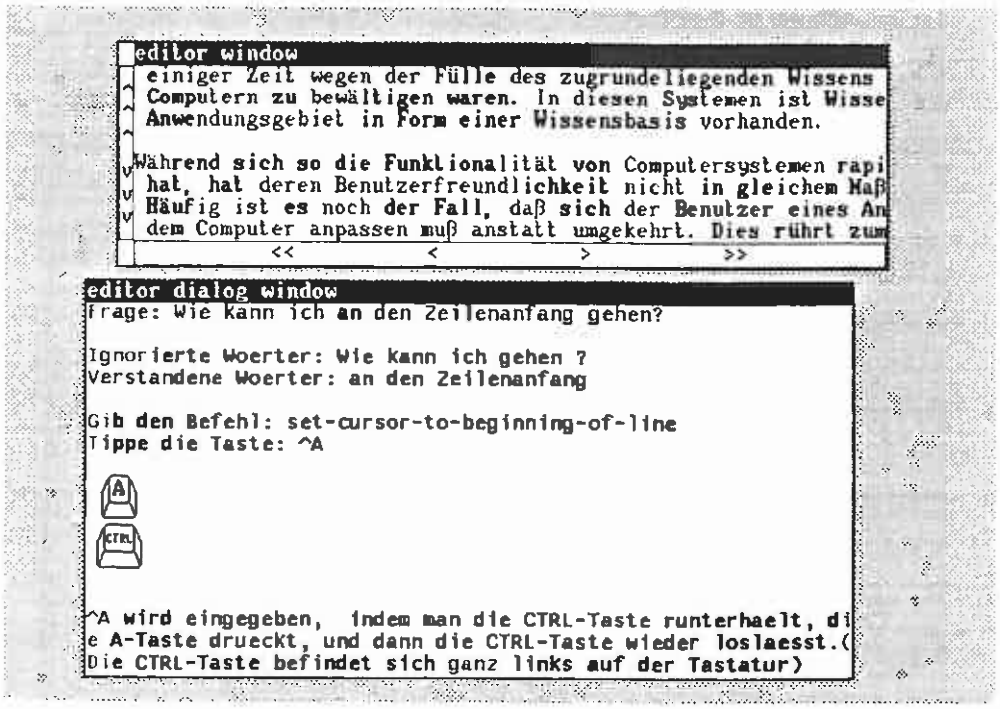

Abbildung 2: Das Hilfesystem PASSIVIST

Aktive Hilfekomponenten helfen, wenn der Benutzer einen Fehler macht, bzw. wenn das System "bemerkt", daß der Benutzer Hilfe braucht. Wenn ein Benutzer wissen will, wie man eine bestimmte Aktion ausführt, kann er ein passives Hilfesystem fragen. Aber in sehr vielen Fällen weiß er nicht einmal etwas über die Existenz bestimmter Kommandos. Ein aktives Hilfesystem kann

durch Protokollierung und Untersuchung der Benutzeraktionen auf die einem Benutzer bekannten Aktionen schließen und Hilfe anbieten, wenn dieser bestimmte Kommandos nicht benutzt, obwohl dies sinnvoll wäre. Das Hilfesystem AKTIVIST [Schwab 84] (siehe Abb. 3) ist ein Prototyp eines aktiven Hilfesystems, das dem Benutzer eines Texteditors bestimmte vereinfachende Operationen erklärt, wenn sie von ihm nicht benutzt werden. Wenn dieser z.B. wiederholt ein Wort Zeichen für Zeichen löscht, wird er darauf hingewiesen, daß es auch eine Taste gibt, die bewirkt, daß ein ganzes Wort auf einmal gelöscht wird.

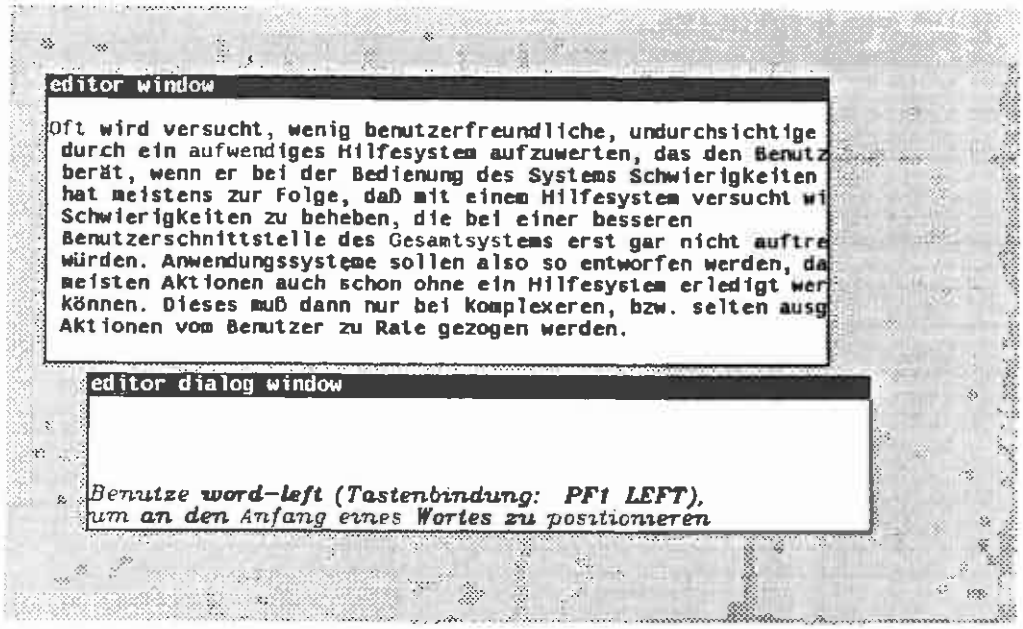

Abbildung 3: Das Hilfesystem AKTIVIST

Statische Hilfe gibt Auskunft über unveränderliche Systemeigenschaften. Sie beinhaltet in etwa die Information, die auch in einem Manual vorliegt. Statische Hilfesysteme geben auf dieselbe Frage, egal zu welchem Zeitpunkt des Dialogs sie gestellt wird, immer dieselbe Antwort.

Dynamische Hilfe berücksichtigt die spezielle Umgebung zum Zeitpunkt der Hilfeanforderung, den Zustand der Interaktion [Hayes 82]. Ein einfaches Beispiel für dynamische Hilfe sind Fehlermeldungen. Sie berücksichtigen oft Benutzereingaben, indem sie nicht einfach umgebungsunabhängig sagen, daß ein Fehler auftrat (z.B. "falsche Eingabe!"), sondern auch, was falsch war (z.B. "die Datei 'verwaltung' existiert nicht"). Leider sind Fehlermeldungen meist sehr kurz und setzen beim Benutzer ein bestimmtes Grundverständnis voraus. Um bei unverständlichen Meldungen weiterzukommen, sollte der Benutzer weitere Fragen über den Fehler stellen können. Eine dynamische Hilfekomponente sollte dabei nicht von der statischen Hilfekomponente abgekoppelt sein, son-

dern bei Nachfragen des Benutzers auf die statische Hilfeinformation zugreifen. Neben Hilfe bei Fehlern soll dynamische Hilfe auch zu jedem beliebigen anderen Zeitpunkt gegeben werden, z.B. wenn vom Benutzer eine Eingabe verlangt wird und er nicht weiß, was er eingeben soll oder warum er es eingeben soll.

Hilfeleistungen erfordern vielfältiges Wissen im System, das unterschiedlich genutzt werden kann. Erklärungen für den Benutzer können bei entsprechender Darstellung des Wissens aus der Wissensbasis generiert werden. Das Problem der Inkonsistenz zwischen Programm und Erklärung, das auftritt, wenn Programmänderungen durchgeführt aber nicht dokumentiert werden, entfällt dann. Die Suche nach einer bestimmten Information wird durch konzeptionell gegliedertes Wissen erleichtert. Aus der Dialoggeschichte und dem Systemzustand läßt sich situationsspezifische Hilfe erzeugen. Ist der Ablauf von Aktionen explizit repräsentiert, lassen sich Erklärungen über einzelne Dialogschritte gewinnen.

3. Gestaltbarkeit

Der unterschiedliche Erfahrungstand von Benutzern ist ein Kriterium, das die Gestalt von Benutzerschnittstellen beeinflussen sollte. Es gibt meist nicht den Benutzer eines Systems, sondern jeder Benutzer hat bestimmte Erfahrungen in der Bedienung des Systems sowie spezielles Fachwissen bezüglich des Anwendungsbereichs. Er hat bestimmte Fähigkeiten in der Aufnahme visueller und akustischer Information und der Ausführung motorischer Tätigkeiten, wie der Bedienung einer Tastatur oder eines Zeigeinstruments. Derartige Erfahrungen und Fähigkeiten sind natürlich nicht unveränderlich und können damit auch nicht fest eingeplant werden.

Die wenigsten existierenden Systeme sind auf die Eigenschaften ihrer Benutzer abgestimmt. Die Folge davon ist am Anfang der Benutzung eine mehr oder weniger starke Überforderung, die häufig der Grund für die Ablehnung des Systems ist. Hat der Benutzer jedoch Erfahrung in der Benutzung gesammelt, stößt er oft schon früh an die Grenzen des Systems. Die Arbeit mit dem System wird dann als monoton empfunden, das System selbst als "dumm" und behindernd.

Um diese Mißstände zu mildern oder ganz zu beseitigen benötigen wir Systeme, die sich an die speziellen Eigenschaften und Erfahrungen des jeweiligen Benutzers anpassen lassen. Solche **adaptierbaren Systeme** sollen ihm insbesondere Möglichkeiten zur Gestaltung der Benutzerschnittstelle anbieten [Herczeg 83]. Im folgenden einige solcher Gestaltungsmöglichkeiten:

Interaktionstechnik

Da es häufig keine im Voraus endgültig zu entscheidende Bevorzugung einer bestimmten Interaktionstechnik gibt, sollte man diese Entscheidung dem Benutzer überlassen (z.B. Menüinteraktion vs. Kommandoschnittstelle).

Informationscodierung

Es gibt sehr viele Möglichkeiten Information auf dem Bildschirm darzustellen (z.B. Farbe, Zeichensätze, Größe der Darstellung). Die für einen Benutzer günstigste Art der Informationscodierung ist stark von persönlichen Präferenzen und der visuellen Aufnahmefähigkeit der betreffenden Person abhängig. Eine Einflußnahme auf die Codierungsmethode durch den Benutzer ist daher sinnvoll und wünschenswert.

Namensgebung

Die Vergabe von Bezeichnungen für die Objekte und Aktionen des Systems kann nur sinnvoll von einer mit der Anwendung vertrauten Person geleistet werden. Der jeweilige Benutzer hat oft einen geeigneteren Wortschatz im Rahmen seiner Tätigkeit, als dies der Systemdesigner in seiner relativen Unkenntnis der eigentlichen Anwendung haben kann. Systeme können so realisiert werden, daß diese Bezeichnungen frei wählbar sind. Häufig ist es darüberhinaus zweckmäßig nicht nur eine bestimmte Bezeichnung für ein Objekt oder eine Aktion zuzulassen. Zu diesem Zweck sollten Synonymlisten durch die Benutzer des Systems angelegt werden können.

Defaults

Oft gibt es (z.B. in Formularen) häufig wiederkehrende Eingabetexte, sogenannte Defaults. Diese können sich im Laufe der Arbeit mit dem System ändern, so daß sie nicht unabänderlich vorgesehen werden können. Der Benutzer sollte sie definieren können. Es gibt Fälle, wo eine bestimmte Eintragung nicht gemacht werden kann, weil sie erst noch in Erfahrung gebracht werden muß. Auch in solchen Fällen ist es oft sinnvoll stattdessen einen Default einzutragen. Bei einer Reisebuchung nach Australien ist es zum Beispiel sinnvoll als Transportmittel das Flugzeug anzunehmen, solange nichts anderes bekannt ist.

Definition von Aktionssequenzen

Bei der Ausführung nahezu aller Tätigkeiten sind immer wiederkehrende Abfolgen von Aktionen zu beobachten. Diese Sequenzen sind aufgaben- und benutzerspezifisch. Sie äußern sich bei der Arbeit mit einem Computersystem meist als die ständige Wiederholung von Eingaben. Ermöglicht man dem Benutzer solche Abfolgen zu definieren und zu benennen, so kann ihm sehr viel stupides Eintippen erspart bleiben.

Adaptierbare Systeme vermitteln dem Benutzer ein Gefühl einer **persönlichen Arbeitsumgebung**, die er sich nach seinen Fähigkeiten und Präferenzen angepaßt hat und jederzeit wieder umgestalten kann.

Ein noch über derartige Möglichkeiten hinausgehendes Ziel sind sich an den Benutzer anpassende Systeme. **Adaptive Systeme** kontrollieren die Art und Weise der Interaktion mit einem bestimmten Benutzer, um ihn bei verschiedenen

Problemen besser unterstützen zu können. Einige Beispiele automatischer Anpassung:

automatische Tippfehlerkorrektur
Das System bemerkt, daß sich ein Benutzer ständig in der gleichen Art und Weise vertippt und dies dann korrigiert. In Zukunft korrigiert das System diesen Tippfehler automatisch, sofern es eindeutig möglich ist.

Erhöhung der Ausführlichkeit von Hilfeinformation
Es kommt sehr oft vor, daß ein Benutzer ständig denselben Fehler bei der Verwendung eines Kommandos macht, selbst wenn das System bereits Hilfeinformation ausgegeben hat. Möglicherweise wurde die Hilfeinformation nicht verstanden. Das System informiert ihn beim nächsten Mal ausführlicher über dieses Kommando und gibt ihm Beispiele.

Hinweis auf vorhandene Funktionalität
Sobald der Benutzer mit den für Anfänger angebotenen Funktionen sicher umzugehen gelernt hat, bietet ihm das System komplexere Funktionen an, die zwar schwieriger zu bedienen sind, aber schneller zum Ziel führen [Bauer, Schwab 85].

automatische Defaults
In einem Formularfeld verwendet der Benutzer ständig denselben Eingabetext. Das System bietet daraufhin diesen Text in dem Feld zur Bestätigung an und erspart dem Benutzer das wiederholte Eintippen.

Das Anbieten von Gestaltungsfunktionen, die automatische Adaption sowie die Verwaltung benutzerabhängiger Unterschiede erfordern beim System Wissen über sich selbst (z.B. Systemzustand, Systemfunktionalität) und den Benutzer. Das Wissen über den Benutzer muß darüberhinaus ständig korrigiert und erweitert werden. Dieses vom System verwaltete Wissen sollte vom Benutzer einsehbar und wenn nötig korrigierbar sein.

4. Fehlerbehandlung

Einen großen Teil seiner Zeit am Rechner verbringt der Benutzer damit, Fehler zu verbessern. Diese reichen von einfachen Tippfehlern bis zu Fehlern, die durch mangelnde Kenntnis des Anwendungsgebiets entstehen. Man kann unter anderem folgende Arten von Fehlern unterscheiden:

Fehler auf syntaktischer Ebene
Dies sind Fehler, die typischerweise bei Benutzereingaben auftreten. Dazu gehören beispielsweise Tippfehler, falsche Benennung von Schlüsselwörtern und vertauschte Parameterreihenfolge.

Fehler durch ein falsches Modell vom System
Diese können auftreten, wenn einem Benutzer bestimmte Konzepte des Systems nicht bzw. nicht richtig bekannt sind. Ein Beispiel dafür ist die Wirkung einer Wortlöschung: Wie sind dabei die Wortgrenzen definiert?

Fehler durch mangelnde Kenntnisse über die Anwendung
Während die ersten beiden Fehlerarten eine Unkenntnis von Systemeigenschaften widerspiegeln, beziehen sich diese auf den Anwendungsbereich eines Systems. Ein Beispiel dafür ist, bei einem Textformatierer einen Zeilenabstand von weniger als 1 einzustellen, obwohl weniger als einzeilig zu schreiben sinnlos ist.

Viele Fehler können durch einen klaren Aufbau des Systems und konsequente Verwendung derselben Begriffe für dieselben Konzepte verringert werden. Wenn es trotzdem zu Fehlern kommt, muß der Benutzer durchschauen können, warum diese auftraten. Das System soll die Ursache von Fehlern erklären können, es soll **fehlertransparent** sein.

Viele Fehler treten nur deswegen auf, weil das System starr ist und nur eine genau festgelegte Eingabeform akzeptiert [Norman 81]. Einige dieser Fehler kann auch das System selbst korrigieren. Es soll **fehlertolerant** sein. Wenn nicht automatisch korrigiert werden kann, bringt eine flexible Fehlerbehandlung den Vorteil, daß der Benutzer gezielt nur seinen Fehler verbessern und nicht die ganze Eingabefolge wiederholen muß. Dieses gezielte Korrekturangebot macht den Fehler außerdem transparenter.

Oftmals ist "Fehler" nicht die richtige Bezeichnung für eine Aktion, die zu einem unerwarteten oder unerwünschten Systemzustand führt. Es gibt Situationen, in denen ein Benutzer erkunden will, was eine bestimmte Funktion in einem bestimmten Kontext bewirkt. Der Benutzer hat bei derartigen Erkundungen kein bestimmtes Arbeitsziel im Sinn. Das Ergebnis der Aktion ist letztendlich das Verstehen der Wirkung der ausgelösten Aktion.

Eine andere Situation, bei der durch die Ausführung von Systemfunktionen Resultate (Systemzustände) entstehen, die nicht erwünscht sind, ist die Ausübung von Planungs- und Designprozessen. Hier ist es nur normal, daß immer wieder Resultate herbeigeführt werden, die nicht der Intention des Benutzers entsprechen. Durch das Verwerfen solcher erreichter nicht erwünschter Zustände wird der Planungs- oder Designprozeß geradezu angetrieben. Auch hier wäre es unangebracht von Fehlern zu sprechen.

Gerade die beiden letztgenannten Fälle implizieren den Wunsch des Benutzers, die Wirkung einer getätigten Aktion wieder beseitigen zu können. Wir sprechen in diesem Zusammenhang von **UNDO-Mechanismen**. Es gibt eine ganze Reihe von Realisierungs- und Interaktionsmodellen für UNDO, die höchst anwendungsspezifisch sind. Was höherentwickelte UNDO-Techniken jedoch gemeinsam haben, ist die Notwendigkeit, das System mit Wissen über den Dialogablauf zu versorgen, den Dialog also in irgendeiner Form zu protokollieren, sowie das System mit Methoden zur Neutralisierung von Funktionsausführungen zu versorgen. Darüberhinaus ist es angebracht, dieses Wissen dem Benutzer zugänglich

zu machen, damit dieser den gewünschten Zustand spezifizieren kann, zu dem zurückgekehrt werden soll. Existiert ein solcher UNDO-Mechanismus, so lassen sich damit auch typische Bedienungsfehler des Benutzers, wie sie zu Beginn dieses Abschnitts diskutiert werden, unwirksam machen.

Bei wissensbasierten Systemen sind Aktionen nicht unbedingt in starren Schemata programmiert, sie können als Erfordernisse spezifiziert sein, die der Benutzer nach und nach erfüllt. Wenn Voraussetzungen nicht erfüllt sind, führt dies nicht zu einem Fehler, sondern das System erfragt sie. Wissen über die Wirkung und den Ablauf von Aktionen kann dazu verwendet werden, Fehler zu erklären, zu korrigieren oder zu neutralisieren. Ungewünschte Systemzustände können durch UNDO-Mechanismen auf vorhergehende Systemzustände zurückgeführt werden. Wissensbasierte Systeme erleichtern nicht nur die Fehlerbehandlung, sondern verhindern Fehler durch konsistenten Aufbau, Anpassung an den Benutzer und Hilfeleistungen.

5. Zusammenfassung

Aus den bisherigen Ausführungen geht hervor, daß ein System, das benutzergerecht sein soll, vielfältiges Wissen benötigt. Sein Wissen wird sowohl zur Beantwortung von Benutzerfragen, als auch zur Steuerung des Systems verwendet. Im folgenden fassen wir das für die angesprochenen wünschenswerten Systemeigenschaften benötigte Wissen nocheinmal zusammen:

* **Wissen über Interaktionsformen und Interaktionsschritte**
* **Wissen über den Systemzustand und die Dialoggeschichte**
* **Wissen über die Funktionalität**
* **Wissen über den Benutzer**
* **Wissen über den Anwendungsbereich und die Arbeitsumgebung**

Bei der Realisierung von Systemen mit den beschriebenen Eigenschaften stellt es sich heraus, daß herkömmliche Programmiertechniken nicht besonders geeignet sind. Einen gangbaren Weg stellt die Verwendung von sogenannten Wissensrepräsentationssprachen dar. In unserem Forschungsarbeit verfolgen wir unter anderem die objektorientierte Realisierung solcher Systeme. Die dazu entwickelte und in Weiterentwicklung befindliche Sprache ist **ObjTalk** [Rathke 84]. Die in den drei Abbildungen dargestellten Systeme wurden auf diese Weise realisiert.

Literatur

[Bauer, Schwab 85]
J. Bauer, T. Schwab: *"Aktive Hilfesysteme"*. In *Notizen zu Interaktiven Systemen*, pp 5-15. Fachgruppe Interaktive Systeme, Darmstadt, Mai, 1985.

[Dzida 83]
W. Dzida: *"Das IFIP-Modell für Benutzerschnittstellen"*. *Office Management* 31(Sonderheft), pp 6-8, April, 1983.

[Hayes 82]
P.J. Hayes: *"Uniform Help Facilities for a Cooperative User Interface"*. Technical Report, Carnegie-Mellon University, Computer Science Department, Pittsburgh, June, 1982.

[Hayes, Szekely, Lerner 85]
P.J. Hayes, P.A. Szekely, R.A. Lerner: *"Design Alternatives for User Interface Managment Systems Based on Experience with COUSIN"*. In L. Borman, B. Curtis (editor), *Human Factors in Computing Systems*, pp 169-175. CHI-85 Conference Proceedings, New York, April, 1985.

[Herczeg 83]
M. Herczeg: *"DYNAFORM - Ein interaktives Formularsystem zum Aufbau und zur Bearbeitung von Datenbasen"*. In H. Balzert (editor), Software-Ergonomie", pp 135-146. German Chapter of the ACM, Stuttgart, 1983.

[Herczeg et al. 85]
M. Herczeg, D. Maier, C. Rathke, W.-F. Riekert: *"Vom Dialogsystem zur wissensbasierten Mensch-Computer-Kommunikation"*. In *ONLINE '85. 8.Europäische Kongreßmesse für Technische Kommunikation*, pp 2P-1-2P-14. Düsseldorf, Februar, 1985.

[Lemke 84]
A. Lemke: *"PASSIVIST: Ein passives, natürlichsprachliches Hilfesystem für den bildschirmorientierten Editor BISY"*. Diplomarbeit Nr. 293, Institut für Informatik, Universität Stuttgart, 1984.

[Norman 81]
D.A. Norman: *"The Trouble with UNIX"*. *Datamation* , pp 139-150, November, 1981.

[Rathke 84]
C. Rathke: *"ObjTalk Primer"*. Institutsbericht, Institut für Informatik, Universität Stuttgart, 1984.

[Schwab 84]
T. Schwab: *"AKTIVIST: Ein aktives Hilfesystem für den bildschirmorientierten Editor BISY"*. Diplomarbeit Nr. 232, Institut für Informatik, Universität Stuttgart, 1984.

[Shneiderman 83]
B. Shneiderman: *"Direct Manipulation: A Step Beyond Programming Languages"*. *IEEE Computer* 16(8), pp 57-69, August, 1983.

[Smith et al. 82]
D.C. Smith, Ch. Irby, R. Kimball, B. Verplank: *"Designing the Star User Interface"*. *BYTE* , April, 1982.