Komplexe Menüs, Häkchen und Funktionsobjekte

Jürgen Krause
Universität Regensburg

Zusammenfassung

An empirisch beobachteten, exemplarischen Benutzerschwierigkeiten mit dem Subsystem Menü in komplexen Softwaresystemen und an seiner potentiellen Ersetzbarkeit durch Funktionsobjekte soll dreierlei demonstriert werden:

- Wie inkonsistent und softwareergonomisch unbefriedigend kommerzielle grafische Software nach mehr als 10 Jahren Entwicklungszeit ist (bzw. - bedingt durch die anwachsende Komplexität - wieder geworden ist).
- Wie eng Detailfragen mit globalen Designentscheidungen nach der "richtigen" Auslegung der Objektorientierung verbunden sind.
- Wie weit die technologiegetriebene Entwicklung im Bereich der Multimedialität - schon bei der einfachen Kombination der beiden Grundmodi Sprache und Grafik im Menü und beim Häkchenformalismus - von einem konzeptuellen Verständnis entfernt ist.

1 Einleitung

Der Beitrag befaßt sich mit Menüsystemen grafisch direktmanipulativer Software zur Büroautomatisation - wie sie in Word für Windows Version 2 (im Folgenden WINWORD2) oder in der COMFOWARE (SNI [10]) enthalten sind - und ihrem potentiellen Ersatz durch eine verstärkte Objektorientierung.

Die Linguistische Informationswissenschaft der Universität Regensburg (LIR) führt zu diesen Systemen seit 1987 empirische Studien durch. Mittlerweile wurden vor allem auf der Basis der COMFOWARE über 100 Benutzer getestet. Es zeigte sich immer wieder, daß die Menübedienung Schwierigkeiten macht. Auch nach längerer Eingewöhnungszeit gelingt es Benutzern nicht problemlos, Handlungsoptionen im Menü aufzufinden, richtig anzuwenden und die darin enthaltenen Elemente der Zustandsanzeige adäquat zu interpretieren. Dieses Ergebnis überrascht umso mehr, als Menüs zu den am besten erforschten Komponenten grafischer Systeme gehören (cf. Paap [6] als Überblick).

Neben der Bürosoftware unter MS-DOS/WINDOWS 3.1 wurde N/JOY unter OS/2 herangezogen. Die LIR testete 1991/92 N/JOY1.01 in der englischen Version. N/JOY verkörpert einen Lösungsansatz, der unter dem Stichwort einer erweiterten

Objektorientierung (bzw. der einzig *echten*) angeboten wird. Der Ansatz führt - zumindest vom Anspruch her - zu einer Auflösung des Menüsubsystems.

Die angebotenen Handlungsalternativen sind hier ikonisierten Objekten (Funktionsobjekten) indirekt zugeordnet, ein Verfahren, das auch die WINDOWS-Systeme neben den Menüs (z.B. Löschen über den Papierkorb) verwenden.

Gerade weil die empirischen LIR-Tests wiederholt die inhärenten - und teilweise nicht beseitigbaren -Schwächen des Menüsystems aufgezeigt haben, wäre es eine wesentliche softwareergonomische Verbesserung, wenn sich Menüs durch eine Ausweitung des objektorientierten Ansatzes ersetzen ließen.

2 Breite, Tiefe, Gruppenbildung und Bezeichnungsvielfalt

2.1 Komplexität heutiger Bürosoftware

Das Hauptproblem komplexer grafisch-direktmanipulativer Systeme ist die Größe ihres Funktionsumfanges. So muß WINWORD2 insgesamt 119, AMIPRO V.2 112 und COMFOTEX4 124 Handlungsoptionen in der Menüstruktur unterbringen. Diese Zahlen verdeutlichen, daß herkömmliche Menüs bei funktionsreichen Applikationen zwangsläufig die Grenzen softwareergonomisch optimaler Gestaltung überschreiten. Ihre Vorteile (Gruppenbildung durch Hierarchisierung und Vorlage der Handlungsoptionen statt Merkleistung) lassen sich nicht mehr voll ausspielen. Sie werden durch die Suchproblematik und die notwendig werdenden langen Bedienwege ins Gegenteil verkehrt.

So fordert z.B. der SNI Styleguide [9] - im wesentlichen im Einklang mit der softwareergonomischen Literatur - die folgenden Obergrenzen nicht zu überschreiten:

a) Menüleiste (1. Ebene) maximal 9 Einträge,

b) Pulldown-Menü (2. Ebene) max. 12 Elemente und maximal 7 Gruppen.

c) Maximal dreistufig. Da die Dialogboxen in die hierarchische Stufung einzubeziehen sind, wäre auf Kaskadenmenüs als 2. Ebene zu verzichten.

Die softwareergonomischen Regeln wirken derzeit hauptsächlich in Richtung einer Vertiefung des hierarchischen Systems, entweder über die Einführung einer zusätzlichen Menüebene (Kaskaden-Menü) wie bei AMI2.0 (8 Kaskaden-Menüs mit 34 zusätzlichen Elementen) oder versteckter über die Dialogboxen wie bei WINWORD2 (teilweise dreifach gestaffelt). Die empirischen LIR-Tests ergaben jedoch gerade für tiefe Hierarchien die größten Schwierigkeiten für den Benutzer.

2.2 "Natürliche" Benennung und Hierarchisierung

Im Rahmen unserer wissenschaftlichen Begleitforschung zur COMFOWARE wurden verschiedene Versionen von Menüs für die COMFOTEX-Textverarbeitung entwickelt. Die Grundproblematik blieb jedoch bei aller Änderung die gleiche: Benutzer arbeiteten nicht effizient mit den Menüs. Sie verbrachten viel unnütze Zeit mit dem Öffnen der falschen Pulldown Menüs und interpretierten einzelne Funktionen unzutreffend. Teilweise waren die Probleme der mangelhaften softwareergonomischen Durchdringung früheren Versionen zuzuschreiben. Im Kern geht es jedoch um etwas anderes: Sprachliche Begriffe lassen sich in vielen Anwendungsbereichen in hierarchischen Begriffsnetzen anordnen, die bei der Mehrzahl der Benutzer übereinstimmen. Die Kategorisierung nach Ober- und Unterbegriffen ist eine mächtige kognitive Grundtechnik der Informationsverarbeitung. Andererseits gibt es Bereiche und Begriffe, bei denen dies nicht möglich ist (cf. auch Bengler [1]). Die Studien zu COMFOTEX ergaben deutliche Hinweise, daß Teile der Textverarbeitung zu letzterer Gruppe gehören.

Konsequenterweise müßte man somit sehr flache Menühierarchien wählen, wofür jedoch der Platzverbrauch zu groß ist. Deshalb werden die Designer auch weiterhin "künstliche" Terme und Strukturen bilden, die der Benutzer - im besten Fall - im Laufe der Zeit lernt.

2.3 Fazit Komplexität, Benennung und Hierarchisierung

Die komplexen Menüs heutiger Bürosoftware enthalten somit inhärente Schwachstellen, die in Anforderungen des Gegenstandsbereichs und in Regeln der sprachlichen Kategorisierung begründet sind. Ihre Auswirkungen lassen sich durch sorgfältige Gestaltung abmildern, aber nicht beseitigen. Die Menüs sind deshalb im Sinne der Softwareergonomie immer nur suboptimal gestaltbar.

3 Häkchenformalismus und Zustandsanzeige

Die LIR-Tests ergaben wiederholt, daß Benutzer zum Häkchenformalismus komplexer Applikationen und den mit ihm verbundenen Sonderfällen - auch nach mehreren Monaten regelmäßigen Gebrauchs - kein mentales Modell aufbauen.

Der Häkchenformalismus ist ein Beispiel für die zunehmende Differenzierung der Gestaltungsmittel grafischer Benutzeroberflächen, ohne daß die Konsequenzen dieser Syntaxerweiterungen durchdacht wären. Vom plausiblen Einzelfall ausgehend entwickelt sich eine immer inkonsistenter werdende Gesamtstruktur. Die bei den

einfachen Ursprungssystemen noch klare Semantik wird differenziert und in der Bedeutung erweitert, ohne daß dem eine Weiterentwicklung und Differenzierung der Oberflächengestaltung folgt. Interessant an der Häkchenanzeige ist zudem, daß sich hier auf einer unteren Ebene die Modalitätsmischung zwischen grafischer Darstellung und verbalen Funktionsbezeichnungen (cf. Abschnitt 1) fortsetzt:

- Funktionseinträge, die Zustandsbeschreibungen entsprechen (z.B. *kursiv*) bekommen ein Häkchen, wenn sie mit der Maus aktiviert werden. Dies entspricht dem Ankreuzen von Listenelementen in der realen Welt, z.B. bei Einkaufszetteln.
- Funktionen, die in einem aktuellen Systemzustand nicht benutzt werden können, sind grau hinterlegt.

Die Grundproblematik der Häkchendarstellung läßt sich gut an der Analyse eines Beispiels aus dem Microsoft Windows Application Style Guide des Software Development Kits (SDK [8], S. 18) klarmachen. Der einzige Unterschied zum SDK Beispiel ist, daß ich 'Underline' in 'Single Underline' und 'Double Underline' aufgeteilt habe, wie dies in vielen Systemen üblich ist.

Style
Normal
Bold
Italic
Single Underline
Double Underline
Outline
Strikeout
Left Aligned
Centered
Right Aligned

Typ a): Aus der ersten Gruppe 'Bold/Italic/.../Strikeout' können mehrere Optionen gleichzeitig ausgewählt und mit Häkchen versehen werden (Mehrfachauswahl).

Eine früher getroffene Auswahl ist durch Häkchen markiert. Soll die Zuweisung der Option rückgängig gemacht werden, wird der Eintrag mit dem Häkchen erneut angeklickt. Neu gesetzte Häkchen führen nicht dazu, daß andere Häkchen der gleichen Gruppe verschwinden.

Die Ergänzung des Menüs durch Häkchen wäre unproblematisch, wenn sie wirklich nur aktuell mögliche Handlungsalternativen bezeichnen würde. Dies trifft jedoch nicht zu. Das Lösen des Häkchens ist semantisch eine versteckte UNDO-Funktion. Das Anklicken von '<Häkchen>+ 'bold' heißt letztlich 'UNDO bold'. Nur diese Interpretation paßt zur obigen Grundregel.

Typ b): Innerhalb der zweiten Gruppe 'Left aligned/Centered/Right aligned' kann jeweils nur eine Option ausgewählt werden (Einfachauswahl). Der Mechanismus zur Auswahl gewünschter Optionen ist ein völlig anderer als bei Typ a).

- Das Anklicken eines nicht mit Häkchen versehenen Eintrags löscht den früher ausgewählten (altes Häkchen verschwindet), der den aktuellen Objektzustand vor dem Auslösen der Option kennzeichnete.
- Der beschriebene Sachverhalt ließe sich noch als Typ a) ergänzendes, adaptives Systemverhalten interpretieren, wenn nicht gleichzeitig das Anklicken von '<Häkchen> + Option' (! nicht grau) eine andere Semantik wie bei Typ a) hätte: Es geschieht nichts; die alte Auswahl wird (unnötigerweise) bestätigt.

Damit greift Typ b) auf die gleichen formalen Mittel wie a) zu, verbindet sie aber mit anderen Systemreaktionen. Wann welcher Typ vorliegt, ist formal nicht erkennbar, günstigstenfalls aus der Semantik der verbalen Benennungen zu erschließen: Lassen sie sich als Entgegensetzungen interpretieren (eine Schrifthöhe kann nicht gleichzeitig 10 und 16 Punkte groß sein), liegt Typ b) vor.

Typ c): Klickt der Benutzer 'Normal' in der ersten Gruppe von 'Style' an, hat dies wiederum völlig andere Konsequenzen. Alle Häkchen der Gruppe verschwinden. 'Normal' ist eine Metaoption, die auf einen inhärent definierten Standard Bezug nimmt. Formal aktiviert sie eine systemseitig definierte Anzahl von Eigenschaften und löscht gleichzeitig alle hierzu abweichenden. Deshalb kann das Häkchen von 'Normal' auch nicht getilgt werden, wenn es erneut angeklickt wird.

Der Häkchenformalismus, dessen oben diskutierte Grundtypen durch eine Fülle von Sonderfällen überlagert werden, ist eine prinzipielle Schwachstelle komplexer Menüsubsysteme. Besonders deutlich wird dies, wenn einander widersprechende Attribute im gleichen markierten Bereich vorkommen (cf. hierzu Krause [3]). Die realisierten Lösungen sind in sich inkonsistent, wenig transparent und voller Ambiguitäten. Benutzer haben keine Chance, auch nur die Grundregeln der Häkchenverwendung zu erschließen und damit in ihr mentales Modell zu integrieren.

Einerseits liegt dies an der Verletzung altbekannter softwareergonomischer Forderungen wie der nach Konsistenz, zu einem Teil spiegeln die Schwierigkeiten jedoch auch die generellen Probleme einer Mischung sprachlicher und grafischer Elemente wider, deren Konsequenzen heute - an der Schwelle zum breiten Einsatz multimedialer Systeme - viel zu wenig verstanden werden (cf. Krause [4]).

4 Auslegung der Objektorientierung

Menüsysteme komplexer Systeme lassen sich - wie in den Abschnitten 1 und 2 gezeigt - partiell verbessern, die Grundprobleme jedoch nicht beseitigen. Das enorme

Anwachsen der Funktionalität überfordert sie. Die exemplarisch dargestellten Inkonsistenzen der Zustandsanzeige und die immer länger werdenden Bedienwege durch mehrstufige Hierarchisierung sind nur die am deutlichsten sichtbaren Zeichen dieses Dilemmas, dem meines Erachtens nur durch konzeptuelle Veränderungen begegnet werden kann. Versucht man dies, erweist sich die Auslegung des Begriffs der Objektorientierung als zentral.

Grafische Benutzeroberflächen legen die objektorientierte Sichtweise fast zwangsläufig nahe. Reale Objekte werden auf dem Bildschirm visuell dargestellt. Zwar ist es nicht unmöglich, auch Handlungen bildhaft wiederzugeben; die Vorteile der Visualisierung gegenüber der verbalen Ausdrucksweise betreffen jedoch hauptsächlich konkrete Objekte wie Schränke, Ordner, den Papierkorb usw.

Diese Objekte stehen in einer Reihe von Beziehungen, von denen in unserem Kontext die Generalisierung bzw. Spezialisierung und die Klassifikation bzw. Instanziierung die wichtigsten sind (cf. Meyer [5], Sager [7]). Die damit möglichen Hierarchisierungen und Klassenbildungen (mit Merkmalsvererbung) sind ein wesentliches Kennzeichen aller objektorientierter Systeme.

Was ist somit gemeint, wenn Systeme wie N/JOY oder WINTOOLS für sich reklamieren, die alleinig richtige Objektorientierung - im Gegensatz z.B. zu WINWORD2 oder COMFOTEX - zu verkörpern?

Die Frage macht darauf aufmerksam, daß der Begriff der Objektorientierung zwar relativ gut bei der objektorientierten Programmierung definiert ist, die Übertragung auf die Benutzeroberflächen aber nur intuitiv, ohne theoretische Diskussion erfolgte. Generell wurde diesem Problem in Krause [3] nachgegangen. Hier beschränke ich mich darauf, die Auslegung der Objektorientierung in Systemen wie COMFOTEX4 oder WINWORD2 der von N/JOY gegenüberzustellen.

4.1 Objektorientierung als Objekt-Funktions-Modell

Zentraler Bedienmechanismus von Systemen wie COMFOTEX4 oder WINWORD2 ist ein Objekt-Funktions-Schema. Der Benutzer geht von einem Objekt aus und markiert es (Leitfrage: Mit was will ich als nächstes etwas tun?). In einem zweiten Schritt (somit im Unterschied zu kommandoorientierten Dialogsprachen nachgeordnet) sucht er sich die Handlungsoption im Menü (Leitfrage: Was will ich mit dem ausgewählten Objekt tun? - Tastenkombinationen und Toolboxen bleiben hier ausgeklammert).

Im Menü sind die Handlungsoptionen verbal symbolisiert und in einer hierarchischen Struktur mehrstufig geordnet. Das Problem, daß einzelne Handlungen nicht global wirken, sondern nur auf bestimmte Objekte, wird durch Graufärbung der Einträge und durch wechselnde Menüeinträge gelöst.

Parallelisiert man obige Sichtweise mit der differenzierten Definition von Objekt-Orientierung für die Programmierung, die z.B. Sager [7] gibt, fällt - trotz weitgehender Übereinstimmung - ein Punkt auf, der nicht übertragen werden kann:

"Methoden sind der Verarbeitungsteil in den Objekten. Sie bestimmen das Verhalten der Objekte. ... Auch Methoden definiert man in der Klasse. Sie werden durch Nachrichten, die ein Objekt erhält, aktiviert. ... In einem Objekt sind damit alle Daten und die Funktionen, die auf diese Daten wirken, zusammengefaßt (Kapselung). ... Methoden bestehen wie Prozeduren und Funktionen in konventionellen Sprachen aus Anweisungsfolgen." (Sager [7], S. 38)

Konzeptualisiert man Menüs strikt im Sinne der objektorientierten Programmierung sind sie eigenständige Objekte mit der Fähigkeit, Nachrichten (die Funktionseinträge) an andere (markierte) Objekte zu senden. Im Gegensatz zur obigen Objekt-Funktions-Konzeptualisierung wäre das Löschen eines Dokuments vom Benutzer wie folgt zu konzeptualisieren: Markieren wählt eine Datei als Objekt1 aus. Danach wird ein zweites Objekt gesucht, das Objekt1 eine Nachricht schicken kann (das Menü). Der Benutzer klickt die zu übertragende Nachricht an und schickt sie damit an Objekt1. Dort löst die Nachricht eine entsprechende Methode in Objekt1 aus: Objekt1 löscht sich selbst (zum hier ausgeklammerten Verstoß gegen die Kapselung cf. Krause [3]).

Kein Benutzer ohne Programmiererfahrung "denkt Menüs" so; er konzeptualisiert vielmehr nach obigem Objekt-Funktions-Schema, in dem Objekte sich nicht selber löschen, sondern gelöscht werden. Deshalb scheint es auch vernünftig, Menüs in die zugehörigen Fenster zu integrieren und nicht - wie z.B. bei den NEXT-Rechnern - als eigenständige Spezialfenster auszulagern. Letztere Designvariante unterstützt die Sichtweise der objektorientierten Programmierung, die als Oberflächenkonzeptualisierung vom Benutzer jedoch erst eingeübt werden müßte (verlernen der "natürlicheren" Objekt-Funktions-Sichtweise).

4.2 Funktionsobjekte

Analysiert man Produkte wie WINTOOLS oder N/JOY, erweist sich der Gegensatz Funktionsobjekt versus Menüsystem als Kern des Anspruchs, stärker objektorientiert zu sein wie WINWORD2 oder COMFOTEX4.

Läßt man die speziellen Schwierigkeiten und Gestaltungsbrüche weg, die sich aus einer durchgehenden Anwendung dieses Prinzips in der Textverarbeitung ergeben (cf. hierzu Weingärtner [11]), und problematisiert auch nicht die negativen Folgen für das Mausbedienungskonzept, das dann die einfache "drag and drop" - Bedienung ersetzen muß, ergibt sich idealiter ein einfaches Objekt-Objekt-Schema, dessen konzeptuelle Unterschiede zu dem von Abschnitt 3.1 herausgearbeitet werden sollen.

a) Der Benutzer markiert das zu bearbeitende Objekt.

b) Er sucht sich ein zweites Objekt ("aktives Werkzeug"), aktiviert es als Funktionsobjekt und wendet es auf das in Schritt a) markierte Objekt (das "passive") an. Das Standardverfahren für die Zuweisung ist das "drag and drop"-Prinzip.

Wesentlich für die These der globalen Verwendung von Funktionsobjekten ist, daß sie die Menüs ersetzen sollen. Darum muß die Handlung dem Werkzeug eindeutig inhärent sein. Z.B. tut ein Reißwolf nichts anderes als Objekte zu zerstören. Deshalb ist es überflüssig, die Handlung "Löschen" explizit zu benennen.

Die Metapher hinter den Funktionsobjekten sind Handlungen in der Welt, die mit Hilfe von dezidierten Werkzeugen ausgeführt werden.

COMFODESK4, der Desktop der COMFOWARE, enthält z.B. das Löschen mit dem Reißwolf: Der Benutzer entscheidet sich für das zu zerstörende Objekt und markiert es (z.B. ein Textdokument). Er 'ergreift' es mit der Maus, zieht es auf den Reißwolf und läßt es los, wodurch das Dokument zerstört wird.

Weitere Vorgänge, die beim Objekt-Objekt-Schema problemlos zur natürlichen Vorgehensweise in der realen Welt parallelisiert werden können, sind alle Formen des Ablegens eines Objekts in einem anderen. Will der Benutzer einen Brief in einem Ordner ablegen, aktiviert er das Objekt, das er ablegen möchte, ergreift es mit der Maus und läßt es über dem zweiten Objekt, dem Ordner, los, wodurch es in den Ordner eingefügt wird.

Beiden als stimmig empfundenen Beispielen ist gemeinsam, daß die aktiven Objekte (Werkzeuge) eineindeutig mit einer Handlung verbunden sind und ihre Lokalisierungsaktionen mit denen der realen Welt übereinstimmen.

Schon bei der Bleistiftmetapher in N/JOY ist das nicht mehr der Fall. Der Benutzer markiert eine Textstelle, an der er mit einer bestimmten Schrift etwas hinzufügen möchte. Dazu wird das Werkzeug des Stiftes 'Helvetica 12' gewählt. Bleibt man beim "drag and drop"-Prinzip, müßte der Benutzer jetzt das Dokument auf den Bleistift ziehen. In der realen Welt geht das natürlich gerade umgekehrt. Die Richtung

stimmt nicht mehr. Deshalb muß auf das abstraktere Prinzip des Markierens des passiven Objekts mit anschließender Aktivierung des zweiten (aktiven) Objekts (Werkzeugs) übergegangen und hierzu wiederum auf abstrakt bestimmte Maus-Modi (z.B. Differenzierung in rechte und linke Maustaste) zurückgegriffen werden. Nimmt man dann noch die in Weingärtner [11] besprochenen Beispiele hinzu, bei denen es keine Werkzeuganwendung in der realen Welt gibt, die die Textverarbeitungsfunktion auslöst, verliert ein durchgehend angewandtes Konzept der Funktionsobjekte endgültig seine "Unschuld". Es kann nicht mehr intuitiv mit Rekurs auf das Handeln in der realen Welt und die natürliche kognitive Konzeptualisierung durchgeführt werden, sondern wird zu einem zwar klaren und effizienten, aber theoretisch-abstrakten Konzept zur Auslösung von Handlungen.

Das Arbeiten mit Funktionsobjekten läßt sich bei einigen Fällen mit dem Vorgehen in der realen Welt in Einklang bringen, nicht jedoch überall. Dort wo es eine metaphorisch "natürliche" Entsprechung gibt, sollten Funktionsobjekte eingesetzt werden. Geht man über diesen Bereich hinaus, verzichtet man auf die Vorerfahrungen des Benutzers und damit auf die intuitive Erschließbarkeit des Verfahrens.

Deshalb sollten Funktionsobjekte immer dann eingesetzt werden, wenn das Bild des (Ein)legens/Einschiebens des Objekts auf/in das Funktionsobjekt in der realen Welt eine Parallele hat und eine hohe Eindeutigkeit der Funktionszuordnung gegeben ist (in diesem Sinn positiv: Drucken, Dokument in Schrank ablegen, Dokument auf Kopierer ziehen usw). Zudem muß die Richtung der Aktion stimmen: Auch in der realen Welt soll das Objekt auf das Funktionsobjekt zubewegt werden und nicht umgekehrt (z.B. kein Funktionsobjekt Schreibstift). In diesen engen Grenzen sind Funktionsobjekte ein wesentlicher Bestandteil direktmanipulativer Benutzeroberflächen und erhöhen die Selbsterklärungsfähigkeit.

5 Fazit

Funktionsobjekte können Menüs nicht generell ersetzen. Sie sind dort sinnvoll, wo sich das Objekt-Objekt-Schema mit den Vorgängen in der realen Welt parallelisieren läßt. Hier eingesetzt erhöhen sie den direktmanipulativen Charakter des Gesamtsystems.

Generalisiert angewendet sind Funktionsobjekte ein theoretisch-abstraktes Konzept der Datenorganisation, das als spezielle Sichtweise auf die Welt gelernt und eingeübt werden muß, bevor es seine Vorteile entfalten kann. Dies erscheint bei der Programmierung keine ernstliche Hemmschwelle, wohl aber in den hier besprochenen Teilbereichen grafisch-direktmanipulativer Benutzeroberflächen.

Darauf, daß es andere Anwendungen gibt, bei denen sich eine strikte Parallelisierung der Objektorientierung im Sinne der objektorientierten Programmierung durchaus auch für eine softwareergonomisch sinnvolle Gestaltung von Benutzeroberflächen fruchtbar machen läßt, kann hier nicht mehr eingegangen werden. Ein Beispiel aus dem Bereich Werkstoffinformationssysteme enthält Krause [2].

Literatur

[1] Bengler, K.: Experimentelle Untersuchung zur Repräsentation semantischer Strukturen. Diplomarbeit Universität Regensburg (1990)

[2] Krause, J.: WING-M2. Ein objektorientiertes Werkzeugsystem für Werkstoffdatenbanken. WING-IIR Arbeitsbericht. LIR Universität Regensburg (1992a)

[3] Krause, J.: Menüproblematik, Häkchenformalismus und Objektorientierung. COMFOLIR Arbeitsbericht. Universität Regensburg (1992b)

[4] Krause, J.: A mulitilayered empirical approach to multimodality. In: Maybury, M. (ed.): Intelligent multimedia interfaces. (To appear 1993)

[5] Meyer, B.: Objektorientierte Softwareentwicklung. Wien et al (1990)

[6] Paap, K.R.: Design of Menus. In: Helander, M. (ed.). Handbook of Human-Computer Interaction. Amsterdam (1988) 205-235

[7] Sager, W.: Objektorientierte Programmierung. Computer Magazin 7 (1991) 34-40

[8] SDK: Microsoft Windows Application Style Guide of the Software Development Kit. Microsoft Corporation (1991)

[9] Siemens Nixdorf Informationssysteme AG: Styleguide. Richtlinien zur Gestaltung von Benutzeroberflächen. München (1990)

[10] Siemens Nixdorf Informationssysteme AG: COMFOTEX Version 4.0. Referenzhandbuch. München (1991)

[11] Weingärtner, M.: Benutzertests mit N/JOY. Sept 1992. LIR Arbeitsbericht. Regensburg (1992)

Prof. Dr. Jürgen Krause
Universität Regensburg
FG Linguistische Informationswissenschaft
Universitätsstr. 31
8400 Regensburg

email: krause@vax1.rz.uni-regensburg.dbp.de.