Seminar Software–Ergonomie

Thema 4:

Menü, Maske, Fenstertechnik, Direkte Manipulation

von Andreas Ackermann
Erlangen, den 2. Dezember 1997

HS an der FAU Erlangen im WS 1997/98
Ergonomische Gestaltung Computergestützter Arbeit
S. Olschner

Inhaltsverzeichnis

1 Einleitung

Die rasante Entwicklung auf dem Gebiet der Computerhardware ermöglicht heute Benutzerschnittstellen, die noch vor wenigen Jahrzehnten nur mit Supercomputern in Forschungslabors zu realisieren waren. Die heute jedem bekannten und als selbstverständlich angesehenen graphischen Benutzeroberflächen, wie z.B. Windows95 sind eigentlich erst in den letzten acht Jahren für eine breite Anwenderschicht verfügbar geworden.

Allerdings hat die Entwicklung anwenderfreundlicher Software, die diese neuen Möglichkeiten ausnutzt, nicht mit dem technischen Fortschritt mithalten können. Dies mag zum einen daran liegen, daß neue Bedienungskonzepte erst erdacht und erprobt werden müssen, andererseits aber auch daran, daß der durchschnittliche Programmierer zwar durchaus in der Lage ist, strukturell gute Softwarelösungen zu erstellen, aber aufgrund seiner einseitig ingenieurtechnischen Ausbildung gravierende Fehler bei der Gestaltung der Benutzerschnittstelle macht.

Aber es ist genau diese Benutzerschnittstelle, durch die sich eine Software dem Anwender präsentiert und aufgrund derer ein Programm als angenehm und gut bedienbar oder umständlich und schlecht eingestuft wird. Deshalb gilt es in der Zukunft, eine für den Endanwender einfache, transparente und durchdachte Bedienung zu erreichen und dem Punkt Benutzerschnittstelle eine angemessene Bedeutung zukommen zu lassen, anstatt, wie es auch heute noch vielfach Praxis ist, ihn fast gänzlich zu vernachlässigen.

2 Allgemeine Gestaltungsprinzipien

Die Benutzeroberflächen heute üblicher Systeme lassen sich in text- (konsolen-) orientierte und graphische Schnittstellen einteilen. Trotz der grundsätzlich unterschiedlichen Interaktionsmöglichkeiten des Benutzers mit der Software lassen sich die in diesem Kapitel zusammengefaßten Überlegungen auf beide Arten von Schnittstellen anwenden. Auch wenn diese Regeln immer noch sehr allgemeiner Natur sind und viel Gestaltungsfreiraum lassen, so ist dennoch immer darauf zu achten, ein einmal gewähltes Konzept durch die ganze Applikation hindurch aufrechtzuhalten.

2.1 Wahl von Bezeichnern

Bei der Benennung von Objekten, wie Eingabefeldern oder Menüpunkten, sollten möglichst kurze Bezeichnungen gewählt werden. (z.B. `Nachname:` statt `Geben Sie den Nachnamen ein:`). Dies dient nicht nur der Platzersparnis auf dem Bildschirm, sondern auch der schnelleren Aufnahme durch den Benutzer, da sich die Darstellung auf das Wesentliche beschränkt. Da von links nach rechts gelesen wird, sind signifikante Schlüsselwörter möglichst weit links anzuordnen, damit irrelevante Objekte sofort als solche erkannt werden. Innerhalb einer Applikation sind die Bezeichnungen für gleiche Objekte konsistent zu halten. Um viel Information auf den Bildschirm zu bringen, darf abgekürzt werden, wobei allerdings nach Möglichkeit auf bereits bekannte Abkürzungen zurückgegriffen werden sollte [3], S. 47.

Um bei grafischen Benutzeroberflächen (GUIs, Graphical User Interfaces) Objekte auch mit der Tastatur selektieren zu können, hat es sich eingbürgert, Tastaturabkürzungen (Shortcuts) einzuführen. Zusammen mit einer Sondertaste muß eine alphanumerische Taste gedrückt werden, um das Objekt (z.B. ein Eingabefeld) zu aktivieren. Der zu verwendende Buchstabe erscheint beim Objektbezeichner unterstrichen. Dies ermöglicht ein intuitives Arbeiten und läßt dem Benutzer die Wahl, sich für Maus- oder Tastaturbedienung zu entscheiden. Jedoch zeigt sich im Zuge der Sprachanpassung, daß bei der Übersetzung von Objektbezeichnern die im englischen Originial verwendeten Shortcuts oft nicht mehr vorkommen, so daß andere Tastaturkürzel vergeben werden müssen. Das führt dazu, daß ein und dasselbe Programm je nach Spracheinstellung unterschiedlich zu bedienen ist.

Es gilt auch, auf eine eindeutige sprachliche Formulierung zu achten. So läßt z.B. der in Abbildung 1 gezeigte Sicherheitsrequester, der nach Anwählen von `Programmende` erscheint, den Benutzer im Unklaren, welcher Button das Programm nun tatsächlich beendet.

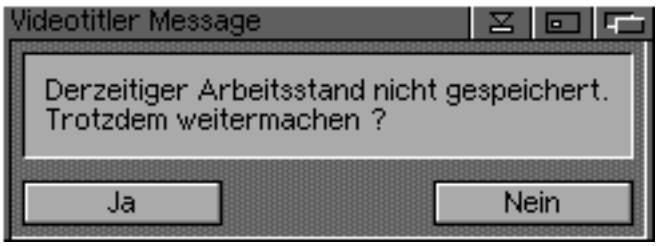

Abbildung 1: *Unklarer Entscheidungsrequester*

2.2 Anordnung von Objekten

Auch eine sinnvolle Aufteilung einer Bildschirmseite trägt stark zu einer effizient nutzbaren Schnittstelle bei. Zusammengehörige Objekte sollten sofort als solche erkennbar sein. Wichtige Informationen sind so zu plazieren, daß sie bevorzugt wahrgenommen werden. Untersuchungen haben ergeben, daß dies zum einen im hellsten Bereich des Bildschirms der Fall ist und zum anderen eine Präferenz für das linke obere Viertel besteht.

Zieht die Aktivierung einer Option das (Un)verfügbar werden weiterer Optionen nach sich, so müssen die betroffenen Schalter unbedingt entsprechend markiert werden. (z.B. `parallel | seriell` ⇒ `Baud: 300 400 1200`). Die Markierung nicht anwählbarer Schalter kann entweder farblich (bunt → blaßgrau) oder durch Rasterung erfolgen.

Objekte werden am besten anhand eines (gedachten) Gitters, das über das Dialogfenster gelegt wird, ausgerichtet. Bei der tabellarischen Anordnung von Optionen ist ferner zu beachten, daß wichtige Einstellungen zuoberst und untergeordnete entsprechend weiter unten erscheinen. So macht es z.B. wenig Sinn, den Benutzer zuerst ein Füllmuster für den Hintergrund auswählen zu lassen und in der nächsten Zeile erst die Auswahl zwischen Muster und leerem Hintergrund anzubieten.

Eine Belegung mit sinnvollen Vorgaben (Default-Werten) bei Eingabe- und Optionsfel-

dern beschleunigt die Arbeit und gibt vor allem dem Anfänger die Möglichkeit, ohne Vorkenntnisse bereits erste Ergebnisse zu erzielen. Allerdings darf dies nicht dazu führen, daß routinemäßig Default-Werte verändert werden müssen (z.B. das Ersetzen von `neue Schublade` beim Einrichten einer neuen Schublade).

Aktionsauslösende Knöpfe (Buttons), wie `Ok`, `Cancel`, `Load` etc. bzw. Statuszeilen bei Terminalanwendungen sind immer am unteren Bildsschirmrand anzuordnen. Damit ergibt sich bereits eine Grobgliederung, die der Benutzer in nahezu allen Eingabefenstern wiederfinden kann und die ihm damit Gelegenheit gibt, die Aufmerksamkeit wichtigeren Dingen zuzuwenden. Eine ansprechende Formatierung und überlegter Einsatz von Trennlinien runden eine gut gestaltete Dialogseite ab.

Abbildung 2: *Ansprechend gestaltetes Optionsmenü [5], S. 126*

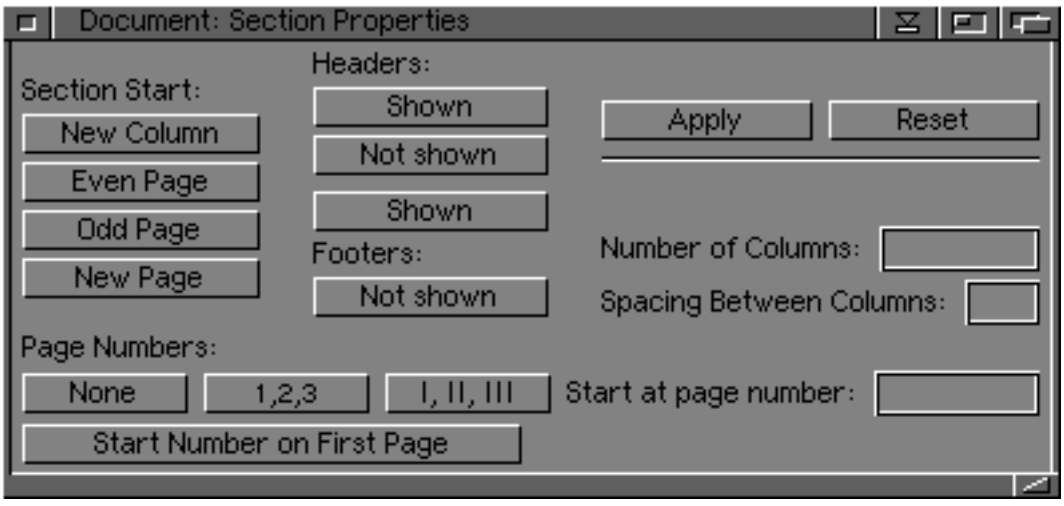

Abbildung 3: *Ohne Berücksichtigung von Guidelines erstelltes Menü [5], S. 127*

2.3 Gestaltung von Textmenüs

Ein wichtiger Punkt beim Erstellen von Menüs ist die Strukturierung. Menüs können breit sein, d.h. viele Informationen auf einer Ebene enthalten, oder tief sein (wenige Punkte

auf einer Ebene, dafür aber eine Vielzahl von Ebenen). Breite Menüs sind wenig übersichtlich, da sie zuviel Information auf einmal präsentieren, andererseits erfordern tiefe Menüs eine Vielzahl von Auswahlen, bis man sich zum gewünschten Punkt durchgearbeitet hat. Oftmals bietet es sich innerhalb der Menüstruktur an, von einer Unterebene aus eine weitere Unterebene direkt zugänglich zu machen, ohne daß der Anwender über die Hauptseite zurückgehen muß. Dies ermöglicht zwar ein schnelles Arbeiten, führt aber zur Verwirrung, da man sich durch Anwahl eines Unterpunktes plötzlich in einem anderen Zweig der Hierarchie wiederfindet.

Als Anordnungsschema für die Menüpunkte einer Seite hat sich eine logische Gruppierung (Zusammenfassung von Objekten mit ähnlichen Eigenschaften) als am zweckmäßigsten erwiesen. Testreihen haben ergeben, daß Probanden bei dieser, im Gegensatz zu alphabetischer oder zufälliger Anordnung, am effizientesten mit dem angebotenen Menüsystem arbeiten konnten.

2.4 Antwortzeiten bei interaktiven Systemen

Die gestiegenen Datenübertragungsraten und Verarbeitungsgeschwindigkeiten von Computersystemen werden immer wieder durch höhere Ansprüche an das System und komplexere Aufgaben relativiert. Deshalb ist es auch heute und in Zukunft immer noch wichtig, sich über das Antwortverhalten Gedanken zu machen. Direkte Manipulation erfordert sofortige Reaktion des Rechners auf Benutzereingaben. Während ungeübte Benutzer in der Anfangsphase auch mit trägen Systemen zufrieden sind, ist die Arbeit für den Experten die Arbeit mit solchen Systemen inakzeptabel. In diesen Fällen ist es sinnvoll, unter Umständen auf Kosten einer einfachen Bedienung, die Antwortzeiten zu reduzieren. Aktuell ist dies am WWW zu beobachten. Die größere Übertragungskapizität wird durch die zunehmende Verwendung von Grafiken auf Webseiten praktisch kompensiert.

2.5 Online-Hilfe

Sowohl für ungeübte Benutzer als auch für Experten kann sich eine Online-Hilfe als sehr nützlich erweisen. Vor allem bei Verwendung als Kurzreferenz beschleunigt sie das Arbeiten erheblich. Sind jedoch größere Passagen durchzulesen, ist einem gedruckten Manual der Vorzug zu geben, denn die Online-Hilfe unterbricht die Arbeit, verdeckt relevante Informationen und überfordert somit das Kurzzeitgedächtnis. Ein guter Ansatz in dieser Richtung sind die sogenannten 'Bubble-Help'-Systeme, die nach einer kurzen Verweilzeit über einem Bedienungselement dessen Funktion in Form einer Sprechblase kurz erläutern.

2.6 Einfache Bedienbarkeit

Eine gute Benutzeroberfläche zeichnet sich dadurch aus, daß alle wichtigen Funktionen leicht und ohne Umwege erreichbar sind. In einer Textverarbeitung ist es sicher kein guter Ansatz, die Druckfunktion erst nach drei weiteren Dialogfenstern zugänglich zu machen. Gute Software sollte dem Anwender immer die Aktionen direkt anbieten, die

er am wahrscheinlichsten als nächstes ausführen möchte. Tritt z.B. beim Einlesen einer CD ein Fehler auf, so ist es sinnvoll, dies nicht nur mitzuteilen, sondern auch gleich die Möglichkeit anzubieten, das Optionsfenster zu öffnen, in dem man die Lesegeschwindigkeit reduzieren kann.

3 Direkte Manipulation

Nach Shneiderman [2] wird direkte Manipulation als Sammelbegriff für Benutzerschnittstellen mit den folgenden Eigenschaften definiert:

- Permanente Sichtbarkeit aller Objekte
- Ersetzung komplexer Kommandos durch physische Aktionen, wie Mausbewegungen, Selektionsaktionen und Funktionstastenbetätigung
- Schnelle, durch `UnDo` umkehrbare, einstufige Benutzeraktionen mit unmittelbarer Rückmeldung

3.1 Grenzen der direkten Manipulation

Die Umsetzung dieses Ansatzes ist heute durch Windows95 hinlänglich bekannt. Drag&Drop ermöglicht das intuitive Verschieben von Dateien zwischen Verzeichnissen oder das Laden von Projektdateien durch simples Loslassen ihrer Piktogramme (Icons) über der gewünschten Applikation.

Dabei handelt es sich aber um einfache Vorgänge, von denen sich der Benutzer leicht ein mentales Modell erarbeiten kann. Durch Dateioperationen sind die Möglichkeiten von Drag&Drop sicherlich noch nicht ausgereizt – allerdings bleibt die Frage, ob sich *beliebig* komplexe Sachverhalte auf diese Weise ausdrücken lassen.

3.2 Direkte Manipulation – Textuelle Steuerung

Was bei einem neuen Benutzer noch den Spieltrieb weckt, kann für den geübten jedoch schnell lästig werden. Durch zeitaufwendiges Herumschieben von Symbolen auf dem Bildschirm fühlt sich der Experte bald unterfordert. Deshalb ist es dringend notwendig, alternativ zur Mausbedienung zumindest Tastaturabkürzungen anzubieten oder noch zusätzlich eine terminalorientierte Schnittstelle zu implementieren. Dieser zweigeteilte Ansatz bringt den Vorteil mit sich, daß der Anfänger spielend den Umgang mit der Software erlernen kann, da ihm bei einer graphischen Benutzeroberfläche immer alle möglichen Aktionen angezeigt werden. Wächst sein Verständnis, so kann er sich nach und nach die Tastaturkürzel aneignen und damit effizient arbeiten.

4 Graphische Benutzeroberflächen aus Programmierersicht

Des Benutzers Freud ist des Programmierers Leid. Es ist nicht zu leugnen, daß eine graphische Benutzeroberfläche einen erheblich höheren Programmieraufwand erfordert, als eine textorientierte Schnittstelle.

Es gibt zwar inzwischen ausgereifte Interfacebuilder, die dem Programmierer vorgefertigte Elemente wie Radiobuttons, Listen, Schalter, Texteingabefelder, etc. zur Verfügung stellen, zum Teil auch eine Konstruktion der Benutzeroberfläche durch Drag&Drop erlauben und danach den entsprechenden Quellcode erzeugen. Ferner existieren Objektbibliotheken, die ebenfalls eine relativ einfache Einbindung ins eigene Programm erlauben. Trotzdem muß der Umgang mit diesen sehr komplexen Werkzeugen erst erlernt werden und es ist nicht zu vergessen, daß diese Werkzeuge zwar das Interface auf den Bildschirm bringen, aber für die dahintersteckende Funktionalität (z.B. (In-)aktivierung von Optionen in Abhängigkeit von anderen Einstellungen) der Programmierer selbst zu sorgen hat.

Gute Interfacebuilder nehmen ihm aber dennoch einen großen Teil der Arbeit ab. So sorgen sie z.B. dafür, daß sich das Layout an der gewählten Schriftgröße orientiert, bieten die Möglichkeit, Rahmen, Hintergründe und Farben benutzerdefiniert zu gestalten, vereinfachen die Erstellung multilingualer Oberflächen und enthalten bereits eine Vielzahl vorgefertigter Bedienelemente, wie Dateiauswahl, Schriftauswahl, Farbauswahl, Einstellung der Bildschirmauflösung oder einfache `Ja/Nein` -Requester.

Noch einen Schritt weiter gehen die automatischen Layout-Systeme, die nicht mit festen Koordinatenangaben für die einzelnen Bedienelemente arbeiten, sondern lediglich Angaben für die Gruppierung und Lage der Objekte untereinander als Eingabe verwenden. Auch die erforderliche Mindestgröße für ein Fenster wird nicht vom Programmierer vorgegeben, sondern vom Layoutsystem errechnet. Das Ergebnis sind in jedem Fall ansprechende, gut gestaltete Interaktionsfenster. Ein weiterer Vorteil besteht darin, daß bei Änderung einer Objektbezeichnung oder einer Sprachanpassung das Layout nur neu berechnet werden muß, anstatt alle Buttons von Hand neu zu positionieren, falls der ursprünglich vorgesehene Platz nicht mehr ausreicht.

Um im Textmodus einen Dateinamen vom Benutzer zu erfragen, wurde oft einfach so vorgegangen:

```
Please type a Filename:
```

Der Code für diese Abfrage sieht so aus:

```
#include <stdio.h>

char *GetFileName( void ) {

    static char buffer[MAXPATHLEN];

    printf( "Please type a filename:\n");
    scanf( "%s", buffer );
    return buffer;
}
```

Abbildung 4: *Objektauswahl des MUI-Builders [6]*

Ein Interfacebuilder hingegen generiert für prinzipiell die gleiche Aufgabe folgenden Code (dies ist nur die Programmierschnittstelle; der eigentliche Requester, um den sich der Anwendungsprogrammierer allerdings nicht zu kümmern braucht, umfaßt mehrere tausend Codezeilen):

Abbildung 5: *Vom MUI-Builder erzeugtes Dialogfenster*

```
#include "test1.h"

struct ObjWI_label_2 * CreateWI_label_2(void)
{
        struct ObjWI_label_2 * Object;

        APTR    GROUP_ROOT_2;

        if (!(Object = AllocVec(sizeof(struct ObjWI_label_2),
                MEMF_PUBLIC|MEMF_CLEAR)))
                return(NULL);

        Object->STR_PA_label_0 = String("", 80);

        Object->PA_label_0 = PopButton(MUII_PopFile );

        Object->PA_label_0 = PopaslObject,
                MUIA_HelpNode, "PA_label_0",
                MUIA_Popasl_Type, 0,
                MUIA_Popstring_String, Object->STR_PA_label_0,
                MUIA_Popstring_Button, Object->PA_label_0,
        End;

        GROUP_ROOT_2 = GroupObject,
                Child, Object->PA_label_0,
        End;

        Object->WI_label_2 = WindowObject,
                MUIA_Window_Title, "File to Load:",
                MUIA_Window_ID, MAKE_ID('2', 'W', 'I', 'N'),
                WindowContents, GROUP_ROOT_2,
        End;


        if (!Object->WI_label_2)
        {
```

```
                FreeVec(Object);
                return(NULL);
        }

        DoMethod(Object->WI_label_2,
                MUIM_Window_SetCycleChain, Object->PA_label_0, 0 );
        return(Object);
}

void DisposeWI_label_2(struct ObjWI_label_2 * Object)
{
        MUI_DisposeObject(Object->WI_label_2);
        FreeVec(Object);
}
```

5 Ein Beispiel: Der Reqtools Dateirequester

Als Beispiel für ein gelungenes Eingabefenster soll hier der Reqtools Dateirequester vorgestellt werden. Als fertiges Modul implementiert, kann er mit ca. 20 Zeilen Quellcode in eigene Programme eingebunden werden und bietet dem Benutzer ein hohes Maß an Flexibilität. Der Requester speichert jeweils Position und Größe des Fensters des vorherigen Aufrufs. Je nach Platzangebot kann man so die angezeigte Anzahl von Dateien selbst bestimmen. Die Auswahl erfolgt entweder über die Cursortasten und `Return` oder über die Maus. Für das Öffnen eines Unterverzeichnisses genügt bereits ein einfacher Mausklick, da diese Aktion durch Anklicken von `Mutterverzeichnis` leicht wieder rückgängig gemacht werden kann. Ein einfacher Klick auf eine Datei befördert den Dateinamen in das zugehörige Texteingabefeld, ein Doppelklick lädt sie sofort. Als hilfreiche Besonderheit ist auch noch die Möglichkeit vorgesehen, neue Unterverzeichnisse zu erzeugen.

Bei der Verwendung dieser Art von Requestern ist programmiererseitig darauf zu achten, daß kontextabhängig mit unterschiedlichen Dateinamenspuffern gearbeitet wird. So ist es durchaus legitim, innerhalb eines Projektmenüs für `Laden` und `Speichern` jeweils den Dateinamen und das Verzeichnis des letzten Aufrufs einer der beiden Funktionen anzugeben, jedoch sollte es vermieden werden, beim Aufruf von `Optionen speichern` eine Projektdatei als Vorgabe im Requester anzuzeigen. Vielmehr ist es hier sinnvoll, einen getrennten Puffer zu verwenden, der per Default z.B. `application/presets` enthält.

Durch die systemweite Verwendung solcher Requester ist bereits ein großer Schritt in Richtung einheitlicher Bedienung unternommen, denn in jedem Programm, das in irgendeiner Art und Weise Dateien verwendet, erfolgt damit die Auswahl einer Datei auf demselben Weg.

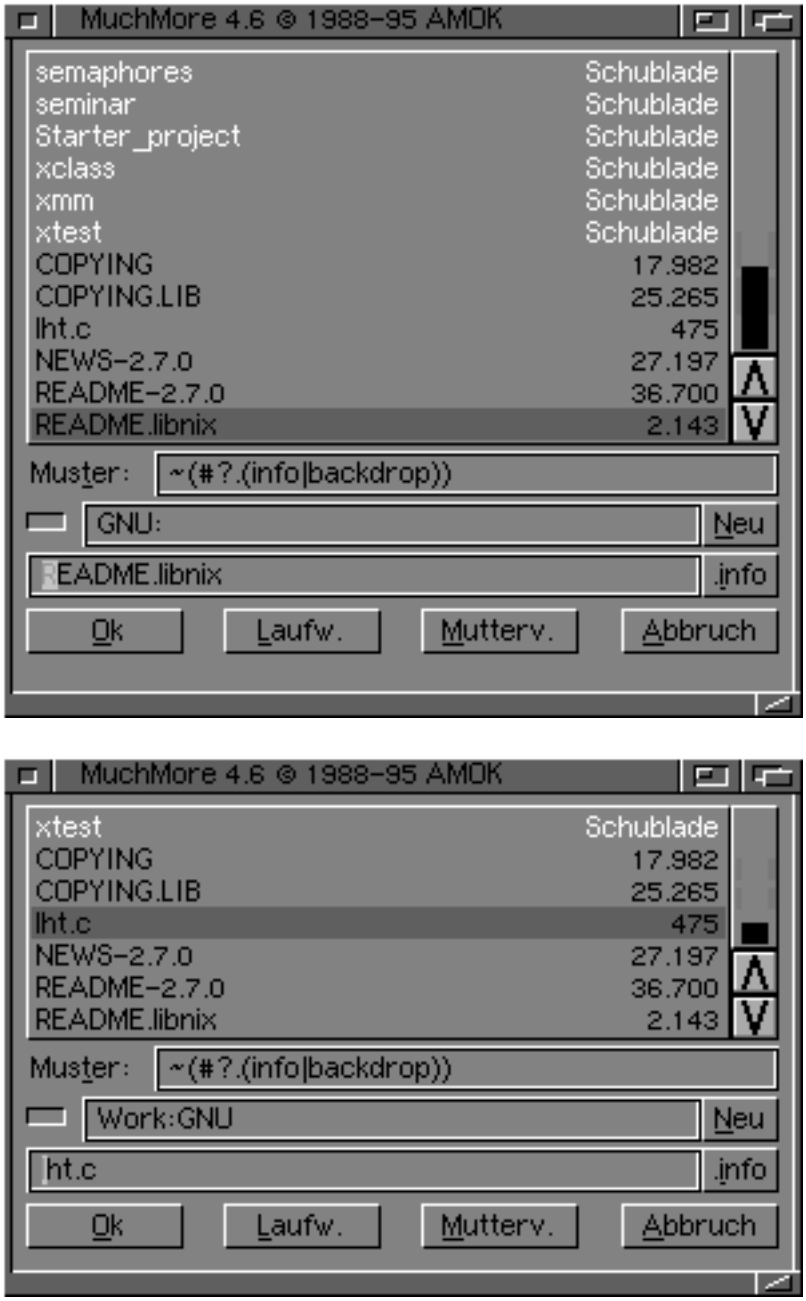

Abbildung 6: *Der Reqtools Dateirequester [7]*

6 Beispiel-Dialogfenster

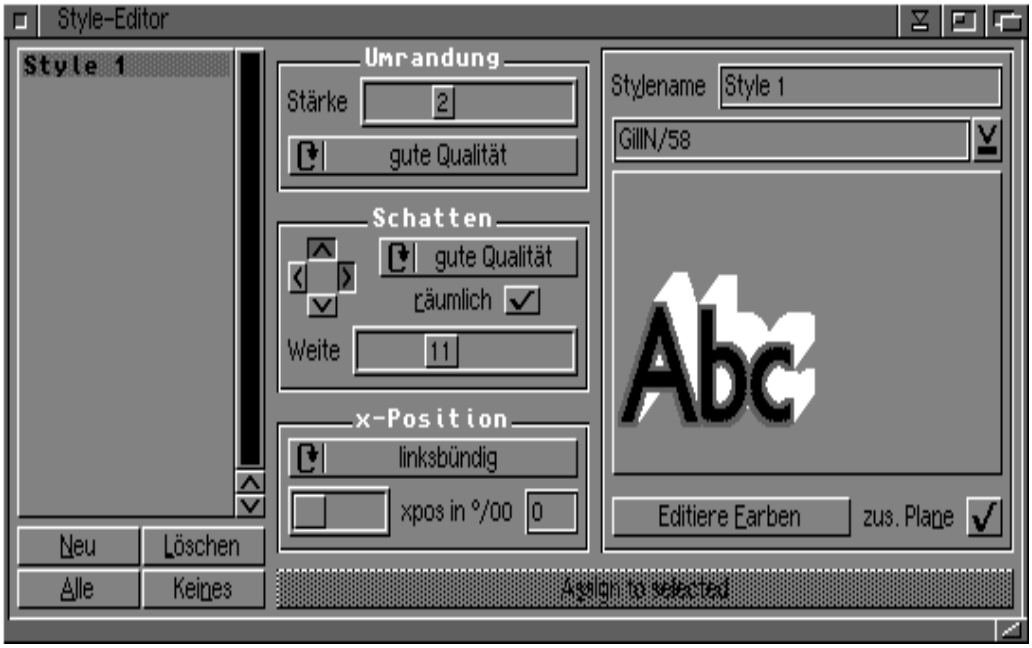

Abbildung 7: *Einstellungsfenster für Schriftstil*

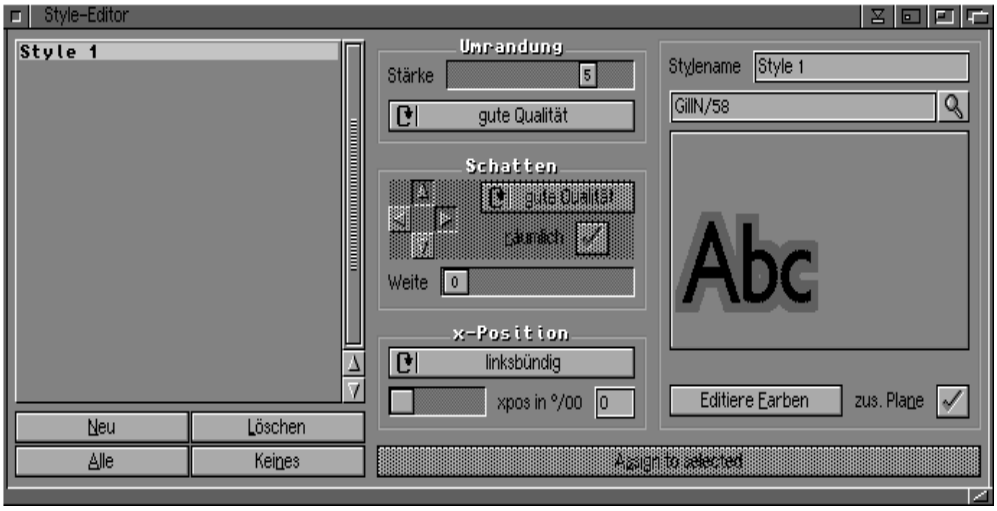

Abbildung 8: *Fenster aus Abbildung 7 mit benutzerdefinierten Gestaltungselementen*

7 Zusammenfassung

Diese Ausführungen zeigen deutlich, daß es nicht einfach genügt, einige Bedienungselemente wahllos über den Bildschirm zu verteilen, das Ganze mit einer netten Grafik zu versehen und dann zu hoffen, eine akzeptable Benutzeroberfläche zu erhalten. Vielmehr ist der Entwurf einer guten Schnittstelle ein kreativer Prozeß, der sich unter Umständen

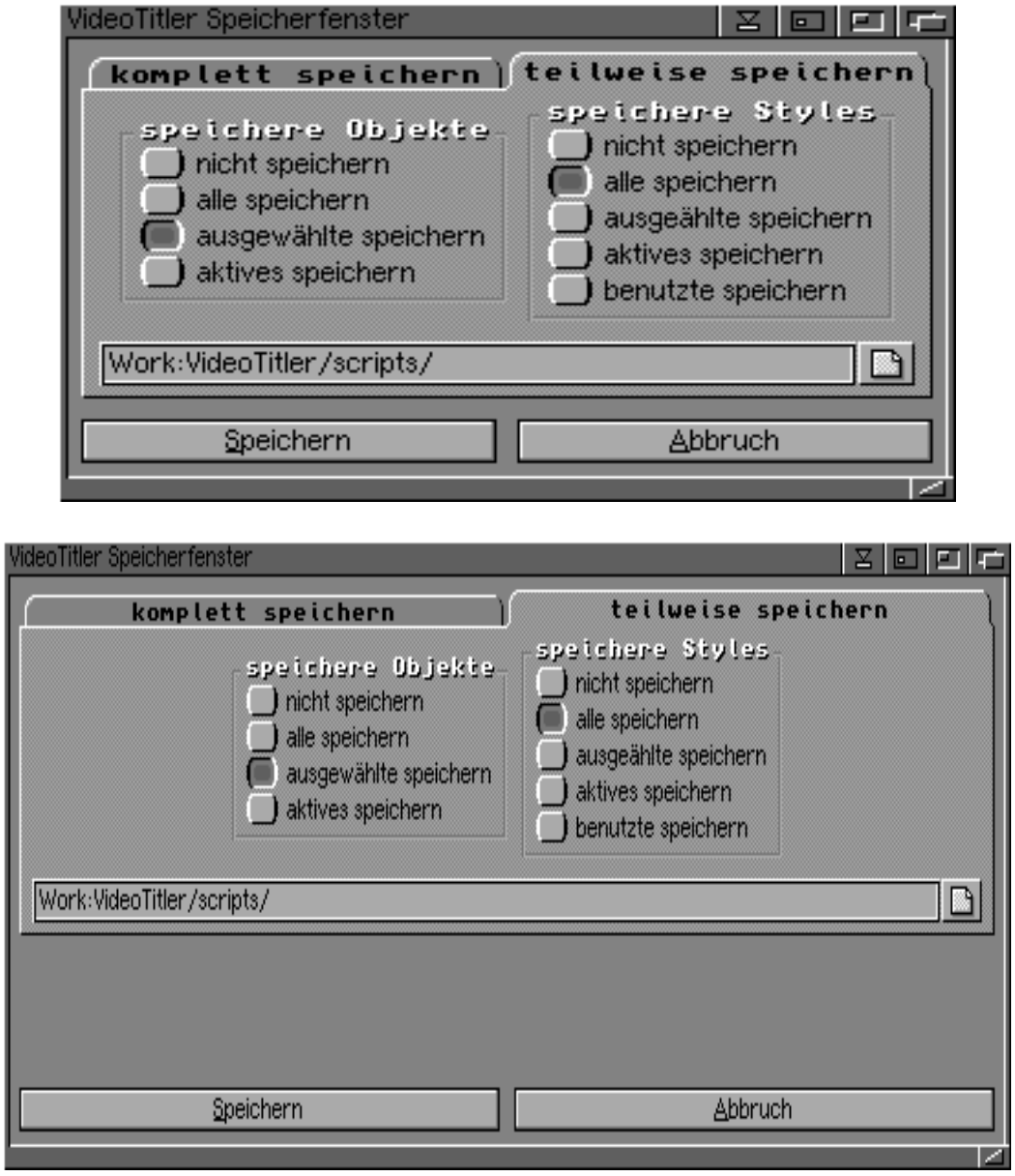

Abbildung 9: *Automatisches Layout bei unterschiedlichen Fenstergrößen*

auch wesentlich auf die Struktur des Programms selbst auswirkt. Anordnung, Beschriftung und Abhängigkeiten der Eingabeobjekte untereinander sind sorgfältig zu bestimmen. Dabei ist immer darauf zu achten, was der Benutzer eigentlich vom Programm erwartet, so daß für ihn die nötigen Bedienschritte, um eine Aufgabe auszuführen, nicht auferlegt und, künstlich, sondern logisch und natürlich erscheinen.

Es wird sicher nur in den seltensten Fällen gelingen, eine optimale Benutzbarkeit zu gewährleisten. Viel zu inhomogen ist oft der Benutzerkreis für ein Programm als daß auf die Bedürfnisse jedes einzelnen eingegangen werden könnte. Auch verhindert oft die Komplexität einer Aufgabe von vorneherein eine 'einfache' Schnittstelle. In einem gewissen Rahmen jedoch, unter Berücksichtigung eigentlich recht weniger Gestaltungsprinzipien, ist es dennoch möglich, dem Anwender einen gutes Stück entgegen zu kommen und Software zu entwickeln, deren Bedienung nicht frustriert, sondern Spaß macht.

Literatur

[1] Balzert H., Hoppe, H.V., Oppermann, R., Reschke, H. Rohr, G. & Streizt, N.A. (Hrsg.) *Einführung in die Software-Ergonomie*
Berlin: deGruyter,1988

[2] Shneiderman B. *Designing the user interface*
Addison-Wesley, 1992

[3] Zieger, J. & Ilg, R. (Hrsg.) *Benutzergerechte Softwaregestaltung*
München: Oldenbourg, 1993

[4] Ackermann, D. & Ulich, E. (Hrsg.) *Software-Ergonomie '91*
Stuttgart: Teubner 1991

[5] Sun Microsystems, Inc. *Open Look - Graphical User Interface Application Style Guidelines*
Addison-Wesley, 1989

[6] Stefan Stuntz *MUI-Magic UserInterface*
http://ftp.uni-erlangen.de/pub/amiga/aminet/dev/mui/mui38dev.lha, 1994-97

[7] Nico François *reqtools.library*
http://ftp.uni-erlangen.de/pub/amiga/aminet/util/libs/ReqToolsDev.lha, 1997