Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Prozessinformatik

Betreuer: Dipl.-Ing. Christian Heller

Hauptseminar
Wintersemester 2003/2004
zum Thema

Recherche und Evaluierung eines Konzeptes für Graphische Benutzeroberflächen in CYBOL

Bearbeiter: Martin Fache
Matrikelnummer: 29714
Termin: 20. Januar 2004

Inhaltsverzeichnis

1. Einleitung

1.1. Problemstellung

Mit CYBOL wurde eine XML-basierte Programmiersprache zur Entwicklung medizinischer Anwendungen geschaffen. Sie basiert auf CYBOP[1], einer Sammlung von neuen Programmierkonzepten, die sich an der Natur orientieren. Mit Hilfe von CYBOL soll es in der Zukunft möglich sein, einfach und problemorientiert medizinische Anwendungen zu implementieren. Zum Zeitpunkt der Erstellung dieser Arbeit fehlen CYBOP Funktionen zur graphischen Ausgabe auf dem Bildschirm.

1.2. Zielsetzung

Die Aufgabe dieser Arbeit ist es, ein Konzept zur graphischen Darstellung von CYBOL-Applikationen zu erforschen. Dabei beschränkt sich der Autor auf die Betriebssystem-Plattform Linux und das X-Window-System, da diese zukünftig das Haupteinsatzgebiet der CYBOL-Anwendungen sein soll. Weiterhin soll ein Prototyp geschaffen werden, der einfache Grundfunktionen implementiert. Eine Integration dieser Grundfunktionen in CYBOL ist nicht Gegenstand dieser Arbeit.

1.3. Aufbau der Arbeit

Kapitel 2 beginnt mit einer grundsätzlichen Diskussion möglicher Konzepte zur graphischen Darstellung. Aus den Ergebnissen des Kapitels 2 ergibt sich die nähere Betrachtung des X-Window-System und der Xlib in Kapitel 3. Anschließend werden die in Kapitel 3 erarbeiteten Grundlagen im Kapitel 4 in einen konkreten Zusammenhang mit CYBOL gebracht.

[1] [cybop.net 2004]

2. Vergleich von Konzepten zur graphischen Ausgabe in CYBOL

Um Konzepte zur graphischen Ausgabe in CYBOL darzustellen ist es nötig CYBOL bzw. dessen Funktionsweise zu verstehen. Nur mit diesem Verständnis ist es möglich die verschiedenen Konzepte im Zusammenhang mit CYBOL zu sehen und zu bewerten. Daher geht Abschnitt 2.1. auf die Funktionsweise bzw. die Philosophie von CYBOL ein.

2.1. Funktionsweise von CYBOL

CYBOP steht für "Cybernetics Oriented Programming". Es handelt sich dabei um eine Sammlung von Konzepten der Softwareentwicklung, die sich an der Natur orientieren. Mit CYBOL wurde eine auf XML basierende Programmiersprache geschaffen[2], die sich an den Prinzipien des "Cybernetics Oriented Programming" orientiert. CYBOL besteht zum einen aus hierarchisch aufgebauten statischen Strukturen und zum anderen aus einem ebenfalls hierarchisch aufgebauten dynamischen Teil, mit dem die Prozesse eines Softwaresystems abgebildet werden. Beide Teile sind komplett in Form von XML-Dateien abgelegt und miteinander verknüpft. Diese XML-Strukturen werden vom so genannten CYBOI, dem Interpreter der Sprache gelesen und im Speicher bereitgestellt. Der Interpreter führt also Programme, die in CYBOL geschrieben sind, aus. Für eine typische graphische Anwendung werden also Beispielsweise Definitionen zur Größe des Fensters, Art und Position eines Navigationsmenüs, Anzahl und Beschriftung der Menüeinträge, Position und Art von Buttons und vieles mehr benötigt. Diese Informationen sind im statischen Teil von CYBOL hierarchisch verknüpft festgelegt und werden vom Interpreter gelesen und in einer festgelegten Form im Speicher hinterlegt.

2.2. Mögliche Realisierung einer graphische Benutzeroberfläche für CYBOL

Die Aufgabe einer Software zur Darstellung des Inhaltes in graphischer Form ist es, diese Strukturen zu erkennen und aus diesen Informationen eine graphische Oberfläche zu zeichnen. Bei der Realisierung einer solchen Software lassen sich grundsätzlich zwei verschiedene Vorgehensweisen unterscheiden. Die erste Möglichkeit ist, ein bestehendes graphisches Toolkit, wie Tcl/Tk, Gtk+, Qt, OSF/Motif oder wxWindows zu verwenden. Dabei müssen die im Speicher abgebildeten Strukturen den verschiedenen im Toolkit definierten Widgets[3] zugewiesen werden. Diese Möglichkeit wird in Abschnitt 2.2.1. Widget-Toolkits behandelt.
Die zweite Vorgehensweise ist eine Realisierung mit Basisfunktionalitäten des jeweiligen Betriebssystems. Bei Linux handelt es sich dabei um das X11-Window-Systems, welches Standard zur graphischen Darstellung unter Linux ist und mit der Xlib[4] gesteuert werden kann. In diesem Fall besteht die Hauptaufgabe der Software darin, die im Speicher vorhandenen Informationen zu analysieren und diese dann mit Hilfe von Xlib-Zeichenroutinen abzubilden. Diesen Weg beschreibt Abschnitt 2.2.2.

[2] [cybop.net 2004]
[3] vgl. Kapitel 2.2.1.
[4] vgl. Kapitel 2.2.2.

2.2.1. Widget-Toolkits

Als Widget-Toolkits werden Programmbibliotheken bezeichnet, die so genannte Widgets mit einem in der Regel gemeinsamen Look & Feel definieren. Widget ist ein Kunstwort der IT-Welt, was aus den Worten „Window“ und „Gadget“ besteht. Laut Langescheits Wörterbuch ist ein Gadget ein Apparat. Also ist ein Widget wörtlich übersetzt ein „Fensterapparat“, wobei mit Widgets alle visuellen Teile eines Programms gemeint sind, wie zum Beispiel ein Menü, ein Menüeintrag, eine Statusleiste oder ein beliebiger Button.
Vertreter von Widget-Toolkits sind zum Beispiel das GTK- und Tcl/Tk-Toolkit aus der Open-Source-Welt.

Eine mit einem beliebigen Toolkit realisierte graphische Oberfläche für CYBOL muss Graphik-Strukturen bzw. –Objekte im Speicher erkennen und diesen CYBOL-Objekten jeweils ein Widget aus dem Toolkit zuweisen.
Der Vorteil bei der Verwendung eines schon vorhandenen Toolkits liegt darin, dass die einzelnen GUI-Elemente bzw. Widgets nicht aufwendig selbst entworfen und implementiert werden müssen. Man kann die Entwicklung der Widgets anderen Programmierern überlassen, die ihre Toolkits in der Regel unter der GPL veröffentlichen. Rechtlich steht dem Einsatz eines unter der GPL veröffentlichten Toolkits nichts im Weg. Ein Einsatz würde CYBOL aber abhängig von anderen Programmierern bzw. deren Projekten machen. Ein weiterer Nachteil in diesem Zusammenhang sind mögliche Konflikte von CYBOL-Programmversionen und verschiedenen Versionen des gewählten Toolkits. Beispielsweise gibt es gravierende Unterschiede zwischen den sehr intensiv weiterentwickelten Tcl/Tk-Versionen[5]. Versionskonflikte sind vorprogrammiert. Außerdem ist es den CYBOL-Entwicklern nicht möglich, aktiv die Entwicklung dieses Toolkit zu beeinflussen. Möglicherweise besitzen Widgets nicht die von CYBOL geforderte Funktionalität. Oder andere Eigenschaften wie Farbe und Größe lassen sich nicht ausreichend beeinflussen, da die Eigenschaften in CYBOL sehr detailliert beschrieben sind und auch so abgebildet werden sollen. Eine Konsequenz könnte sein, dass Änderungen an CYBOL gemacht werden, um den Ansprüchen des Toolkits zu genügen. Das ist aber nicht im Sinn der CYBOL-Entwickler.

2.2.2. X11-Basisfunktionalität mit der Xlib-Bibliothek

Die Entwickler von X11 haben dem Programmierern von graphischen Oberflächen für das X-Windows-System ein mächtiges Werkzeug zur Verfügung gestellt. Die Xlib-Bibliothek. Sie ist eine Sammlung von Funktionen mit denen ein Entwickler Windows erzeugen, verändern und zerstören können, ohne direkt per X-Protokoll mit dem X-Server kommunizieren zu müssen. Auf die verschiedenen Komponenten von X11 und ihr Zusammenspiel geht Abschnitt 3. „Die Funktionsweise von X11 mit der Xlib“ näher ein.

Die Verwendung von Xlib-Funktionen kommt dem CYBOL-Konzept sehr entgegen, weil die Struktur der Anwendung sehr detailliert im Speicher beschrieben ist und sich die graphische Anwendung alleine auf das sture Zeichnen der GUI-Elemente beschränken kann. Im CYBOL-Konzept ist vorgesehen, dass die Anwendungslogik einmal auf der Ebene der XML-Definitionen und zum anderen auf der Ebene des XML-Interpreters implementiert werden soll. Daraus ergibt sich der Vorteil, dass bei der Darstellung einer CYBOL-Anwendung verschiedene Wege wie zum Beispiel die Darstellung in einer Konsole oder über Webtechnologien die Anwendungslogik nur einmal implementiert werden muss. Die

[5] [tcl.kt 2004]

Implementierung der verschiedenen Darstellungswege unterscheidet sich naturgemäß, aber die Anwendung soll immer gleich funktionieren.

Ein Nachteil bei der Verwendung der Xlib-Bibliothek ist die sehr aufwendige Implementierung der einzelnen Elemente. Aber damit kann sichergestellt werden, dass diese Elemente genau die Eigenschaften und Funktionen der CYBOL-Struktur repräsentieren. Auch ergeben sich keine Abhängigkeiten des CYBOL-Codes von anderen Programm-Bibliotheken[6].

Da es den Entwicklern bei CYBOL um eine konsequente Neuentwicklung geht, wiegt das Argument der höheren Kompatibilität der Xlib-Bibliothek zum CYBOL-Konzept mehr als eine möglicherweise einfachere Implementierung einer graphischen Oberfläche mit einen bereits bestehenden Widget-Toolkit. Daher beschränkt sich diese Arbeit im Folgenden auf die Betrachtung einer Implementierung mit Hilfe der Xlib-Bibliothek.

[6] vgl. Kapitel 2.2.1

3. Die Funktionsweise von X11 mit der Xlib

X ist eine portable Standard-Software, die am MIT (Massachusetts Institut of Technologie) vor allem von Robert W. Scheifler und Jim Gettys entwickelt wurde. Mit Hilfe von X ist es möglich, komplexe graphische Anwendungen auf dem Bildschirm darzustellen. Es existieren für alle heutigen Rechnerarchitekturen Implementierungen des X-Window-Systems (kurz X11). Im Folgenden wird darauf eingegangen, wie X11 funktioniert. Kapitel 4 geht darauf ein, welche Komponenten in CYBOL eingesetzt werden können und wie dies geschehen kann.

3.1. Ein Überblick über die X11-Architektur

Der Kern des X-Window-System ist das Basis-Window-System, was oft auch als X-Server bezeichnet wird. Der X-Server kommuniziert hardwareabhängig mit der Rechner-Hardware auf dem er installiert ist. Er stellt dem übrigen X-Window-System Funktionen zur Darstellung von Graphik über das X-Netzprotokoll zur Verfügung. Es gibt für eine graphische Anwendung keine Möglichkeit direkt mit dem X-Server zu kommunizieren oder direkt Grafikhardware, Maus oder Tastatur zu beeinflussen. Jede Kommunikation mit dem X-Server findet über das X-Netzprotokoll statt. Dadurch ist es möglich, die Kommunikation der Anwendung mit dem X-Server vollkommen hardwareunabhängig zu gestalten.

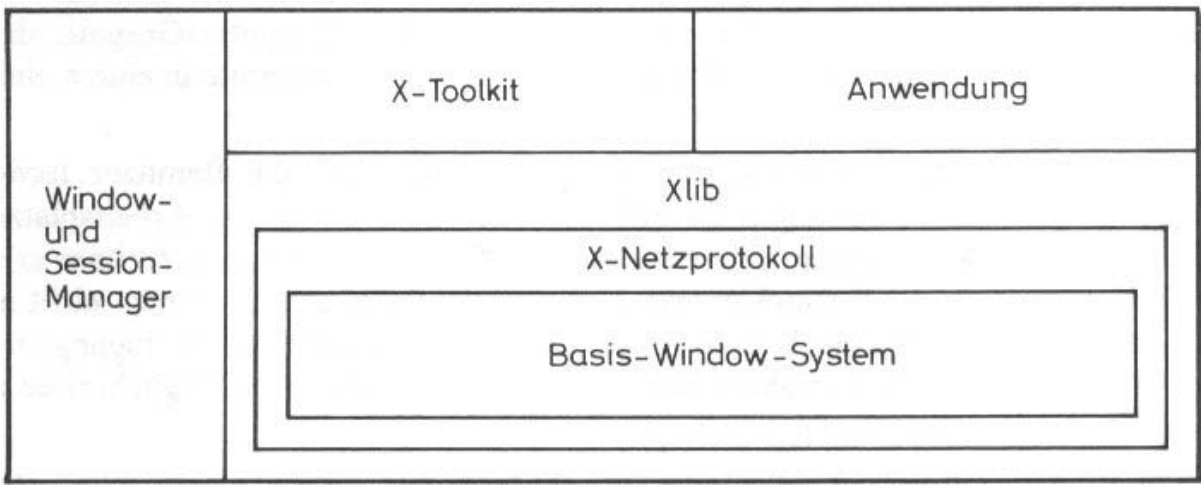

Abb. 3-1 Die Struktur von X[7]

Wie auf Abbildung 3-1 zu erkennen ist, benutzen Anwendungen in der Regel nicht direkt das Netzprotokoll, sondern die X-Library, die kurz Xlib genannt wird. Die Xlib ist eine Schnittstelle zwischen dem X-Netzprotokoll und einer X-Anwendung. Sie besteht aus ca. 500 Routinen, die alle im X-Netzprotokoll enthaltenen Funktionen abbilden. Außerdem stellt die Xlib dem Anwendungsprogrammierer geeignete Funktionen zur Verfügung, mit denen die gebräuchlichsten Anwendungsfälle abbildbar sind. Damit stellt die Xlib die niedrigste Schnittstelle mit dem X-Server dar. Auf die einzelnen Routinen gehen die Abschnitte 3.3. bis 3.9. genauer ein.

[7] Abbildung entnommen aus [JONES 1991 S.2]

Das X-Toolkit, kurz Xtk genannt, ist eine Sammlung von GUI-Elementen. Das Xtk unterscheidet sich von anderen Toolkits dadurch, dass es zur Basis-Installation eines X11-systems gehört. Es existieren weitere Widget-Toolsets, wie das OSF/Motif Widget-Set, welche auf dem X-Toolkit basieren. Die Vor- und Nachteil solcher Toolsets wurden bereits im Abschnitt 2.2.1. Widget-Toolkits hinreichend beschrieben.

Zu einem X-Window-System gehört auch immer ein Window-Manager. Dieser ist für die Verwaltung der verschiedenen Fenster auf einem Bildschirm verantwortlich. Das schließt das Verändern der Fenstergröße, das Verschieben des Fensters auf dem Bildschirm, das Stapeln bzw. Tiling der einzelnen Fenster ein. Diese Funktionen müssen von graphischen Anwendungen nicht selbst implementiert werden. Der Window-Manager wird vom X-Server aber auch nur als Anwendung behandelt. Beispiele für Window-Manager sind unter anderem fvwm, window-maker oder kwm, der Window-Manager von KDE.

Der so genannte Session-Manager, manchmal auch Display-Manager genannt, ist ein Programm, welches dem Benutzer ein grafisches Login bietet und darüber hinaus meist eine graphische Auswahl des in der Session zu benutzenden Window-Manager bietet, wie bei den Display-Managern kdm und gdm üblich.

3.2. Der Screen

Im Zusammenhang mit X11 versteht man unter Screen in der Regel den Ausgabebildschirm. X11 unterstützt die gleichzeitige Ansteuerung von mehreren Screens durch einen X-Server. So ist es zum Beispiel möglich, dass ein lokaler Benutzer am lokalen Bildschirm arbeitet, während ein zweiter Benutzer an einem anderen X-Terminal im Netz mit einem zweiten Screen des lokalen X-Servers arbeiten kann. Auch ist es möglich, dass an einem Rechner zwei oder mehr Bildschirme angeschlossen sind, die von einem X-Server verwaltet werden, wie die Abbildung 3-2 verdeutlicht.

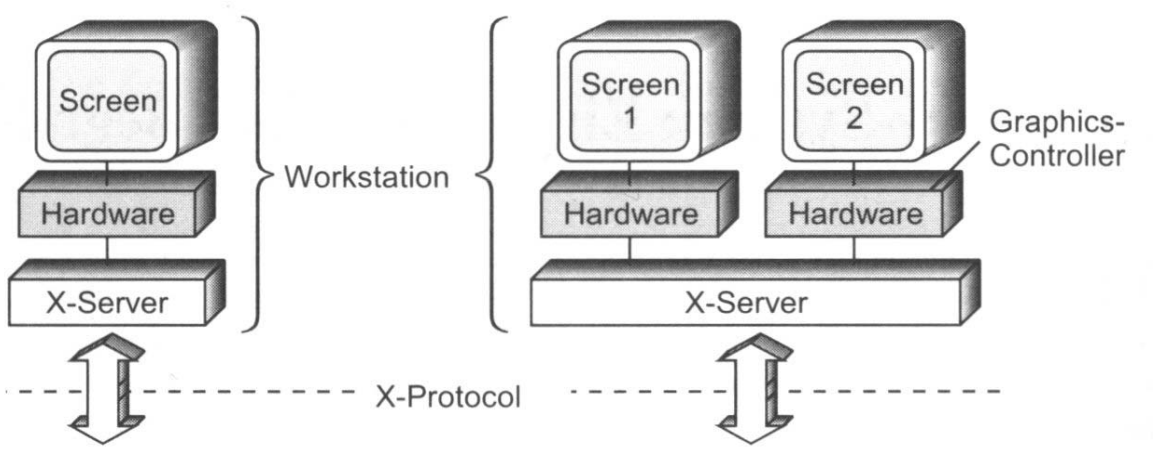

Abb. 3-2 Single- und Multi-Screen-Umgebung[8]

[8] Abbildung aus [KÖRBER u.a. 1998]

3.3. Das Display

Unter Display versteht man in X11 die Zusammenfassung der Komponenten X-Server, die von X-Server verwalteten Screens und die beteiligten Eingabegeräte, also die komplette dem X-Server zugeordnete Hardware. In X11 ist eine Datenstruktur mit dem namens Display definiert, die Informationen zum jeweils verwendeten Display enthält.

Einer Anwendung, die auf einem Screen dargestellt werden soll, muss das zu verwendende Display und der zu verwendende Screen bekannt sein. Eine Verbindung der Anwendung zum Display wird mit

cybol_display = XOpenDisplay(display_name);

hergestellt, wobei in *cybol_display* ein Zeiger auf die oben genannte Datenstruktur zurückgegeben wird. In d*isplay_name* ist ein Zeiger auf den Namen des Displays gespeichert, mit welchem die Anwendung verbunden werden soll. Dies kann ein lokales Display, aber auch das Display eines per Netzwerk angebundenen Rechners sein. Auch ist es möglich bei mehreren angeschlossenen Bildschirmen die Bildschirmnummer anzugeben. Wenn zum Beispiel der zweite Bildschirm des ersten Displays auf dem Rechner piggy.rz.tu-ilmenau.de angesprochen werden soll, muss *display_name* piggy.rz.tu-ilmenau.de:0.1 heißen. Der Rang der Displays bzw. der Screens beginnt hier mit 0. *XOpenDisplay()* ist eine Standardfunktion der Xlib.

3.4. Ressourcen

Eine X11-Anwendung benötigt zur Steuerung des Arbeitsplatzrechners Zugriff auf Ressourcen des X-Servers. Während der Laufzeit werden viele Ressourcen erzeugt, verändert und auch wieder zerstört. Die Ressourcen existieren im X-Server. Jede Ressource kann von allen X11-Anwendungen genutzt werden, die vom X-Server verwaltet werden. Konkrete Ressourcen in X11 sind Windows, Graphikkontext, Schriftarten, Farbtabellen, Pixmaps und Cursor. Diese werden im Folgenden kurz erläutert.

3.5. Windows

Ein Window bezeichnet in X11 ein Fenster. Es kann mit dem Xlib-Aufruf *XCreateSimpleWindow* ein einfaches Fenster erzeugt werden.

```
cybol_window = XCreateSimpleWindow (cybol_display,
                    DefaultRootWindow(cybol_display),
                    position_x, position_y,
                    width, length, border_width);
```

Der Xlib-Funktion wird das Display übergeben, auf dem das Fenster dargestellt werden soll. Mit der zweiten Angabe wird dem Fenster das Vater-Fenster zugewiesen. Unter X11 existiert immer mindestens ein Fenster, das Fenster des Window-Managers. Mit der Angabe des Vater-Fensters wird dem neu erzeugten Fenster ein Platz in der Fensterhierarchie zugewiesen und es erbt bestimmte Eigenschaften des Vater-Fensters. Es wäre möglich, ein anderes Vater-

Fenster als das Fenster des Window-Managers anzugeben. Bei der Darstellung eines Popups wäre es sinnvoll als Vater-Fenster das Fenster des Hauptprogramms anzugeben.

Mit dieser Funktion wird das Fenster erzeugt, aber noch nicht abgebildet. Dies übernimmt die Xlib-Funktion XMapRaised mit

XMapRaised (cybol_display, cybol_window);

Diese Funktion ändert die Stapelreihenfolge der Geschwister-Fenster so, dass das neu erzeugte Fenster ganz oben liegt. Danach werden alle Geschwister-Fenster auf dem Bildschirm abgebildet.

3.6. Graphikkontext (GC)

Der GC (Graphikkontext, engl.: Graphic Context) ist eine Ressource, auf die sich die Elemente beziehen, die mit der Xlib gezeichnet werden. Im CG sind Eigenschaften wie Hintergrundfarbe, Vordergrundfarbe, Schriftart, Füllmuster, Linienbreite gespeichert. Zum Zeichnen eines Graphikelementes ist es nötig einen GC anzugeben. Das Element wird darauf hin mit den im angegebenen CG gespeicherten Eigenschaften gezeichnet.

Ein GC wird mit dem Aufruf

cybol_gc = XCreateGC (cybol_display, cybol_window, 0, 0);

erzeugt. Die letzten beiden Parameter (*value, valuemask*) haben in diesem Bsp. den Wert 0. Dies ist der einfachste Weg zur Erzeugung eines GCs. Mit diesen Parametern können Eigenschaften des GCs direkt bei der Erzeugung beeinflusst werden. Sind die Parameter auf 0 gesetzt, wird ein GC mit den Default-Einstellungen erzeugt. Zur Beeinflussung der Eigenschaften eines GCs verwendet man entweder die Funktion *XChangeGC*, der wiederum die Parameter *value* und *valuemask* übergeben werden müssen, oder die so genannten Convenience-Funktionen, die von der Xlib bereitgestellt werden. Zur Änderung der Vordergrundfarbe wird die Funktion

XSetForeground (cybol_display, cybol_gc, cybol_foreground);

aufgerufen, wobei *cybol_foreground* die Pixelfarbe bezeichnet. Nähere Informationen zum Thema Pixel-Werte und Farben enthält der Abschnitt 3.8.
Laut Oliver Jones „Einführung in das X-Windows-System“[9] wird die Performance der Anwendung durch die Verwendung der Convenience-Funktionen, statt der Funktion *XChangeGC* nicht wesentlich beeinflusst, aber die Lesbarkeit des erzeugten Codes stark erhöht. Dem wiedersprechen Körber und Frank[10], die wesentlichen Performance-Vorteile der Funktion *XChangeGC* gegenüber einer Folge von Convenience-Funktionen sehen.

Ein GC kann von verschiedenen Elementen genutzt werden. Auch ist es möglich, GCs anderer Anwendungen, zum Bsp. die des Desktop-Fensters, zu benutzen. Davon ist aber abzuraten, da sich die Eigenschaften eines GCs zur Laufzeit ändern können.

[9] [JONES 1991 S. 141]
[10] [KÖRBER u.a. 1998 S. 197f]

3.7. Schriftarten

In X11 können alle Schriftarten (engl.: Fonts) gezeichnet werden, die auf dem System installiert sind. Dazu muss die Schriftart mit

cybol_font = XLoadFont (cybol_display, Fontname);

geladen werden. Wie schon im Abschnitt 3.6. beschrieben, beziehen sich zu zeichnende Elemente immer auf einen GCs. Diesem GC wird mit

XSetFont (cybol_display, cybol_gc, cybol_font);

die geladene Schriftart zugewiesen. Nun ist es möglich mit

XDrawString (cybol_display, cybol_window, cybol_gc,
position_x, position_y, darzustellender_String, strlength);

einen einfachen String in der oben gewählten Schriftart darzustellen.

3.8. Farben

Unter X11 existiert für jedes Display eine Standard-Farbtabelle, die Default-Colormap. In dieser Farbtabelle sind alle Farben, die aktuell von allen X11-Anwendung eines Displays verwendet werden festgelegt und mit einem Index versehen. Eine Applikation kann entweder einen Eintrag aus dieser bestehenden Farbtabelle benutzen oder einen neuen Eintrag anlegen.

Es gibt prinzipiell zwei Möglichkeiten für eine Anwendung, eine Farbe in dieser Tabelle zu verändern. Zum einen kann eine Anwendung einen Eintrag exklusiv für sich reservieren. Diesen Eintrag kann die nach dem Anlegen beliebig verändern. Andere Applikationen haben keinen Zugriff auf den Eintrag. Die Zweite Möglichkeit besteht darin, einen Eintrag in der Farbtabelle für die gemeinsame Nutzung zu reservieren. Der einmal angelegte Eintrag ist dann nicht mehr änderbar, da auch andere Anwendungen simultan auf den Farbeintrag zugreifen könnten.
Dabei ist zu beachten, dass die Anzahl der Einträge in der Default-Colormap begrenzt sind. Sollte eine Applikation mehr Farbdefinitionen benötigen, als in der Standard-Farbtabelle noch verfügbar sind, besteht die Möglichkeit eine eigene Farbtabelle anzulegen. Da X11 nicht unbegrenzt Farben darstellen kann, muss beim Zugriff auf die selbst definierte Farbtabelle die originale Default-Colormap durch die neue Farbtabelle ersetzt werden. Das hat gravierende Auswirkungen auf andere Applikationen, die weiterhin auf den Index der alten Farbtabelle zugreifen wollen, sich hinter dem Index aber jetzt ein neuer Wert verbirgt. Es ist also nicht empfehlenswert eigene Farbtabellen zu verwenden.

Ein Ausweg aus dem Dilemma der begrenzten Farbtabelle ist die Verwendung von bereits vorhandenen Farben. Dazu muss die aktuelle Default-Colormap geladen werden.

cybol_color_map = DefaultColormap (cybol_display, cybol_screen);

Zum Auslesen eines Index-Wertes wird die Funktion XAllocColor verwendet. Dieser Funktion ist sind RGB-Grundfarbenwerte (auf einer Skala von 0 bis 65535) in der Form

```
gray.red = 32768; gray.green = 32768; gray.blue = 32768;
XAllocColor (cybol_display, cybol_color_map, &gray);
```

zu übergeben. XAllocColor liefert in gray.pixel den Index der Farbe in der Farbtabelle, die der übergeben Struktur am nächsten kommt, zurück. Die Variable gray ist eine XColor-Struktur.

Um diese Farbe zum Zeichnen eines Elementes zu verwenden, muss diese noch einem Graphikkontext zugewiesen werden. Um die Vordergrundfarbe zu ändern wird dabei

```
XSetForeground (cybol_display, cybol_gc, gray.pixel);
```

genutzt.

Die hier erläuterten Funktionen stellen nur ein Bruchteil der Möglichkeiten der Xlib zur Manipulation von Farben dar.

3.9. Events

Bei der Benutzung einer Anwendung werden durch Aktionen des Benutzers eine Reihe von Events ausgelöst. Aktionen des Benutzers sind zum Beispiel das Verschieben des Fensters, einem Mausklick oder eine Tastatureingabe. Die durch diese Aktionen ausgelösten Events werden, versehen mit einem Zeitstempel, in die Event-Schlange (engl. Event-Queue) der Anwendung geschrieben werden. In diese Schlange werden nur Events aufgenommen, die von der Anwendung mit der Funktion *XSelectInput* angefordert werden.

```
XSelectInput (cybol_display, cybol_window, ButtonPressMask);
```

fordert zum Beispiel mit Hilfe der Maske *ButtonPressMask* Events an, die im Zusammenhang mit dem Drücken einer Maustaste stehen. Mit Hilfe dieser Masken kann die Übermittlung der Events an die Event-Schlange explizit übermittelt werden.

4. Die graphische Oberfläche für Cybol – Eine Beispielanwendung

Im Folgenden werden Teile einer Beispielanwendung beschrieben, die im Zusammenhang mit diesem Hauptseminar entstanden ist. Sie ist Grundlage für die Graphische Ausgabe in Cybol. Die Anwendung wurde in C geschrieben.

4.1. Initialisierung

Da es sich in diesem Stadium um ein, vom übrigen CYBOL-Code losgelöstes, eigenständiges Programm handelt, werden als erstes Strukturen im Speicher erzeugt, die den zukünftigen vom CYBOL-Interpreter erzeugten Strukturen ähneln. Dies erfolgt mit Hilfe einiger struct-Anweisungen.

```
struct menu_item {
   char name[50];
};
struct menu {
   char name[50];
   int angeklickt;
   struct menu_item menu_items[20]; //max. 20 Items
};
struct menu_bar {
   struct menu menus[5]; // max. 5 Menues
};
struct frame {
   int size_x;
   int size_y;
   struct menu_bar menu_bar1;
} cybol_anwendung;
```

Diese werden anschließend mit realen Werten gefüllt. Zur Initialisierung gehören auch alle bereits in Kapitel 3 ausführlich beschriebenen Initialisierungsroutinen der Xlib. Es muss eine Verbindung zum Display hergestellt werden[11] und ein einfaches Fenster erzeugt werden[12]. Danach werden mehrere Graphic Contexts erzeugt, bei denen die Attribute für Schriftarten[13] und Farben[14] angepasst werden. Mehrere Graphic Contexts erscheinen sinnvoll, da mehrere Farben zum Zeichen verwendet werden sollen. Nach der Initialisierung der Graphic Contexts müssen noch Events angefordert werden[15], wobei die Anwendung auf Tastendruck, Mausklick und Veränderung der Fenstergröße, Überdeckung durch ein anderes Fenster, Wiederaufbau des Fensters reagieren soll. Dies erfolgt mit

```
XSelectInput (cybol_display, cybol_window,
   ButtonPressMask | KeyPressMask | ExposureMask);
```

Danach wird das Fenster erstmalig gezeichnet[16].

[11] vgl. Kapitel 3.3
[12] vgl. Kapitel 3.5
[13] vgl. Kapitel 3.7
[14] vgl. Kapitel 3.8
[15] vgl. Kapitel 3.9
[16] vgl. Kapitel 3.5

4.2. Die Event-Schleife

Die Elemente der Applikation werden erst innerhalb einer Schleife gezeichnet, die durch eine Abbruchbedingung verlassen werden kann. Das Zeichnen erfolgt immer nach einen bestimmten Event, dem Expose-Event. Das Expose-Event wird immer dann vom Window-Manager an die Anwendung gesendet, wenn sich die Eigenschaften durch eine Interaktion des Benutzer ändern. Dies wäre zum Bespiel das Ändern der Fenstergröße oder das Aufdecken des Anwendungs-Fenster. Da die CYBOL-GUI immer genau den aktuellen Zustand der Anwendung im Speicher zeichnen soll, ist es sinnvoll, dass auch die CYBOL-Applikation selbst solche Expose-Events auslöst, wenn sich der Zustand der Anwendung ändert. Auch das erstmalige Zeichnen des Fensters löst einen solchen Expose-Event aus. Die Schleife der Beispiel-Anwendung sieht so aus:

```
done = 0;
while (done ==0) {
 XNextEvent (cybol_display, &cybol_event);
 switch (cybol_event.type) {
  case Expose: {
   //Aktionen nach einem Expose-Event
  }
 }
```

4.3. Die graphischen Elemente

Beim Auftreten eines Expose-Events soll der aktuelle Zustand im Speicher abgebildet werden. Dazu werden im Folgenden Elemente auf ihr Vorhandensein geprüft und anschließend gezeichnet.

4.3.1. Die Menüleiste

Die Menüleiste ist ein unverzichtbares Element jeder graphischen Anwendung. Daher wird diese immer gezeichnet. Da die Fenstergröße variieren kann, je nach dem, wie der Benutzer diese bestimmt, wird mit

```
XGetWindowAttributes (cybol_display, cybol_window, &cybol_window_attributes);
```

unter anderem die aktuelle Fenstergröße ermittelt. Mit Hilfe dieser Daten kann die Menüleiste über ganze Fensterbreite gezeichnet werden:

```
XDrawLine (cybol_display, cybol_window, gc_menu_border_bottom, 0, 21,
        cybol_window_attributes.width, 21);
XDrawLine (cybol_display, cybol_window, gc_menu_border_bottom,
        (cybol_window_attributes.width-1), 1, (cybol_window_attributes.width-1), 21);
XFillRectangle (cybol_event.xexpose.display, cybol_event.xexpose.window, gc_menu,
        1, 1, (window_attributes.width-2), 20);
```

Die ersten beiden Befehle zeichnen jeweils einen dunklen Rahmen um die Menüleiste, damit sie sich optisch vom Hintergrund besser abhebt. Der Dritte Befehl zeichnet die eigentliche Menüleiste.

Mit den folgenden Anweisungen werden die Menüpunkte aus dem Speicher gelesen und nach Überprüfung gezeichnet:

```
for (count_menu=0;count_menu<5;count_menu++) {
    if (strlen(Anwendung.menu_bar1.menus[count_menu].name)>0) {
        XDrawImageString (cybol_event.xexpose.display,
        cybol_event.xexpose.window, gc_menu_font, (5+indent_x), 16,
        cybol_anwendung.menu_bar1.menus[count_menu].name,
        strlen(cybol_anwendung.menu_bar1.menus[count_menu].name));
   }
   indent_x = indent_x +
  (strlen(cybol_anwendung.menu_bar1.menus[count_menu].name) * 6) +10;
}
```

Dabei ist zu bemerken, dass dieses Code-Fragment noch nicht sehr ausgereift ist. Die Positionierung der Menüpunkt-Bezeichnung beruht auf der Forderung, dass eine Festbreitenschriftart genutzt wird, bei der alle Buchstaben eine feste Breite von 6 Pixeln besitzen. Auch ist fraglich, ob die Positionsbestimmung überhaupt erst beim Zeichnen ausgeführt wird, oder ob diese Vorgabe nicht schon durch die CYBOL-Applikation im Speicher festgehalten werden sollte.

4.3.2. Die Menüeinträge

Menüeinträge sollen angezeigt werden, wenn diese angeklickt oder per Tastatur aktiviert werden. Dies wird in diesem Modell durch das Attribut *angeklickt* dargestellt. Ist der Wert dieses Attributes gleich *1*, wird das entsprechende Menü gezeichnet

```
for (count_item=0; ((count_item<19) && (cybol_anwendung.menu_bar1
.menus[count_menu].angeklickt==1)); count_item++) {
   if (strlen(cybol_anwendung.menu_bar1.menus[count_menu].
   menu_items[count_item].name)>0) {
        indent_y = indent_y + 17;
        XFillRectangle (cybol_event.xexpose.display, cybol_event.xexpose.window,
        gc_menu, (3+indent_x), 20 + (count_item*17), indent_menu_item_x, 19);
        XDrawImageString (cybol_event.xexpose.display, cy-
        bol_event.xexpose.window, gc_menu_font, (5+indent_x),
        33 + (count_item*17), cy-
        bol_nwendung.menu_bar1.menus[count_menu].menu_items[count_item]
        .name, strlen(cybol_nwendung.menu_bar1.menus[count_menu]
        .menu_items[count_item].name));
   }
}
```

Auch hier ist zu bemerken, dass dieses Code-Fragment noch weit von einem Einsatz in einer realen Anwendung entfernt ist. Auch ist hier Programmlogik zur Positionsbestimmung enthalten, deren Ergebnisse eher durch die Applikation schon vor dem Aufruf der Zeichenroutine berechnet und im Speicher bereitgestellt werden sollten. Das Ergebnis der Zeichenroutinen der Beispielanwendung ist in Abb. 4-1 zu sehen.

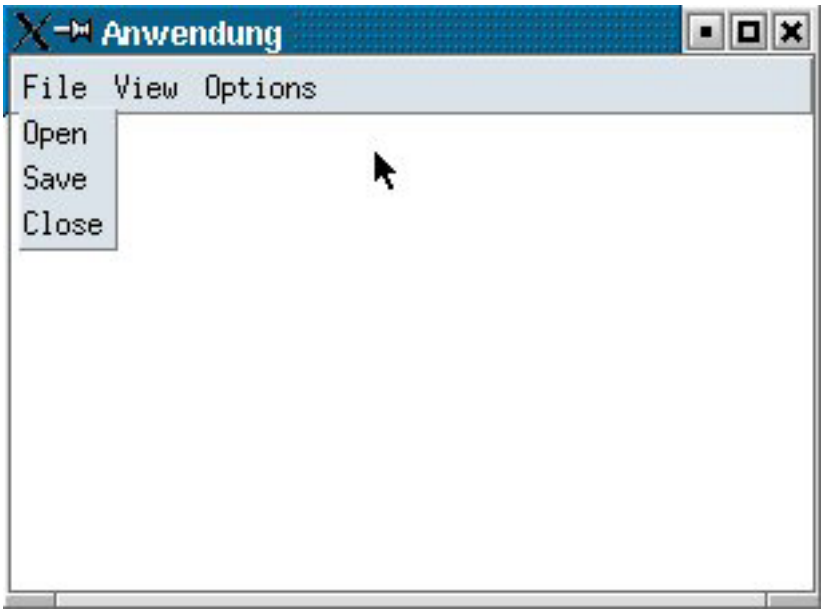

Abb. 4-1 Beispielanwendung mit aktiviertem Menü

4.4. Interaktion mit dem Benutzer

Zur Demonstration von Möglichkeiten der Interaktion mit dem Benutzer werden, wie in Kapitel 4.2 erwähnt, neben den Expose-Events auch Events, die beim Drücken einer Maustaste oder einer Taste des Keyboards ausgelöst werden, von der Anwendung angefordert. Diese Events sollten später nicht Teil der Grafik-Anwendung sein, sondern Teil des Erstellungsprozesses der im Speicher gehaltenen Informationen zur Darstellung der Anwendung.

Zur Demonstration der Funktionsweise wurde die Event-Schleife um einige Bedingungen ergänzt. Die folgenden Anweisungen werten einen Klick einer Maustaste aus. Dabei wird in Abhängigkeit der Position zum Zeitpunkt des Mausklicks ein Menü aktiviert oder alle Menüs deaktiviert. Nach dem Festlegen des neuen Zustandes wird durch XClearArea das komplette Fenster von allen Zeichnungen gereinigt und ein Expose-Event ausgelöst, der beim nächsten Durchlauf der Event-Schleife das erneute Zeichnen des neuen Zustandes bewirkt.

```
case ButtonPress:
  if ((cybol_event.xbutton.x<30) && (cybol_event.xbutton.x>3) &&
 (cybol_event.xbutton.y<21) && (cybol_event.xbutton.x>1)) {
    Anwendung.menu_bar1.menus[0].angeklickt = 1;
    Anwendung.menu_bar1.menus[1].angeklickt = 0;
    Anwendung.menu_bar1.menus[2].angeklickt = 0;
    XClearArea (cybol_display, cybol_window, 0, 0, 0, 0, True);
 }
 else if ((cybol_event.xbutton.x<65) && (cybol_event.xbutton.x>38) &&
(cybol_event.xbutton.y<21) && (cybol_event.xbutton.x>1)) {
    Anwendung.menu_bar1.menus[0].angeklickt = 0;
    Anwendung.menu_bar1.menus[1].angeklickt = 1;
    Anwendung.menu_bar1.menus[2].angeklickt = 0;
    XClearArea (cybol_display, cybol_window, 0, 0, 0, 0, True);
 }
```

```
 else if ((cybol_event.xbutton.x<120) && (cybol_event.xbutton.x>70) &&
(cybol_event.xbutton.y<21) && (cybol_event.xbutton.x>1)) {
    Anwendung.menu_bar1.menus[0].angeklickt = 0;
    Anwendung.menu_bar1.menus[1].angeklickt = 0;
    Anwendung.menu_bar1.menus[2].angeklickt = 1;
    XClearArea (cybol_display, cybol_window, 0, 0, 0, 0, True);
 }
 else {
    Anwendung.menu_bar1.menus[0].angeklickt = 0;
    Anwendung.menu_bar1.menus[1].angeklickt = 0;
    Anwendung.menu_bar1.menus[2].angeklickt = 0;
    XClearArea (cybol_display, cybol_window, 0, 0, 0, 0, True);
 }
break;
```

Neben den Mausklicks wird exemplarisch noch der Druck der Taste *q* geprüft, um beim Druck der Taste *q* die Event-Schleife und anschließend das Programm zu verlassen.

```
    case KeyPress:
        i = XLookupString ( &cybol_event, text, 10, &cybol_key, 0);
        if ( i == 1 && text[0] == 'q') done = 1;
        break;
```

Damit wird die Event-Schleife verlassen, da die Bedingung *done=1* zum Durchlaufen der Schleife nicht mehr gegeben ist. Anschließend werden noch verschiedene Programm-Ressourcen freigegeben und die Verbindung mit dem Display geschlossen.

Die hier dargestellten Möglichkeiten zur Interaktion mit dem Benutzer sind ein minimaler Ausschnitt aus dem technisch Möglichen. Zum Betrieb einer modernen Anwendung gehört Verhalten, welches vom Benutzer erwartet wir, wie Drag & Drop, scrollbare Inhalte, Veränderung des Cursors bei Änderung von dessen Funktion und Anderes.

5. Zusammenfassung

Ein Konzept für graphische Oberflächen mit Cybol unter X11 stellt die Implementierung der graphischen Funktionen mit Hilfe der Bibliothek Xlib dar. Dabei sollen die Graphikfunktionen keine Anwendungslogik enthalten, sondern ausschließlich ein im Speicher mit Variablen festgehaltenes Bild der Applikation auf den Monitor bringen. Die Trennung von Anwendungslogik und „Zeichen“-Logik birgt die Möglichkeit, andere graphische Ausgaben (wie webbasierte oder konsolenbasierte Ausgabe) zu implementieren, ohne dass die Anwendungslogik neu entworfen werden muss. Die vorgestellte Beispiel-Anwendung zeigt erste Ansätze zur praktischen Integration der Xlib-Funktionen in CYBOL. Bis eine graphische Funktionalität, die mit anderen modernen Anwendungen vergleichbar ist, in CYBOL verfügbar ist, wird noch viel Arbeit bei der Implementierung der GUI entstehen, deren Basis mit dieser Arbeit gelegt wurde.

6. Literaturverzeichnis

[cybop.net 2004] CYBOP - Cybernetics Oriented Programming
http://www.cybop.net

[tcl.tk 2004] Tcl/Tk Release Notes
http://www.tcl.tk/software/tcltk/relnotes/

[JONES 1991] Jones, Oliver
Einführung in das X Window System
1. Auflage Hanser-Verlag, Wien, München 1991

[KÖRBER u.a. 1998] Körber, Ulrich / Frank, Ulrich
OSF/Motif und das X-Window-System
1. Auflage, Hanser-Verlag, Wien, München 1998