KfK 4868
Mai 1991

Das XGAP-System

Ein Werkzeug zum Aufbau interaktiver graphischer Benutzerschnittstellen

H. Keller, E. Kugele, M. Gauges
Institut für Datenverarbeitung in der Technik
Projekt Schadstoffbeherrschung in der Umwelt

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe

Institut für Datenverarbeitung in der Technik
Projekt Schadstoffbeherrschung in der Umwelt

KfK 4868

Das *XGAP*-System

Ein Werkzeug zum Aufbau interaktiver graphischer Benutzerschnittstellen

H. Keller, E. Kugele, M. Gauges*

* FH Karlsruhe, FB I

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

Zusammenfassung:

Dieser Bericht beschreibt das System XGAP, ein Werkzeug zum Aufbau interaktiver graphischer Benutzerschnittstellen mit integriertem Editor. Der Editor dient zur Definition graphischer Objekte, die als Schablonen zur Erzeugung beliebig vieler Objekte innerhalb einer Anwendung dienen. Die Fensterverwaltung ermöglicht das Erzeugen, Manipulieren und Löschen von Fenstern während der Ausführung einer Anwendung. Die Kommunikation zwischen dem Anwendungsprogramm und den Fenstern erfolgt über synchrone oder asynchrone Operationen. Das XGAP-System ist mit C und X Windows realisiert.

The XGAP-System - A Tool for the Creation of Interactive Graphical User Interfaces

Abstract:

This report describes the system XAP, a tool for the design of interactive graphical user interfaces with an integrated editor. The editor is used for the definition of graphical objects, used as models for the instantiation of an infinite number of objects within an application. The window management makes it possible to create, manipulate and destroy windows during a session. The communication between the application program and the windows is processed by synchronous or asynchronous operations. The XGAP-system is implemented with C and X Windows.

Inhalt

1 Vorwort

Der vorliegende Bericht ist folgendermaßen gegliedert:

- Kapitel 2: Aufgabenstellung
  Hier werden die Anforderungen und Zielsetzungen an das Fensterverwaltungssystem, an die Ein- / Ausgabeoperationen und an den graphischen Editor erläutert.
- Kapitel 3: Einführung in das X Window System
  Dieses Kapitel beschreibt die Grundkonzepte des X Window Systems.
- Kapitel 4: Architektur Gesamtsystem
  In diesem Kapitel werden alle Module des XGAP-Systems kurz erläutert.
- Kapitel 5: Fensterverwaltung und Event-Abfrage
  Hier wird die asynchrone Eventabfrage erklärt und beschrieben, wie Fenster erzeugt und verwaltet werden.
- Kapitel 6: Fensterarten mit Widget-Hierarchien
  Beschreibung der verschiedenen Fenstertypen mit ihren Datenstrukturen.
- Kapitel 7: Fensterfunktionen
  Beschreibung der Anwendungsfunktionen zur Belegung der Fenster und Felder.
- Kapitel 8: Der graphische Editor
  Dieses Kapitel erläutert die Funktionen des Editors.
- Kapitel 9: Literaturverzeichnis
- Kapitel 10: Anhang A: Benutzungsanleitung Editor
  In diesem Anhang wird das Erstellen von graphischen Objekten mit dem Editor erläutert.
- Kapitel 11: Anhang B: Anwendungsbeispiel
  Zwei Objekte werden mit dem Editor erstellt, mit Anwendungsfunktionen ausgegeben und verbunden. Die dazu notwendigen Funktionsaufrufe sind in diesem Kapitel beschrieben.

Eine Beschreibung aller Anwendungsfunktionen mit ihren Ein- und Ausgabeparametern kann aus [Keller 1991] entnommen werden.

Die verwendeten Schriften und Schriftarten haben folgende Bedeutung:

- TIMES ist die Textschrift
- *kursiv* werden englische Ausdrücke, die in ihrer Originalform verwendet werden, beim ersten Auftreten gesetzt (und dort möglichst auch erklärt). Im weiteren Text sind diese Wörter nicht mehr hervorgehoben.
- `COURIER` wird für alle Symbolnamen (Funktionen, Variablen, Konstanten) verwendet.

Die verwendeten Symbole haben folgende Bedeutung:

- [..] enthalten Literaturverweise,
- Verweise auf Funktionen sind mit dem am linken Rand dargestellten Symbol gekennzeichnet. Die Ziffern innerhalb dieses Symbols geben die Funktionsnummer an (die Ziffern vor dem Punkt die Modulnummer, die Ziffern nach dem Punkt die Funktionsnummer innerhalb des Moduls).

2 Aufgabenstellung

Im Rahmen der Entwicklung des K_advice-Systems (siehe [Keller 1988]) wurde das GAP-System als interaktive graphische Benutzerschnittstelle entworfen und mit Ada und GKS Level 2c implementiert. Dabei ergaben sich durch GKS bestimmte Einschränkungen bzgl. der Interaktion mit dem Benutzer.
Die Arbeiten im Projekt INPRO für die wissensbasierten Komponenten (Sprache 1) erfordern ebenfalls eine interaktive graphische Oberfläche, allerdings mit Multi-Window-Technik und einer asynchronen Kommunikation mit diesen Fenstern.

Das Ziel der Arbeit war es deshalb, ein Programmsystem zum Editieren graphischer Objekte und zur Verwaltung beliebig vieler und verschiedenartiger Fenster in der Programmiersprache C unter Verwendung der X Windows Bibliotheken zu entwickeln. Die Programme sollten für VAX-Computer mit dem Betriebssystem VMS der Firma Digital Equipment Corporation (DEC) lauffähig sein, wobei auch ein Einsatz unter Unix (allgemein auf X Window-Basis) ermöglicht werden sollte. X Windows wurde verwendet, da es sowohl eine Bibliothek mit einfachen 2D-Zeichenfunktionen, als auch komplexe Funktionen zur Erzeugung und Gestaltung einer Benutzeroberfläche enthält.

Ausgehend vom GAP-System wurde das ***XGAP*** -System als ein eigenständiges System zur Gestaltung von interaktiven graphischen Benutzerschnittstellen mit einer Fensterverwaltung entworfen. Die Kommunikation mit den Fenstern kann synchron oder auch asynchron erfolgen. Zur Definition von graphischen Objekten wird dem Benutzer ein Editor zur Verfügung gestellt, der die Definition beliebiger farbgraphischer Elemente und Texte erlaubt. Diese können dann innerhalb der erzeugbaren Fenster weiter verarbeitet werden.
Im Anhang sind Beispiele zum Editor und zur Erzeugung und Verwaltung von Fenster zu finden.

Das Gesamtsystem besteht aus 2 wesentlichen Komponenten, die folgende Aufgaben erfüllen:

- XGAP-Fensterverwaltungsfunktionen (s. Kapitel 5 bis 7)
  Mit diesen fensterbezogenen Funktionen können Fenster verschiedener Typen erzeugt werden, die unabhängig voneinander manipuliert und auch gelöscht werden können.
  Ein Fenstertyp, das Standardfenster, enthält z.B. mehrere Felder zur Kommunikation mit dem Benutzer.

  In Arbeitsfenster sowie im Arbeitsfeld der Standardfenster können mit dem Editor erzeugte Objekte ausgegeben werden. Diese Objekte werden als Exemplare bezeichnet und können miteinander verbunden werden.

  Mit einer asynchronen und mehreren synchronen Funktionen können verschiedene Fenster-Events bzgl. aller oder einem spezifizierten Fenster abgefragt werden. Die Event-Klasse kann dabei vorgegeben sein.

- XGAP-Editor (s. Kapitel 8)
  Mit diesem Editor ist es möglich, graphische Grundelemente zu erzeugen, die auch zu einem graphischen Objekt zusammengefaßt werden können. Grundelemente sind:

  - Strecke, Polyline,
  - Quadrat, Rechteck,
  - Kreis, Kreisbogen und
  - Text.

  Außerdem können sog. Eingangs-, Durchgangs- und Ausgangs-Objekte erzeugt werden, die sich aus mehreren Grundelementen zusammensetzen und zur Verbindung mehrerer graphischer Objekte dienen (s. Anhang B).
  Die Grundelemente werden interaktiv erstellt und Element-Attribute (Farben und Kantenarten) können nach der Erstellung geändert werden.

  Es wurde eine Segmentverwaltung entwickelt, um die Daten der Elemente zu speichern und sie, wenn der Benutzer ein Objekt pickt (Selektion per Maus), zu finden.

  Die im Editor definierten graphischen Objekte dienen als Schablone zur Erzeugung von Anwendungsobjekten innerhalb eines Anwendungsprogramms. Schablonen können während erzeugte Fenster existieren definiert werden, da der Editor innerhalb einer Anwendung aufgerufen werden kann (die Anwendung wird solange blockiert).

Die aktuelle Programmversion ist Version 1.2.

In der nächsten Version werden graphische Objekte auch ohne Vorlage vom Editor über direkt verfügbare, interne Funktionen erzeugbar sein (Diese Funktionalität ist jetzt noch direkt zu programmieren.). Damit können dann Objekte ohne Schablone während einer Anwendung beliebig erzeugt und für Kurvenausgaben oder sonstige beliebige graphische Ausgaben verwendet werden.

3 Einführung in das X Window System

3.1 Allgemeines

Die Entwicklung des *X Window Systems* (im weiteren als X, X11 oder X-System bezeichnet) begann 1984 am MIT (Massachusetts Institute of Technology) anläßlich des Projekts ATHENA. Die erste Version, die als Basis für kommerzielle Produkte diente, war Version 10, Release 4 (X10R4). Die momentan aktuelle Version ist X11R4, welche im Januar 1990 erschien. Die meisten Implementierungen basieren auf X11R2 (März 88) bzw. X11R3 (Februar 89). Mit X11R2 ging die Weiterentwicklung und der Vertrieb von X auf das X Windows Consortium über, dem alle großen Computerhersteller angehören. Da jeder Interessierte X zu einem sehr geringen Preis erhalten kann, hat sich das System sehr schnell verbreitet und ist inzwischen als ANSI-Standard vorgeschlagen (und bereits X/Open-Standard).

X wurde als ein verteiltes, netzwerktransparentes, geräteunabhängiges Mehrprozeß-, Fenster- und Grafiksystem entwickelt [Byte 1989].
Dies wird durch ein *Client-Server*-Modell realisiert. Anwendungsprogramme (die Clients) senden Anfragen (*requests*) an einen Server. Dieser verarbeitet die Anfragen und sendet Anworten (*replies*) oder Events (s. S. 6) zurück. Im Falle eines Fehlers werden Fehler-Meldungen zurückgesendet.
In X hat die Bezeichnung Server nicht die gleiche Bedeutung wie in anderen Bereichen der Informatik. Ein File-Server beispielsweise befindet sich auf einer leistungsfähigen Maschine und bedient mehrere Clients, die über ein lokales Netz mit ihm verbunden sein können. Im X-System läuft auf jeder Maschine, bei der Grafiken auf einem Bitmap-Bildschirm ausgegeben und Eingaben von einer Tastatur oder anderen Eingabegeräten erwartet werden, ein Server. Der Server besteht normalerweise aus zwei Teilen, einem Hardware-abhängigen und einem Hardware-unabhängigen. Ein X-Anwendungsprogramm (Client) kann auf der gleichen Maschine wie der Server ablaufen (dann geschieht der Datenaustausch über Interprozeßkommunikation) oder mit dem Server über ein Computernetz verbunden sein (s. Abb. 3-1).
Zur Kommunikation über ein Netz wird das X-Protokoll verwendet, das man als "high-level" Grafik-Beschreibungungssprache bezeichnen kann. Da das Protokoll nur Schnittstellen vereinbart, können für unterschiedlichste Betriebssysteme Server implementiert werden, die der Hardware optimal angepasst sind. Das X-Protokoll benutzt bestehende Protokolle wie TCP/IP oder DECNet.
Eine wichtige Aufgabe des Servers ist das Verwalten von Ressourcen. Eine davon ist die Prozessorzeit zum Zeichnen und zur Textausgabe, eine weitere die Verwaltung des Bildschirmes, der Tastatur und der Zeigegeräte (z.B. der Maus).

Ein Client, der *window manager*, hat besondere Aufgaben. Er teilt den Clients, die auf dem Bildschirm Fenster öffnen möchten, deren Position zu und speichert die Fensterdaten. Außerdem dient der Window-Manager zum Verschieben, Vergrößern und Verkleinern von Fenstern (und zur Verwandlung in Icons). Er bestimmt auch das Aussehen der Fenster, z.B. wie der Fensterrand 'dekoriert' wird. Im Gegensatz zu anderen graphischen Benutzeroberflächen wie GEM oder WINDOWS ist das Erscheinungsbild der X-Fenster nicht vom System vorgegeben, sondern kann je nach Window-Manager völlig verschieden sein. Die Fenster des Window-Managers von MOTIF (einem Toolkit, s. Kap. 3.3) sehen beispielsweise aus wie die Fenster des Presentation Managers von OS/2.

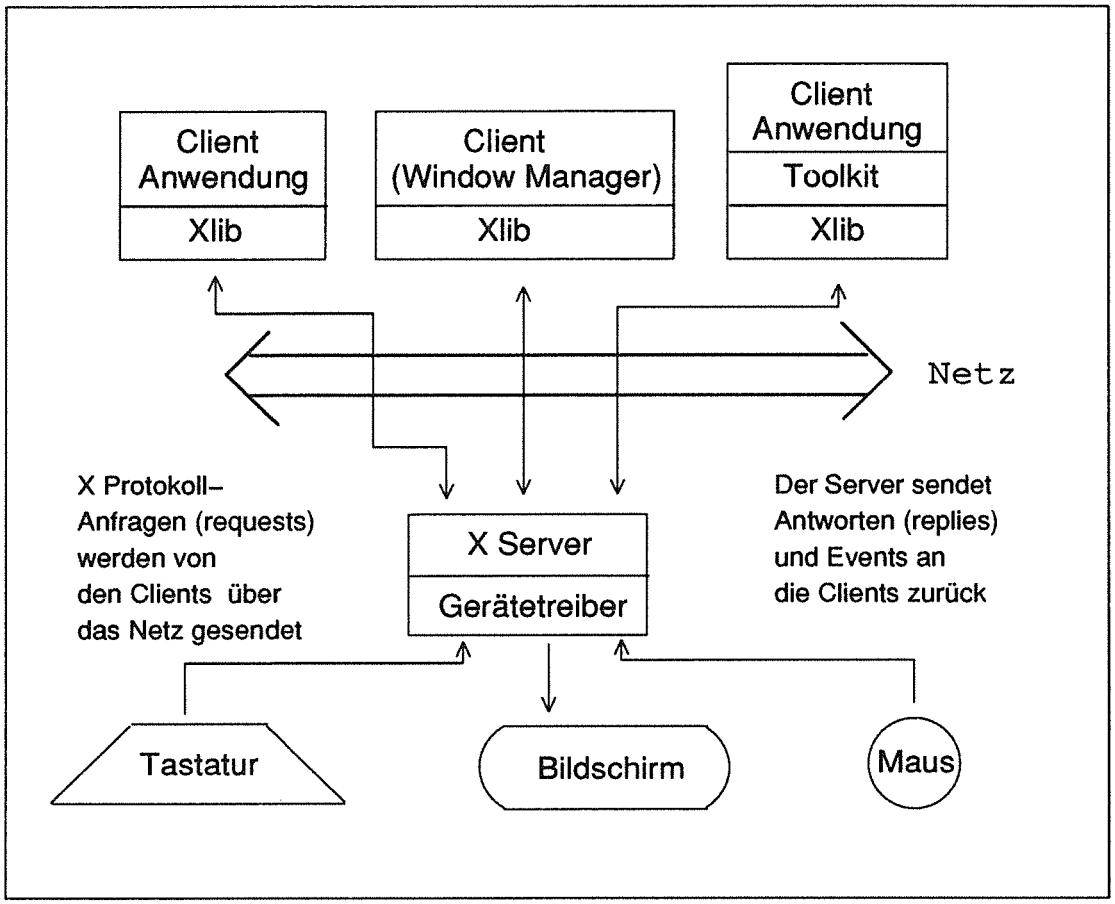


Abb. 3-1

Das Client-Server-Modell von X

(aus [Nye 1990-1], S. 9)

Das X-System ist schichtweise aufgebaut. Die unterste Ebene bilden die *Xlib*-Funktionen (deren Namen mit X beginnen). Diese Funktionen bilden eine Bibliothek mit:

- Grafikfunktionen (Zeichnen von Punkten, Linien, Kreisen und Rechtecken)
- Fensterfunktionen (Öffnen, Verschieben und Schließen von Fenstern)
- *Event*-Funktionen
- Funktionen zum Laden und Ausgeben von Schriftzeichen
- Funktionen zur Kontrolle von Tastatur und Zeigegeräten (z.B. Maus)

Ein Event ist ein Ereignis, das durch eine Benutzeraktion oder eine Systemaktion ausgelöst wird. Dazu gehören z.B. das Bewegen der Maus, das Drücken einer Maustaste oder der Tastatur. Auch das Verschieben von Fenstern kann Events auslösen, wenn dadurch ein anderes Fenster verdeckt wird. Das X-System ist event-orientiert. Im Anwendungsprogramm werden Events, die von X erzeugt wurden, abgefragt und es wird darauf reagiert.

Die nächste Schicht im X-System ist die *X-Toolkit*-Ebene (die Funktionsnamen beginnen mit Xt). Auch diese ist vom X-Konsortium standardisiert.
Das X-Toolkit enthält Funktionen, die mit Hilfe von Xlib programmiert wurden. Ein Xt-Datentyp wird als *widget* bezeichnet. Er enthält Informationen über die graphischen Objekte, die damit realisiert werden. Zu diesen gehören Buttons, Menüs, Dialogfenster u.v.m.

Neben dem standardisierten Xt-Toolkit gibt es weitere herstellerspezifische Toolkits, z.B. DEC-Windows, OPEN LOOK (implementiert von SUN, wird von AT&T zusammen mit UNIX, System V.4 vertrieben) und MOTIF (von HP/Microsoft und DEC, vertrieben von der Open Software Foundation).

3.2 Xlib

Xlib sendet nicht jede Anweisung sofort an den Server, sondern puffert diese, da nicht jeder Befehl sofort eine Antwort benötigt. Das Anwendungsprogramm kann gleich die nächsten Befehle abarbeiten. Damit wird das Netz weniger belastet, da immer eine Gruppe von Anfragen geschickt wird. Im Gegensatz dazu sendet der Server normalerweise die Antworten sofort an den Client zurück. Die Zwischenspeicherung der Events erfolgt in Warteschlangen (s. Abb. 3-2).

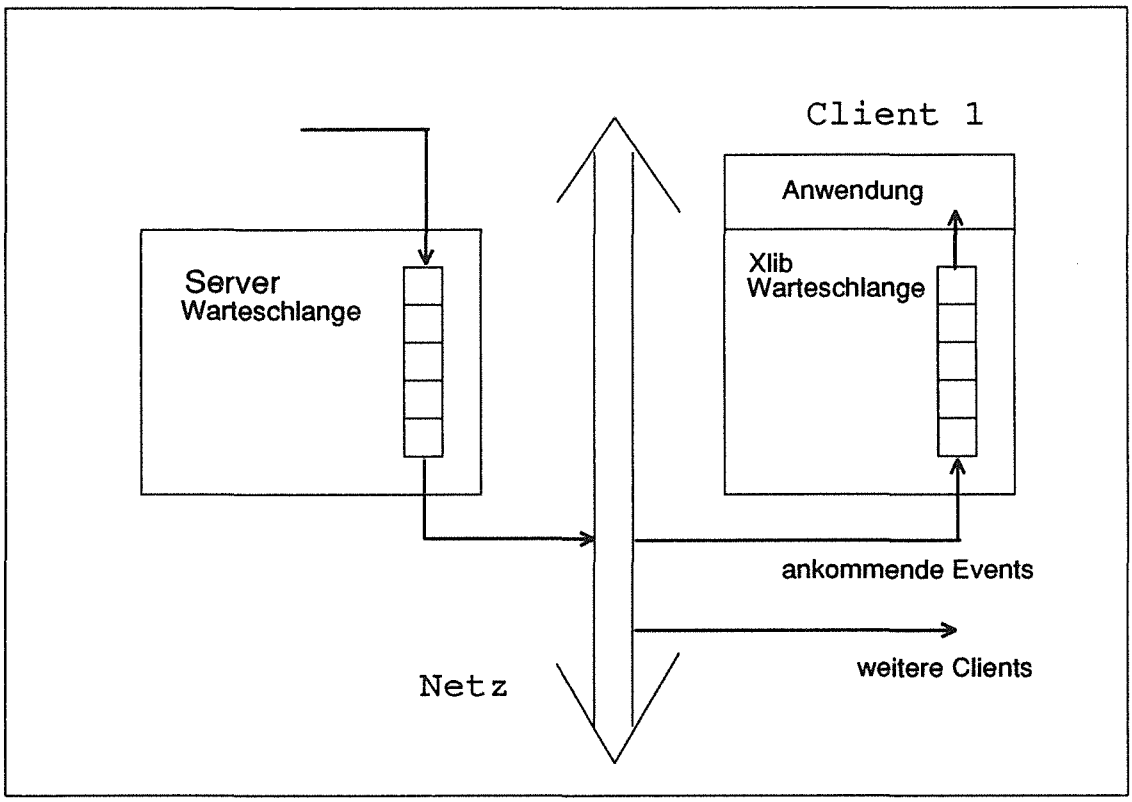


Abb. 3-2
Die Event-Warteschlangen von Client und Server
(aus [Nye 1990-1], S. 19)

Eine weitere Technik, um die Datenmenge im Netz zu reduzieren, ist die Verwendung von *Ressourcen*. Der Server speichert Daten für Fenster, *Pixmaps* (s. Kap. 3.2.2), Farbtabellen, Zeichensätze und *graphische Kontexte* (s. Kap. 3.2.3) in diesen Ressourcen, sie werden vom Client mit einem Integer-Wert identifiziert. Bei jedem Zugriff auf diese Daten wird in der Xlib-Routine dieser Wert verwendet. Damit muß nicht immer die gesamte Datenstruktur über das Netz gesendet werden.
Ein Fenster (*window*) in X wird auf einem Ausschnitt eines Bitmap-Bildschirmes dargestellt. Jedes Fenster hat ein Eltern-Fenster (*parent window*) und kann ein oder mehrere Unterfenster (*child window*), haben. Damit läßt sich eine Fensterhierarchie erzeugen. Das einzige Fenster ohne Eltern ist das Wurzel-Fenster (*root window*), das den gesamten Bildschirm ausfüllt. Alle weiteren Fenster "stammen von diesem ab". Child Windows können nur innerhalb ihrer Eltern liegen. Die Position und Größe des Fen-

sters wird mit dem linken, oberen Punkt der Fensterkante, der Breite und der Höhe bestimmt. Das X-Koordinatensystem hat seinen Ursprung in der linken, oberen Ecke des Bildschirms. Die Koordinatenangaben von child windows sind relativ zu ihren Eltern. Die Child Windows eines Fensters haben eine bestimmte Ordnung (welches liegt auf welchem), dies wird als *stacking order* bezeichnet.
Wenn ein Fenster erzeugt wird, ist es nicht sofort sichtbar. Um darin etwas ausgeben zu können, muß man es sichtbar machen, was als *mapping* bezeichnet wird. Außerdem müssen alle Vorgänger (*ancestors*) sichtbar sein.

3.2.1 Pixel, Farben und Planes

Für jeden darstellbaren Punkt auf dem Bildschirm (*pixel*) wird ein Farbwert (integer) gespeichert. Bei einem Schwarz-Weiß-Bildschirm reicht dazu ein Bit, bei einem Farbbildschirm werden mehrere Bits benötigt. Jedes Bit gehört zu einer Ebene (*plane*), die Anzahl der Bits gibt die Anzahl der Ebenen an. Diese Zahl wird aber nicht direkt als Farbe interpretiert, sondern gibt die Nummer eines Tabelleneintrags an. In dieser Farbtabelle (*colormap*) sind die Farbwerte, getrennt für Rot, Grün und Blau gespeichert. Eine Zeile dieser Tabelle wird als *colorcell* bezeichnet. Der Wert einer Zeile gibt die Farbe eines Bildschirmpunktes an (s. Abb 3-3).

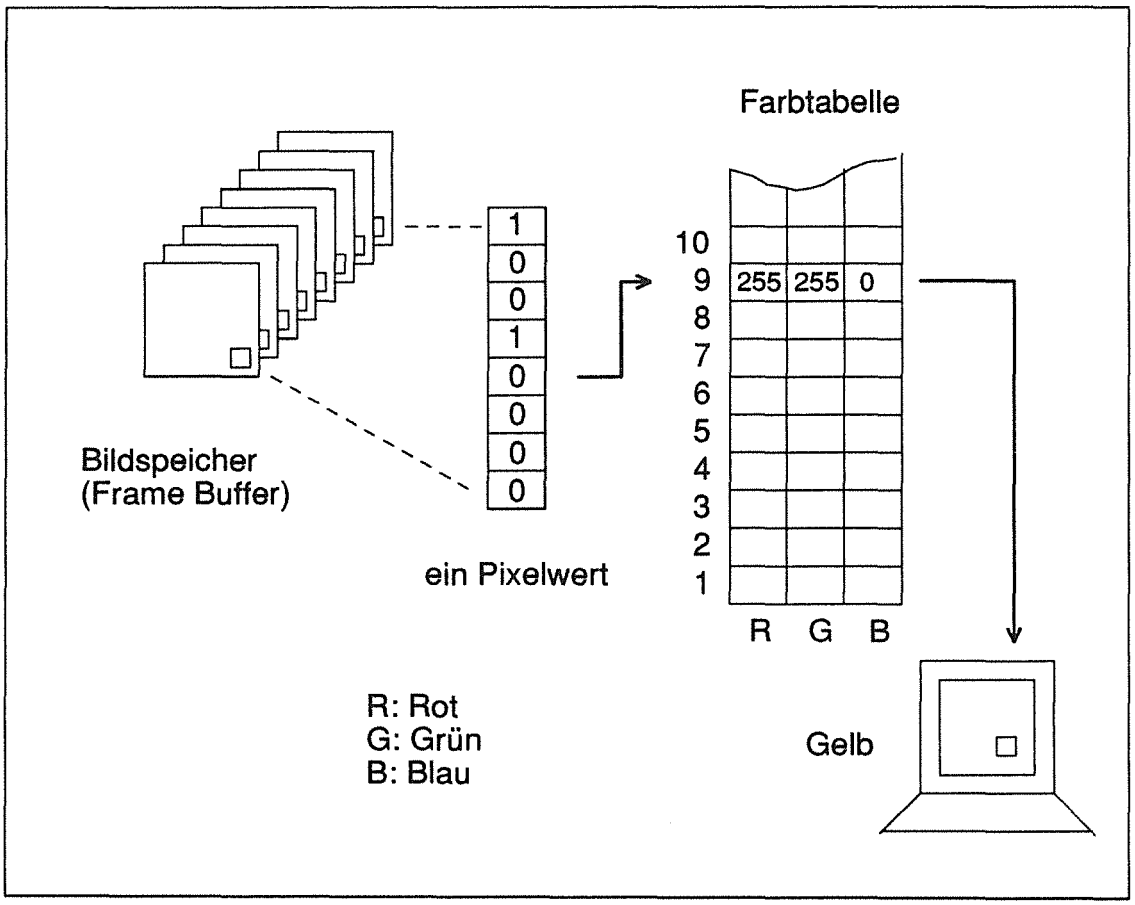


Abb. 3-3
Umwandlung eines Pixelwertes in einen Farbwert mit einer Farbtabelle
(aus [Nye 1990-1], S. 31)

Je mehr Planes ein Bildschirmspeicher hat, desto mehr Farben (= Tabelleneinträge) können gleichzeitig angesprochen und ausgegeben werden. Möchte man mehr Farben ausgeben, als Einträge in der Farbtabelle sind, muß man eine neue Farbtabelle laden.

Bei jeder Grafikoperation wird vor der Ausgabe eines Pixel eine logische Operation durchgeführt, die den bereits bestehenden Pixel-Wert in einem Ausgabefenster (*destination*) mit dem neuen Pixel-Wert (*source*) verknüpft. Das Ergebnis dieser Operation wird als neuer Indexwert für die Farbtabelle verwendet.

3.2.2 Pixmaps und Drawables

Grafiken lassen sich auch in *pixmaps* speichern und später in einem Fenster ausgeben. Eine Pixmap ist ein Teil vom Arbeitsspeicher des Servers und hat, wie ein Fenster, eine Anzahl Planes. Die Pixmap hat keine Fensterattribute wie z.B. eine Position relativ zu anderen Fenstern oder eine Farbtabelle. Fenster und Pixmaps werden als *drawables* bezeichnet.
Eine Pixmap mit der Tiefe 1 heißt *bitmap*.

3.2.3 Der graphische Kontext

Um ein graphisches Element (z.B. eine Linie) zu zeichnen, werden neben Koordinaten- und Zielangaben (Window oder Pixmap) auch weitere Daten benötigt. Dazu gehört die Breite der Linien, die Farben und eventuelle Füllmuster. Diese zusätzlichen Daten werden als Ressource im Server gespeichert und als *graphics context* (GC) bezeichnet. Jeder GC erhält eine identifizierende Nummer, die dann als ein Parameter der Grafik-Funktion benutzt wird.
Die GC's müssen vor der Grafikausgabe erzeugt werden; verwendet man mehrere, können mit der gleichen Anweisung Grafiken mit verschiedenem Aussehen (z.B. punktierte oder durchgezogene Linien) erzeugt werden.

3.2.4 Events

Der Server sendet Events nicht nur beim Bewegen der Maus oder Drücken einer Taste, sondern auch, wenn ein Fenster sichtbar wird oder der Mauszeiger den Rand eines Fensters überquert. Außerdem werden Events gesendet, falls ein Fenster ein anderes überdeckt (*expose*-Event) oder die Größe eines Fensters geändert wird (*gravity notify*). Weitere Events dienen der Kommunikation zwischen Clients und dem Window-Manager. Wird z.B. die Eingabemöglichkeit (*keyboard focus*) von einem Fenster zu einem anderen übertragen, werden `FocusIn` und `FocusOut`- Events gesendet.

Mit der Funktion `XSelectInput` kann angegeben werden, welche Events an den Client gesendet werden. Damit können, falls man nur bestimmte Events wünscht, diese herausgefiltert werden. Jeder Client hat eine eigene Event-Queue, in der vom Server gesendete Events zwischengespeichert werden (s. Abb. 3-2).

In der Event-Datenstruktur wird neben dem Eventtyp und einer laufenden Nummer auch die Identifikationsnummer des Fensters und die Position des Events innerhalb dieses Fensters vom Server gesendet.

Mit einer (normalerweise) unendlichen Schleife im Anwendungsprogramm werden die Events abgefragt und verarbeitet.

3.3 Xt

Das X-Toolkit, das auf den Xlib-Funktionen aufbaut, bietet mit den Widgets die Möglichkeit, mit einer nicht objektorientierten Sprache objektorientiert zu arbeiten. Ein Widget ist ein wiederverwendbares, anpaßbares Stück Code, das unabhängig vom Anwendungsprogramm ist (aus [Nye 1990-3], S. 8).
Ein Widget oder ein *widget set* (Menge von Widgets) bietet z.B.:

- Felder zur Dateneingabe,
- Dialogboxen mit verschiedenen Buttons,
- einen *scrollbar* (Verschiebebalken), um Text oder Grafik, die größer als das sichtbare Fenster ist, zu verschieben und damit sichtbar zu machen.

Widgets, die aus mehreren Grund-Widgets zusammengebaut sind, werden als *composite widget* bezeichet.

Eine Widget-Bibliothek definiert Klassen von Widgets, die bestimmte Eigenschaften haben. Ein Widget ist ein Exemplar (*instance*) einer Klasse. Alle Exemplare einer Klasse haben zunächst die gleichen vordefinierten Eigenschaften (wie z.B. ein Textfeld), die vom Anwendungsprogramm für jedes Exemplar angepaßt werden können. Man kann (wie bei Xlib mit Fenstern) eine Klassenhierarchie aufbauen. Unterklassen erben (*inherit*) Charakteristika von ihren Oberklassen und können zusätzliche Eigenschaften bzw. Funktionen haben.
Ein Beispiel für eine Klassenhierarchie ist anhand der DEC-Widgets in Abb. 3-4 dargestellt:

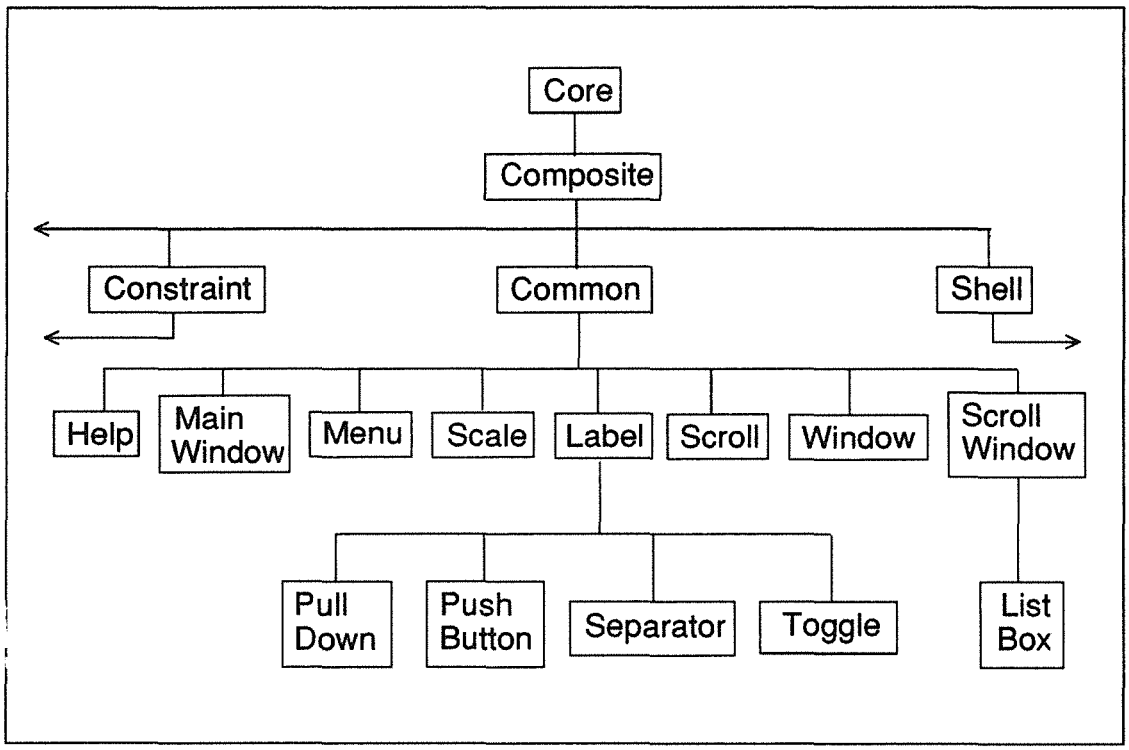


Abb. 3-4
Teil der DEC-Widget-Hierarchie
(aus [XUI 1988], Vol. 2, S. 1-4)

Um Eigenschaften eines Exemplares zu ändern, gibt es die Funktionen

- `XtSetArg` und
- `XtSetValues`.

Mit `XtSetArg` wird die zu ändernde Eigenschaft und ihr neuer Wert angegeben, z.B.:

```
XtSetArg (arg, XtNlabel, "Hallo");
```

Hier wird der Eigenschaft `XtNlabel` die Zeichenkette `"Hallo"` zugewiesen. In `XtSetValues` wird angegeben, welches Widget geändert werden soll; z.B. ändert

```
XtSetValues (label, arg, 1);
```

die mit `arg` übergebenen Argumente im Widget `label`, die Zahl 1 gibt die Anzahl der zu ändernden Eigenschaften an.
Diese Art der Programmierung wird in objektorientierten Sprachen als Versenden von Nachrichten (*messages*) bezeichnet. Die angesprochenen Funktionen sind die Methoden (*methods*) der Widgets. Da der Zugang zu Widgets nur über wenige Funktionen möglich ist, werden sie als eingekapselte (*encapsulated*) Objekte bezeichnet. Dazu gehört auch die Unabhängigkeit der Widgets vom Anwendungsprogramm. Widgets werden, im Gegensatz zu Fensterinhalten, wieder automatisch neu ausgegeben, wenn sie verdeckt waren.

Mit *callback* - Funktionen kann man, von Widgets aus, vorher definierte Funktionen aufrufen. Dies kann z.B. eine Funktion sein, die beim Drücken eines PushButton-Widgets ausgeführt werden soll. Auch die Callback-Funktionen werden mit den oben beschriebenen Funktionen an das Widget übergeben. Eine weitere Möglichkeit, Callback-Funktionen aufzurufen, sind *action* - Routinen und *translation* - Tabellen. Ein Beispiel dafür ist die Callback-Funktion, die beim Editor mit dem Ausgabefenster verbunden ist. Damit kann man, wenn der Benutzer innerhalb des Ausgabefensters eine Taste drückt, eine bestimmte Funktion aufrufen, die die Maus-Zeiger-Koordinaten ermittelt. Diese Koordinaten werden an eine Zeichenfunktion, z.B. zum Zeichnen einer Strecke, weitergegeben.

3.4 DEC-Windows und weitere Toolkits

Das Toolkit-Set, das mit den X-Bändern ausgeliefert wird, ist das ATHENA-Widget-Set. Es bietet einige Grundelemente zur Programmierung einer Benutzeroberfläche an, eignet sich aber nicht zur kommerziellen Verwendung, da es zu wenige Klassen und Methoden hat.
Weitere Bibliotheken sind die erwähnten Toolkits MOTIF und OPEN LOOK, die gegeneinander konkurrieren. Die Firma DEC bietet zu ihrem DEC-Windows ein Widget-Set an; dieses wurde für das XGAP-System verwendet. Als weitere Programmiererleichterung wird ein USER INTERFACE LANGUAGE (UIL) COMPILER angeboten. Da diese Sprache DEC-spezifisch ist, wurde die UIL nicht benutzt. Die DEC-Widgets müßten sich relativ leicht in ein anderes Widget-Set portieren lassen.
Die oberste Klasse der DEC-Widgets ist `CORE`. Attribute dieser Klasse werden an alle in der Hierarchie folgenden Widgets vererbt.
Die Attribute der Klassen `CORE` und `COMPOSITE` sind die gleichen. Dazu gehören u.a

die Position des Widgets (relativ zu seinen Eltern), Breite, Höhe, Hintergrundfarbe und Zeiger auf Übersetzungstabellen.
In der Klasse COMMON kommt u.a. zusätzlich die Vordergrundfarbe und ein Zeichensatz dazu, außerdem ein Zeiger auf Hilfe-Funktionen.
Wenn man die in Abb. 3-4 dargestellte Hierarchie weiter verfolgt und beispielsweise die Klasse LABEL betrachtet, kommen hier als Attribute u.a. zusätzlich eine Zeichenkette und deren Position hinzu.
Geht man noch eine Stufe weiter zu PUSH BUTTON, enthält diese Klasse weitere Attribute wie z.B. Zeiger auf Callback-Funktionen, die bei Anklicken des Buttons aufgerufen werden sollen.

3.5 Verbindung Xlib und Xt

Verwendet man in einem Anwendungsprogramm Xlib und Xt, muß man die Widgets der Xt-Ebene mit den Fenstern der Xlib-Ebene verbinden. Dazu verwendet man folgende Funktionen:

- `display XtDisplay (widget)`
- `screen XtScreen (widget)`
- `window XtWindow (widget)`

Ein Display in X ist eine Workstation mit Tastatur, Zeigeeinrichtung (z.B. Maus) und einem oder mehreren Bildschirmen (*screens*). Man kann bei X mehrere physikalische Bildschirme gleichzeitig verwenden und den Mauszeiger von einem in den anderen bewegen (CAD-Anwendungen).
In Ausgabefunktionen der Xlib, z.B. dem Zeichnen eines Rechtecks, gibt man das Display und das Ausgabefenster an. Beim Zuweisen von Farben aus Farbtabellen wird der Screen benutzt.

4 Architektur Gesamtsystem

Das XGAP-Programm besteht aus 18 getrennt übersetzbaren C-Modulen mit 15 zugehörigen Deklarationsdateien (*.h), in denen die Datenstrukturen deklariert sind und zwei Dateien für Konstantendefinitionen. Außerdem gibt es 36 Bitmap-Dateien (*.bit) für den Editor. Diese werden für die Buttons des Editors und für ein Label in der E/A-Dialogbox benötigt.
In Abb. 4-1 ist die Ebenen-Hierarchie aller Module dargestellt. Die Modulnamen in Rechtecken sind Module ohne Variablenspeicher (ADT, abstrakte Datentypen), die Modulnamen innerhalb Ellipsen sind Module mit Variablenspeicher (ADO, abstrakte Datenobjekte). Alle Funktionen eines Moduls benutzen nur Funktionen von Modulen unterhalb der Ebene dieses Moduls oder Modul-Funktionen gleicher Ebene.
Mehrere Module werden sowohl für den XGAP-Editor, als auch für die Anwendungsfunktionen benötigt (z.B. AUSGABE, DATEI), andere nur für den jeweiligen XGAP-Teil (z.B. XGAP_FENSTER und EDIT_OBJ).

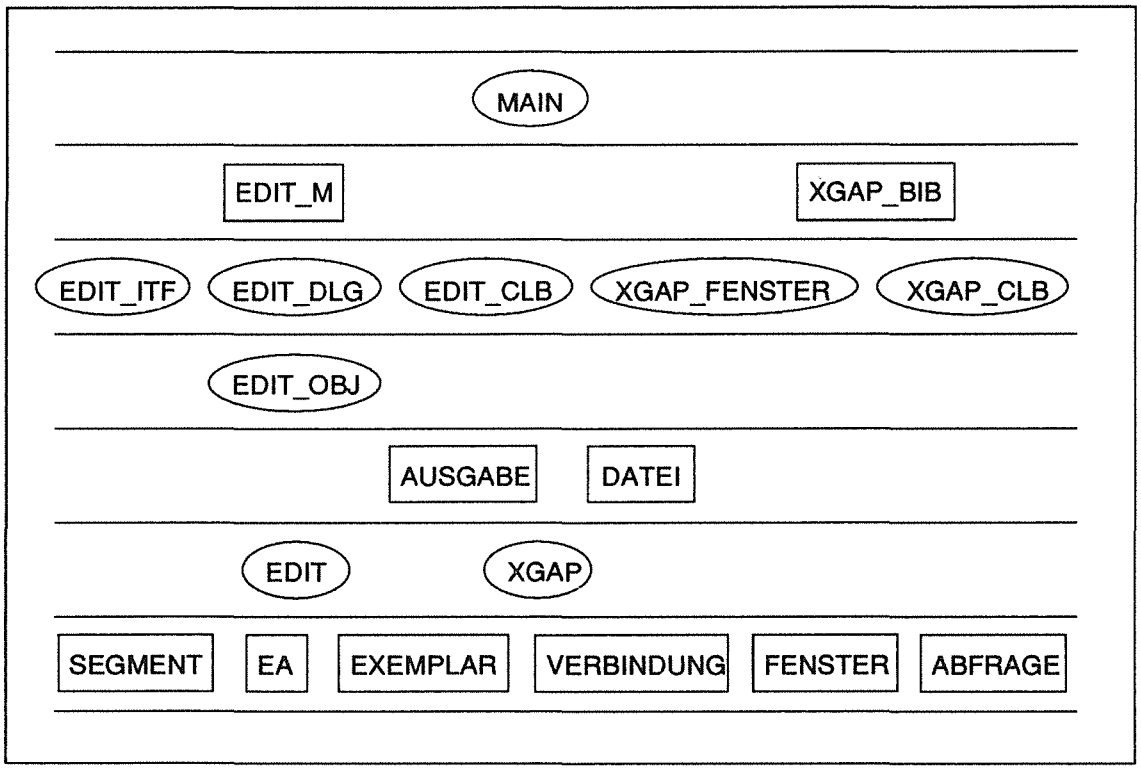


Abb. 4-1
Modulstruktur XGAP_System

Auf alle Datenstrukturen, die in den Deklarationsdateien vereinbart wurden, wird nur indirekt über Funktionen zugegriffen. Für jede Komponente der Struktur gibt es eine Schreib- und eine Lesefunktion. Ein Parameter solch einer Funktion ist ein Zeiger auf die aktuelle Adresse der Datenstruktur.

4.1 Module für den Datenzugriff

Folgende Module enthalten Funktionen für das Schreiben und Lesen der elementaren Datenstrukturen:

- XGAP

- FENSTER
- EDIT
- SEGMENT
- EA
- EXEMPLAR
- VERBINDUNG
- ABFRAGE

4.1.1 Modul XGAP

Das Modul XGAP enthält Variablen und Zugriffsfunktionen, die sowohl vom Editor als auch von den Anwendungsfunktionen benötigt werden. Dazu gehört eine Variable `xgap_speicher` vom Typ `xgap_speicher_typ` (s. Abb. 4-2) zur Parameterübergabe zwischen Funktionen. Der Zugriff auf diesen Speicher erfolgt nicht direkt, sondern über vier Funktionen. Vorbild für die Funktionen war die Methode, wie den Widgets Werte übergeben werden (`XtSetArg`).

Abb. 4-2
Struktur xgap_speicher

```
typedef struct {
            int     ergebnis;
            int     wert;
            int     feld;
            int     x, y;
            int     typ;
            int     unter_typ;
            int     farbe;
            int     kantenbreite;
            int     strichart;
            int     positions_art;
            int     loesch_art;
            int     aktion_aktiv;
            int     aktion_bleibt;
            char    text[max_laenge_eingabe+1];
            char    menu_text[max_laenge_text+1];
            char    name_exemplar[max_laenge_text+1];
            char    name_ea[max_laenge_text+1];
            char    text_ea[max_laenge_text+1];
            char    name_verbindung[max_laenge_text+1];
            char    von_exemplar[max_laenge_text+1];
            char    nach_exemplar[max_laenge_text+1];
            char    von_ausgang[max_laenge_text+1];
            char    nach_eingang[max_laenge_text+1];
            char    segment_text[max_laenge_text+1];
            } xgap_speicher_typ;

xgap_speicher_typ    xgap_speicher;
```

Die ersten zwei Funktionen dienen zum Schreiben, der erste Parameter dieser Funktionen (eine der in `XGAP_KONST.h` vereinbarten Konstanten `xgap_...`, s. Abb. 4-3)

selektiert eine Komponente der Struktur, der zweite enthält die Daten (einen Integerwert oder einen Zeiger auf eine Zeichenkette):

- `setze_xgap_wert` speichert einen Integerwert;
- `setze_xgap_text` speichert eine Zeichenkette.

Die nächsten beiden Funktionen lesen Daten; der einzige Parameter selektiert die Komponente, die mit einer der `setze`-Funktionen geschrieben wurde (s. Abb. 4-3):

- `lese_xgap_wert` gibt einen Integerwert zurück,
- `lese_xgap_text` gibt einen Text zurück.

Die einzelnen Komponenten von `xgap_speicher_typ` werden den Konstanten von Abb. 4-3 zugeordnet (dadurch, daß man bei den Konstanten `xgap_` wegläßt). Die Konstanten von Nummer 1 bis 15 selektieren Integerwerte, die weiteren (ab Nr. 20) Text.

```
#define     xgap_x                  1
#define     xgap_y                  2
#define     xgap_feld               3
#define     xgap_typ                4
#define     xgap_unter_typ          5
#define     xgap_positions_art      6
#define     xgap_loesch_art         7
#define     xgap_farbe              8
#define     xgap_kantenbreite       9
#define     xgap_strichart         10
#define     xgap_aktions_nr        11
#define     xgap_ergebnis          12
#define     xgap_aktion_aktiv      13
#define     xgap_aktion_bleibt     14
#define     xgap_icon_start        15

#define     xgap_text              20
#define     xgap_name_exemplar     21
#define     xgap_name_ea           22
#define     xgap_name_verbindung   23
#define     xgap_von_exemplar      24
#define     xgap_nach_exemplar     25
#define     xgap_von_ausgang       26
#define     xgap_nach_eingang      27
#define     xgap_aktions_name      28
#define     xgap_menu_text         29
#define     xgap_eingabe_text      30
#define     xgap_popup_text        31
#define     xgap_segment_text      32
#define     xgap_text_ea           33
```

Abb. 4-3
Konstanten für xgap_speicher

Während der Initialisierung des XGAP-Systems (Funktion `init_xgap`) werden in diesem Modul auch die aktuellen Display- und Screen-Adressen gespeichert. Dies vermeidet unnötige Client-Server-Kommunikation, da die Adressen ständig auf Client-Seite verfügbar sind.
Um bei der Ausgabe von graphischen Primitiven nicht jedesmal neue graphische Kontexte und Fonts anfordern zu müssen, werden auch diese in der Initialisierungsphase

erzeugt und gespeichert. Mit

```
ermittle_copy_gc ()
```

kann dann z.B. auf den graphischen Kontext `copy_gc` zugegriffen werden.
Außerdem enthält dieses Modul weitere Hilfsfunktionen für den Editor und die Anwendung.

4.1.2 Modul FENSTER

Dieses Modul enthält Zugriffsfunktionen zur Fensterverwaltung. Dazu gehören die Widgets der verschiedenen Fenstertypen, Komponenten zum Speichern von Events und Zeiger auf Exemplar- und Verbindungslisten. Die Datenstrukturen werden in Kapitel 5 und 6 erläutert.

4.1.3 Modul EDIT

Das Modul `EDIT` enthält eine Variable vom Typ `editor_typ` (s. Kap. 8.5.1) zur Speicherung des momentanen Editorzustandes (z.B. gewählter Modus, Farbe etc.) und eine Variable vom Typ `objekt_typ` (s. Kap. 8.5.2) mit Daten eines graphischen Objekts.

4.1.4 Module SEGMENT, EA und EXEMPLAR

Im XGAP-Editor können graphische Objekte erzeugt und manipuliert werden, die aus graphischen Primitiven (Strecke, Kreis etc.), sog. Segmenten und aus zusammengesetzten Objekten, sog. Eingangs- bzw. Ausgangsobjekten (EA) bestehen.
Im Editor kann jedes einzelne Segment und E/A-Objekt manipuliert werden. Diese Grundelemente werden mit dem Editor als ein größeres Objekt gespeichert.
Mit den XGAP-Anwendungsfunktionen werden diese Objekte als Ganzes manipuliert und als Exemplare bezeichnet.
Ein mit dem Editor erzeugtes Objekt kann als Vorlage für beliebig viele Exemplare dienen.
Die drei Module `SEGMENT`, `EA` und `EXEMPLAR` enthalten Zugriffsfunktionen für Variablen von den Typen `segment_typ`, `ea_typ` und `exemplar_typ`. Mehr dazu ab Kap. 6.1.1.

4.1.5 Modul VERBINDUNG

Die in den Ausgabefenstern dargestellten Exemplare können miteinander verbunden werden. Quell- und Zielobjekte sind Eingangs- bzw. Ausgangsobjekte der Exemplare. Eine Verbindung besteht aus einer Polyline mit drei Teilstrecken. Im Modul `VERBINDUNG` wird auf die Komponenten der Struktur `verbindungs_typ` zugegriffen. Die einzelnen Komponenten werden in Kap. 6.1.4 erklärt.

4.1.6 Modul ABFRAGE

Diese Funktionen werden nur für die Funktion `abfrage_xgap_event` (Modul `XGAP_FENSTER`) verwendet. Sie dienen zur Parameterübergabe (Ein- und Ausgabe) von einer Benutzerfunktion an `abfrage_xgap_event` und der Rückgabe von Wer-

ten an die aufrufende Funktion. Die einzelnen Parameter-Funktionen sind in [Keller 1991] beschrieben.

4.2 Grundfunktionen zur Objektausgabe und -speicherung

Auch die Module

- `AUSGABE` und
- `DATEI`

werden, wie die Module `XGAP`, `SEGMENT` und `EA`, sowohl innerhalb Editor-Funktionen als auch in Anwendungsfunktionen benutzt.

4.2.1 Modul AUSGABE

Dieses Modul enthält Funktionen zur Ausgabe eines graphischen Objekts mit Segmenten und E/A-Objekten. Als Parameter wird u.a der Beginn der Segment- und E/A-Listen übergeben und die Adresse des Ausgabefensters.

4.2.2 Modul DATEI

Die Funktion `objekt_speichern` speichert ein graphisches Objekt (= Segmente u. E/A-Objekte) in einer Datei, `objekt_laden` lädt ein Objekt aus einer Datei.

4.3 Module des Anwendungsteils

Der Anwendungsteil des XGAP-Programms besteht aus drei Modulen:

- `XGAP_FENSTER`
- `XGAP_BIB` und
- `XGAP_CLB`

4.3.1 Modul XGAP_FENSTER

Dieses Modul enthält Funktionen zur Erzeugung und zum Löschen der Fenster-Widgets. Außerdem wird mit der Variablen `erstes_fenster` (Typ `fenster_typ_zeiger`) die Fensterliste verwaltet. Auch die Funktion `abfrage_xgap_event` zur asynchronen Abfrage von Events befindet sich in diesem Modul.
Die Funktionen werden in Kap. 5. erläutert.

4.3.2 Modul XGAP_BIB

Es enthält Anwendungsfunktionen zur

- Belegung von Fenster und Felder,
- Erzeugung von Exemplaren und Verbindungen
- und zur Manipulation dieser graphischen Objekte.

Mehr dazu in Kap. 7.

4.3.3 Modul XGAP_CLB

In diesem Modul befinden sich die Callback-Funktionen der Fenster-Widgets. Sie werden beim Anklicken der Aktions- und Menüfelder, der Arbeitsflächen und bei Eingaben aufgerufen.

4.4 Module des Editors

Fünf Module enthalten Funktionen des Editors:

- `EDIT_M`
- `EDIT_ITF`
- `EDIT_DLG`
- `EDIT_CLB` und
- `EDIT_OBJ`

Die Editor-Funktionen werden in Kap. 8 ausführlich erklärt.

4.4.1 Modul EDIT_M

Dieses Modul enthält die Funktion `starte_xgap_editor`, die jederzeit nach `init_xgap` aufgerufen werden kann. Beim ersten Aufruf werden die Widgets des Editors erzeugt, bei allen weiteren Aufrufen werden die Widgets nur noch angezeigt (= *manage*).

4.4.2 Module EDIT_ITF und EDIT_DLG

Mit diesen Modulen wird die Benutzeroberfläche und die Dialogboxen des Editors erzeugt:

- `EDIT_ITF`
  zur Erzeugung des Benutzer-Interfaces (`ITF`) mit der Widgethierarchie,.
- `EDIT_DLG`
  für die Dialogboxen (`DLG`) zur Abfrage eines Dateinamens, Erzeugung von E/A-Objekten u.ä.

4.4.3 Modul EDIT_CLB

Es enthält die Callback-Funktionen, die z.B. beim Drücken eines Auswahl-Buttons oder Aktivieren eines Pulldown-Menüs aufgerufen werden.

4.4.4 Modul EDIT_OBJ

Hiermit werden die Segment- und die E/A-Liste eines graphischen Objekts verwaltet. Außerdem befinden sich in diesem Modul die Funktionen zur Erzeugung, Manipulation, Ausgabe und Speicherung der Editor-Objekte.

5 Fensterverwaltung und Event-Abfrage

Zur Ein- und Ausgabe stehen fünf verschiedene Fenstertypen zur Verfügung:

- Standardfenster
- Arbeitsfenster
- Menüfenster
- Eingabefenster
- Textfenster

Die Verwaltung dieser Fenster erfolgt mit einer einfach verketteten Liste vom Typ `fenster_typ_zeiger` (s. Abb. 5-1). Es können beliebig viele Fenster erzeugt und wieder gelöscht werden.
Um mit einer Datenstruktur verschiedene Widget-Hierarchien verwalten zu können (sie sind unterschiedlich je nach Fenstertyp) wird eine variante Struktur (`union`) benutzt.

```
typedef struct {
            char          *name;
            int           typ;
            Widget        toplevel;
            Widget        mainwindow;
            Widget        haupt_fenster;
            union {
                    standard_fenster_typ standard;
                    eingabe_fenster_typ  eingabe;
                    text_fenster_typ     text;
                    menu_fenster_typ     menu;
                    arbeits_fenster_typ  arbeit;
                } fenster;
            int           event_klasse;
            Bool          event_eingetroffen;
            char          text_1[max_laenge_text + 1];
            char          text_2[max_laenge_text + 2];
            XPoint        position;
            int           wert;
    struct fenster_typ    *naechster;
            } *fenster_typ_zeiger, fenster_typ;

fenster_typ_zeiger        erstes_fenster;
```

Abb. 5-1
Struktur fenster_typ

Die einzelnen Komponenten der Fenster-Struktur haben folgenden Zweck:

- `name` speichert den Namen des Fensters. Jeder Name kommt in der Liste nur einmal vor und dient zur eindeutigen Identifizierung.
- In `typ` wird der Fenstertyp gespeichert, der Komponenten wird dazu eine der Konstanten aus Abb. 5-2 zugewiesen.
  Um das Ende der Fensterliste leicht feststellen zu können, wird immer ein leeres Listenelement mit Typ `kein_wert` angehängt.

Abb. 5-2
Konstanten Fenstertyp

```
#define     standard_fenster            1
#define     standard_fenster_gross      2
#define     arbeits_fenster             3
#define     menu_fenster                4
#define     eingabe_fenster             5
#define     text_fenster                6
```

- `toplevel`, `mainwindow` und `haupt_fenster` sind Widgets, die in jeder Widgethierarchie gebraucht werden, unabhängig vom Fenstertyp.
- in der `union fenster` sind Untertypen aufgeführt, auf die je nach Typ zugegriffen wird.
  Die Untertypen sind ab Kap. 6.1 erläutert.
- Sobald in einem Fenster ein Event erwartet wird, wird `event_klasse` eine der Konstanten von Abb. 5-3 zugewiesen. Solange kein Event reserviert ist, wird hier `xgap_keine_reservierung` gespeichert. `xgap_event_default` kann in den Fenstern verwendet werden, in denen es nur einen Event gibt.

Abb. 5-3
Konstanten für Events

```
#define     xgap_event_aktion           1
#define     xgap_event_eingabe          2
#define     xgap_event_menu             3
#define     xgap_event_popup_menu       4
#define     xgap_event_auswahl          5
#define     xgap_event_position         6

#define     xgap_event_default          9

#define     xgap_keine_reservierung    -89
#define     xgap_kein_event            -77
```

- `event_eingetroffen` ist FALSE, solange kein (vorher reservierter) Event eingetroffen ist. Damit wird die Ankunft eines gewünschten Events festgestellt.
- in `text_1` und `text_2` werden Zeichenketten gespeichert, die von X Windows über die Callback-Funktionen an die Fenster übergeben werden. Mit diesen Texten kann z.B. ein Exemplarname und ein E/A-Name zurückgeliefert werden, wenn das entsprechende Objekt angeklickt wurde.
- `position` speichert bei `xgap_event_position` die Cursorposition relativ zur linken oberen Ecke der Arbeitsfläche.
- in `wert` wird die Button-Nummer eines Aktionsbuttons gespeichert (`xgap_event_aktion`).
- `naechster` zeigt auf das nächste Listenelement.

Auf die einzelnen Komponenten dieser Struktur wird mit Funktionen aus dem FENSTER-Modul zugegriffen. Der Zugriff auf den Fenstertyp erfolgt beispielsweise mit den Funktionen:

```
speichere_fenster_typ (fenster_typ_zeiger adresse,
                       int  zu_speichernder_typ);
```

und `int lese_fenster_typ (fenster_typ_zeiger adresse);`

Die erste Funktion speichert den übergebenen Integerwert in der `typ`-Komponente der übergebenen Adresse. Die zweite Funktion gibt den Integerwert zurück, der momentan in der `typ`-Komponenten der übergebenen Adresse gespeichert ist.

Events können sowohl synchron als auch asynchron abgefragt werden (jeweils auf ein bestimmtes Fenster bezogen). Eine synchrone Abfrage bedeutet, daß auf den gewünschten Event vom aufrufenden Programm gewartet wird. Asynchron heißt, daß zunächst getestet wird, ob der gewünschte Event bereits reserviert ist. Wenn nein, wird er reserviert und die Event-Warteschlange des Anwendungsprogrammes (für jeden Client speichert X die ankommenden Events) abgefragt, ob der reservierte Event dabei ist. Wenn ja, wird er zurückgegeben, wenn nein, wird `xgap_kein_event` zurückgegeben.
Wenn `abfrage_xgap_event` ohne Eventklasse aufgerufen wird, wird getestet, ob ein zuvor reservierter Event eingetreten ist.
Danach wird im Anwendungsprogramm fortgefahren, bis wieder die Funktion `abfrage_xgap_event` aufgerufen wird. Der genaue Ablauf dieser Funktion ist in Abb. 5-4 dargestellt.

Die Funktion `x_warteschlange_lesen` besteht aus zwei `Xt`-Funktionsaufrufen in einer Schleife:

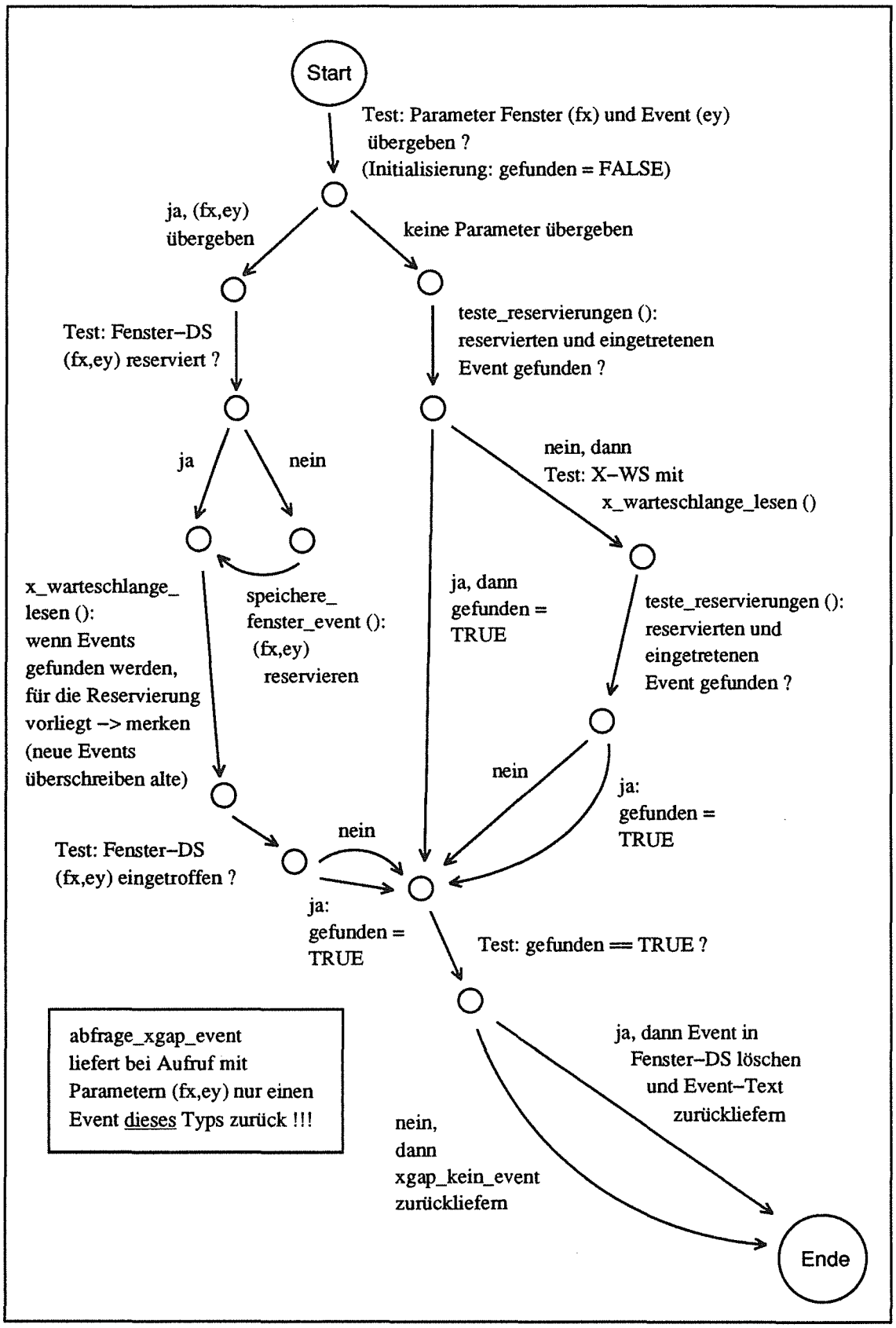
```
ende = FALSE;
do {
    if (XtPending () != 0)
    {
        XtNextEvent (&event);
        XtDispatchEvent (&event);
    }
    else
        ende = TRUE;
} while (ende == FALSE);
```

Mit `XtPending` wird überprüft, ob in der X-Event-Liste noch weitere Events vorhanden sind. Wenn ja, wird der nächste Event angefordert (`XtNextEvent`) und verarbeitet (`XtDispatchEvent`). Wenn nicht, wird die Schleife beendet. Die Abfrage mit `XtPending` muß eingefügt werden, weil die Funktion `XtNextEvent` solange blokkiert, bis wieder ein Event eintrifft.
Mit `XtDispatchEvent` werden die Callback-Funktionen des Moduls `XGAP_CLB` aufgerufen.
Jede dieser Callback-Funktionen startet `speichere_xgap_event`, als Parameter wird neben der Widget-Adresse der entsprechende Eventtyp übergeben. Außerdem je nach Funktion ein Text oder eine Position.
Die Funktion `menu_callback` wird beispielsweise aufgerufen, wenn ein Eintrag in einem Menüfeld oder einem Menüfenster (s. Kap. 6.3) geklickt wird. Der Event ist

dann `xgap_event_menu`, der Text die Zeichenkette des Menüeintrags. Ein Ablaufdiagramm von `speichere_xgap_event` ist in Abb. 5-5 dargestellt.

Abb. 5-4

Funktion abfrage_xgap_event



Mit suche_fenster_zu_widget wird die zum Widget gehörende Fensteradresse festgestellt. Dazu wird die Fensterliste duchlaufen und getestet, in welcher Widget-Hierarchie der einzelnen Fenster das übergebene Widget vorhanden ist (je nach Fenstertyp werden die Widget-Adressen ermittelt und mit übergebener Adresse verglichen). Sobald die zugehörige Fensteradresse gefunden wurde, wird sie von dieser Funktion zurückgeliefert.

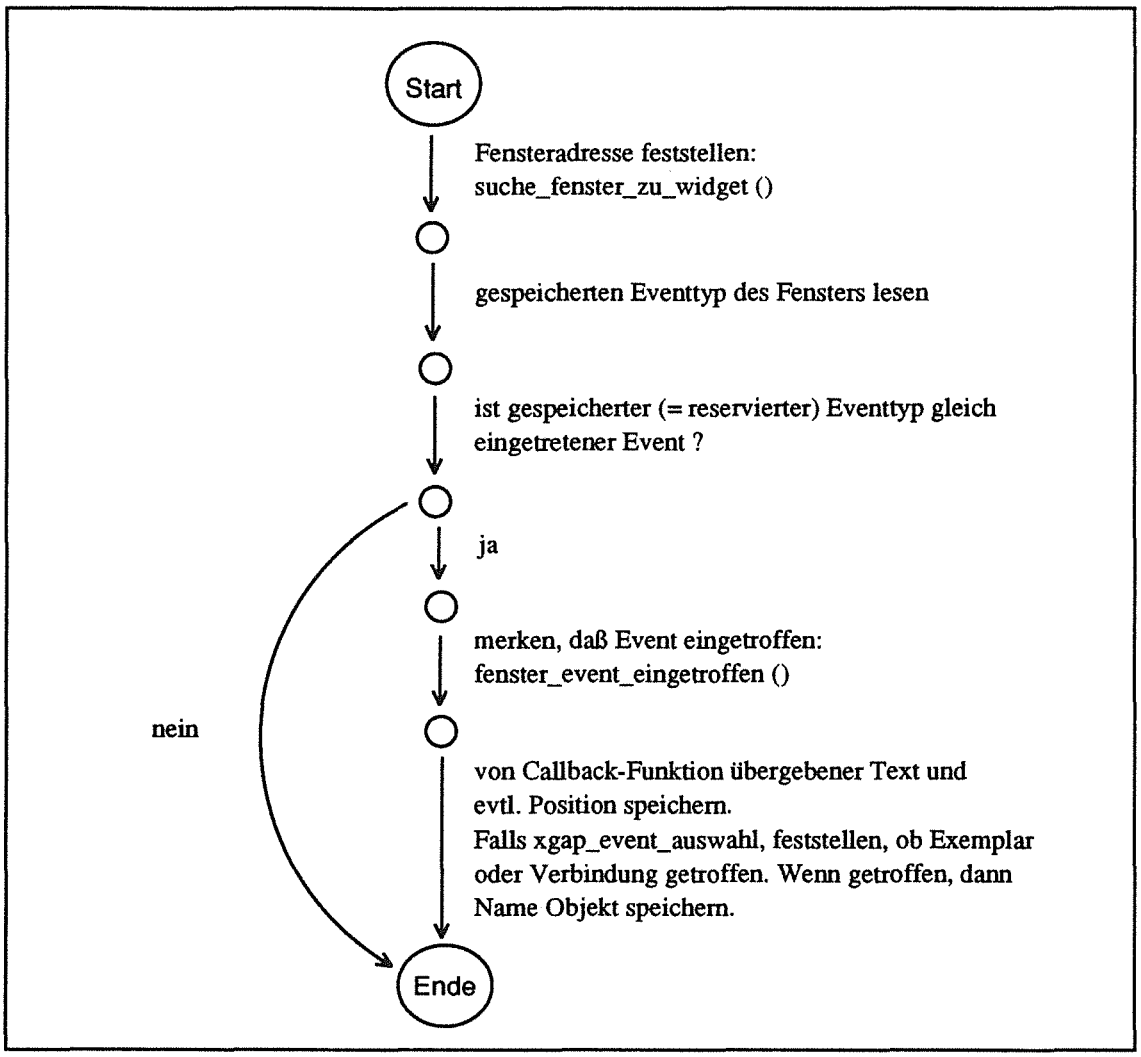


Abb. 5-5
Funktion speichere_xgap_event

6 Fensterarten mit Widget-Hierachien

Im folgenden werden die Unterstrukturen beschrieben, die sich in der `union` Fenster der Struktur `fenster_typ` befinden (s. Abb. 5-1).
Zwei Fensterarten, nämlich die Standard- und die Arbeitsfenster dienen dazu, mit dem Editor erzeugte Objekte auszugeben und zu verbinden. Die verwendeten Datentypen werden in den folgenden Abschnitten erläutert.
Sämtliche Fenster werden mit der Funktion `erzeuge_fenster` (Modul `XGAP_FENSTER`) erzeugt. Die obersten drei Widgets der Hierarchie sind bei jedem Fenstertyp gleich (s. Abb. 6-1).

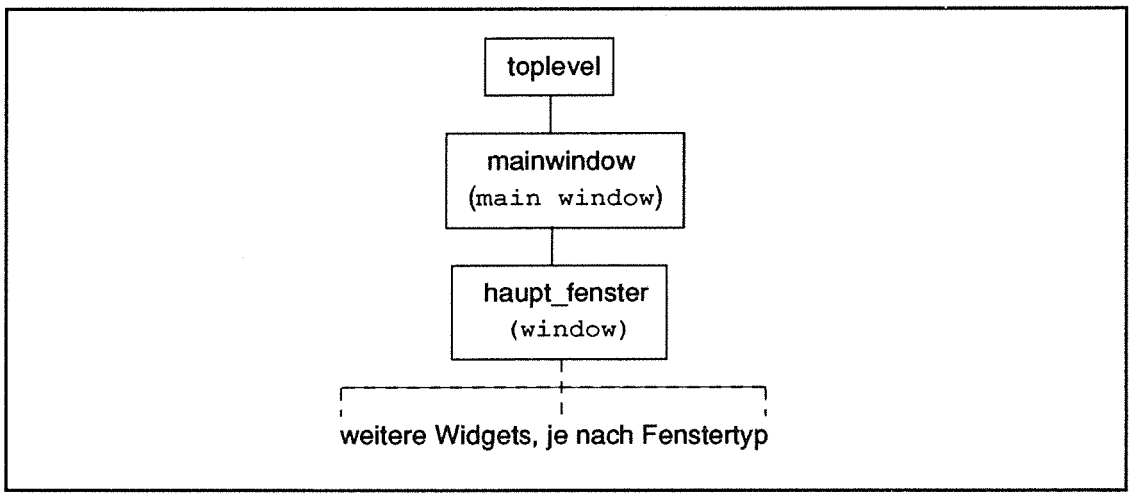


Abb. 6-1
Widget-Hierarchie XGAP-Fenster

Beim Start des XGAP-Systems muß zuerst `init_xgap` (Modul `XGAP_FENSTER`) aufgerufen werden. Damit wird ein Fenster vom Typ `standard_fenster_gross` erzeugt, das den gesamten Bildschirm belegt (es kann auch iconifiziert gestartet werden, s. [Keller 1991]).
Mit `XtInitialize` wird eine Verbindung Client-Server aufgebaut und mit dem Funktionsaufruf `erzeuge_toplevel (..)` die zurückgegebene Adresse in der Fenster-Datenstruktur gespeichert.
Alle weiteren Fenster werden als unabhängige sog. *Application-Shells* erzeugt. Jedes Fenster kann einzeln iconifiziert und verschoben werden. Das mit der Funktion `init_xgap` erzeugte erste Fenster wird erst bei Programmende gelöscht. Es muß bestehenbleiben, um die Kommunikation mit dem Server aufrechtzuerhalten.

6.1 Standardfenster

Das Standardfenster gibt es in zwei Größen. Die große Version mit dem Typ `standard_fenster_gross` wird zu Beginn erzeugt, alle weiteren Standardfenster haben den Typ `standard_fenster`. Die Koordinaten der einzelnen Widgets sind in `XGAP_KONST.h` als Konstanten definiert. Ein Standardfenster besteht aus mehreren Feldern, die in Abb. 6-2 dargestellt sind.

Falls ein Feld mehrere Widgets benötigt, sind diese (und eventuelle weitere Variablen) in einem `struct`-Typ zusammengefasst. Die einzelnen Felder haben folgende Bedeutung:

- Statusfeld (Struktur `status_feld_typ`)
  zur Ausgabe von Meldungen.
- Menüfeld (Struktur `menu_feld_typ`),
  hier können Auswahl-Texte eingetragen werden, die als Buttons angeklickt werden können. Diese Auswahl kann abgefragt werden. Werden mehr Einträge ins Menüfeld geschrieben als hineinpassen, kann man mit einem Verschiebebalken die nicht sichtbaren Einträge sichtbar machen.

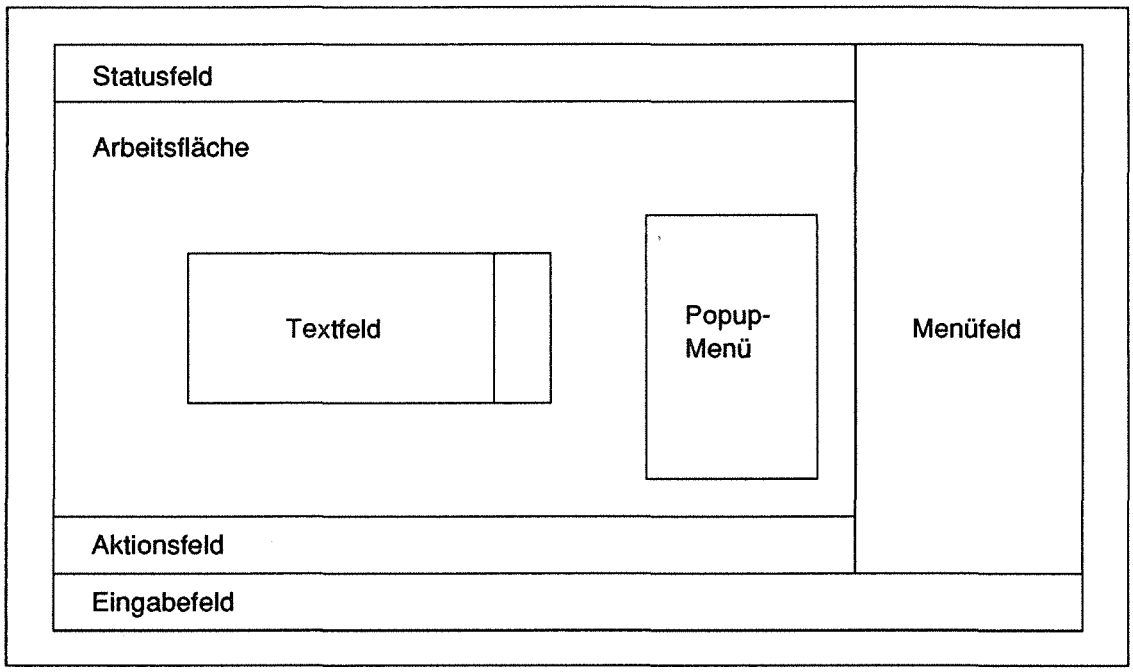


Abb. 6-2
Felder Standardfenster

- Aktionsfeld (Struktur `aktions_feld_typ`)
  mit acht Buttons, die individuell belegt werden können.
- Eingabefeld,
  in diesem Feld kann der Benutzer nach Aufforderung Daten eingeben.
- Textfeld (Struktur `text_feld_typ`);
  dieses Feld erscheint nur bei Bedarf auf dem Bildschirm an einer festen Position. Hier können Textdateien ausgegeben werden. Falls der auszugebende Text nicht komplett in das Fenster paßt, kann man den Textausschnitt mit einem Verschiebebalken verändern. Das Textfeld verschwindet nach Drücken eines zugehörigen Ende-Buttons wieder.
- Popup-Menü (Struktur `popup_feld_typ`)
  hat die gleichen Möglichkeiten wie das Menüfeld, wird aber nur nach Aufruf angezeigt und kann beim Aufruf innerhalb der Arbeitsfläche beliebig positioniert werden.

Die Unterstruktur `standard_fenster_typ` besteht aus weiteren Unterstrukturen (s. Abb. 6-4).

In Abb. 6-3 ist die Widget-Hierarchie dargestellt, die damit erzeugt wird.
Das Menüfeld, das Popup-Menüfeld und das Textfeld wurden mit einer *Listbox* realisiert, die einen Verschiebebalken besitzt und automatisch den ausgegebenen Text an

die Fenstergröße anpaßt.

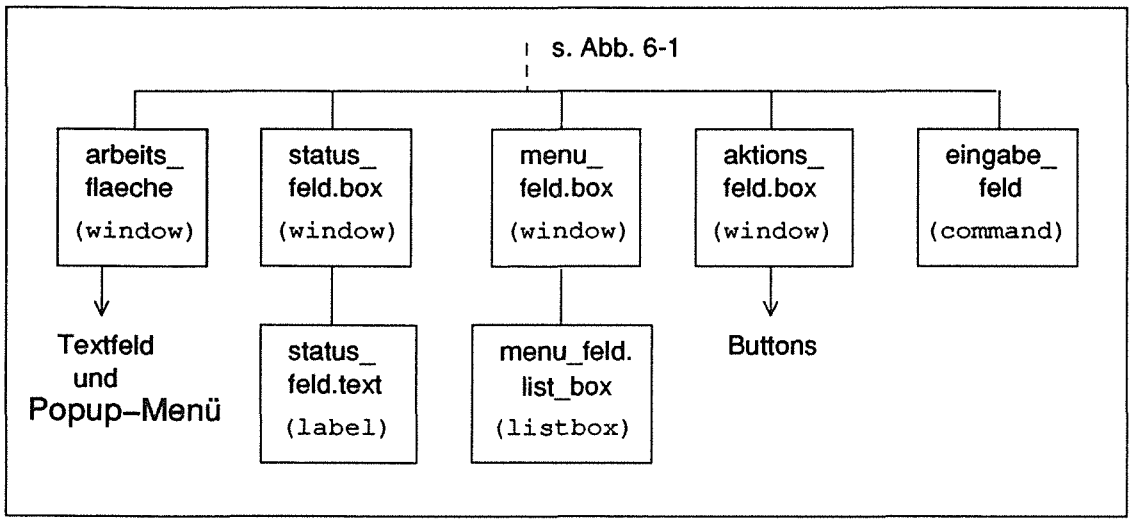


Abb. 6-3
Widget-Hierarchie Standard-Fenster

Das Eingabefeld ist ein *command window* des Widgets `haupt_fenster`; zu Beginn ist es deaktiviert und deswegen keine Eingabe möglich.
Die Buttons werden, wie im Editor, als eine Widget-Liste erzeugt, auf die mit Konstanten als Indizes zugegriffen werden kann; z.B wird mit

```
aktions_feld.button[aktion_1]
```

der erste Button angesprochen.

```
typedef struct {
            Widget                  arbeits_flaeche;
            Widget                  eingabe_feld;
            status_feld_typ         status_feld;
            menu_feld_typ           menu_feld;
            aktions_feld_typ        aktions_feld;
            text_feld_typ           text_feld;
            popup_feld_typ          popup_feld;
            exemplar_typ_zeiger     beginn_exemplare;
            verbindungs_typ_zeiger  beginn_verbindung;
            Bool                    exemplar_vorhanden;
            Window                  ausgabe_window;
            float                   skalierung;
               } standard_fenster_typ;
```

Abb. 6-4
Struktur standard_ fenster_typ

Die letzten fünf Komponenten der Struktur `standard_fenster_typ` (Abb. 6-4) haben folgende Bedeutung:

- `beginn_exemplar` zeigt auf den Beginn der Exemplarliste (s. Abb. 6-5), `beginn_verbindung` auf den Beginn der Verbindungsliste. Jedes Standardfenster hat eigene Objektlisten.
- `exemplar_vorhanden` wird TRUE gesetzt, sobald für dieses Fenster mindestens ein Exemplar vorhanden ist.

- In `ausgabe_window` wird die Adresse des X-Windows gespeichert, in der graphische Primitive ausgegeben werden (damit werden die graphischen Objekte dargestellt).
- `skalierung` wird in der aktuellen Programmversion nicht verwendet.

6.1.1 Exemplare

Die im Standardfenster dargestellten Exemplare bestehen jeweils aus Segment- und E/A-Listen (s. Abb. 6-5).

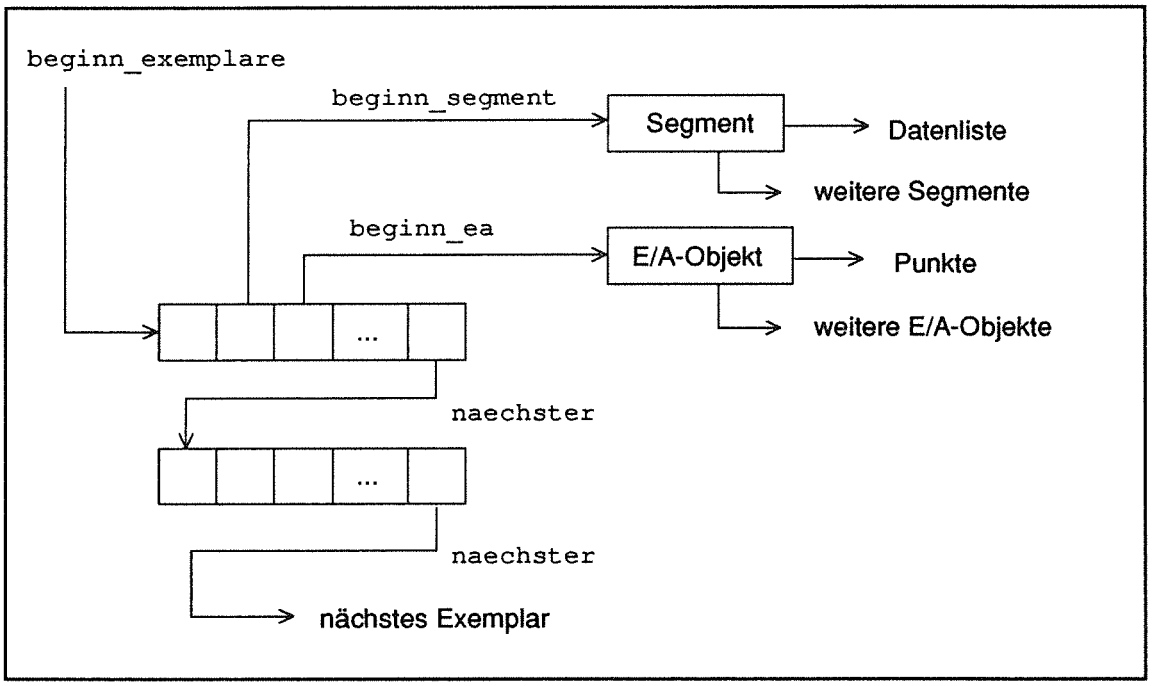


Abb. 6-5
Aufbau der Exemplarliste

Die einzelnen Komponenten der `exemplar`-Struktur (s. Abb. 6-6) haben folgende Bedeutung:

- `objekt_name` speichert den VMS-Dateinamen des Vorlageobjekts.
- `name` ist der Exemplarname (zur Vereinfachung der Speicherverwaltung wurde die Länge der Zeichenketten auf `max_laenge_text` Zeichen beschränkt). Jeder Name kommt in der Liste nur einmal vor.
- In `beginn_segment` und `beginn_ea` werden Zeiger auf den Anfang der Segment- und E/A-Listen gespeichert.
- `anzahl_segmente` und `anzahl_ea` wird beim Erzeugen eines Exemplares aus einem Vorlageobjekt bestimmt,
- genauso `min, breite` und `hoehe`. Diese Variablen speichern die Umrissdaten des Exemplares.
- `ausgeben` ist ein Schalter, der beim Anfordern von Exemplar-Speicherplatz zunächst auf `FALSE` gesetzt wird. Er wird in der Funktion `ausgabe_objekte` geprüft und das Exemplar nur ausgegeben, wenn der Schalter `TRUE` ist. Damit können Objekte logisch gelöscht und wieder angezeigt werden.

- Von dem Rechteck, welches das Exemplar umgibt, wird eine `region` gespeichert. Damit kann einfach festgestellt werden, ob der Benutzer ein Exemplar angeklickt hat.
- `naechster` zeigt auf das nächste Exemplar.

```
typedef struct {
        char                    *objekt_name;
        char                    name[max_laenge_text+1];
        segment_typ             *beginn_segment;
        ea_typ                  *beginn_ea;
        int                     anzahl_segmente;
        int                     anzahl_ea;
        XPoint                  min;
        int                     breite, hoehe;
        Bool                    ausgeben;
        Region                  region;
struct exemplar_typ_deklaration  naechster;
    }   exemplar_typ_deklaration;

typedef exemplar_typ_deklaration      *exemplar_typ_zeiger,
                                      exemplar_typ;
```

Abb. 6-6
Struktur exemplar_typ

Die Typdeklarationen für Exemplare, Verbindungen, Segmente und E/A-Objekte sind zweigeteilt. Bei Exemplaren gibt es z.B. zwei Deklarationsdateien mit den Namen

- `EXEMPLAR_P.h` und
- `EXEMPLAR.h`.

In der ersten Datei befindet sich nur eine Deklaration für den Exemplartyp, ohne seine Komponenten. `EXEMPLAR_P.h` wird in allen Modulen verwendet, die auf die Exemplar-Struktur nur indirekt über Funktionen zugreifen. Die zweite Datei mit der vollständigen Deklaration, `EXEMPLAR.h`, wird nur im Modul `EXEMPLAR` verwendet. (Genauer gesagt wird, weil in der Sprache C ein Typ zur Übersetzungszeit vollständig deklariert sein muß, auch in `EXEMPLAR_P.h` die vollständige Deklaration geladen).

6.1.2 Segmente

In Abb. 6-8 ist die Struktur `segment_typ` dargestellt, auf die von den Exemplaren aus zugegriffen wird. Ein Segment dient zur Darstellung graphischer Primitive mit verschiedenen Attributen.

Mit der `segment_typ`-Struktur wird eine einfach verkettete Liste erzeugt, in der die Segmentattribute und Koordinatenwerte gespeichert sind. Für diese Struktur wird bei Bedarf Speicher angefordert.
Die einzelnen Variablen (s. Abb. 6-7) haben folgenden Zweck:

- `modus` gibt den Segment-Typ an, der Komponenten wird eine der Konstanten aus Abb. 6-9 zugewiesen (2 bis 8).

- `umriss_farbe`, `fuell_farbe`, `kanten_breite` und `strichart` sind Attribute des jeweiligen Segmentes. Die möglichen Werte sind in Abb. 6-9 dargestellt.

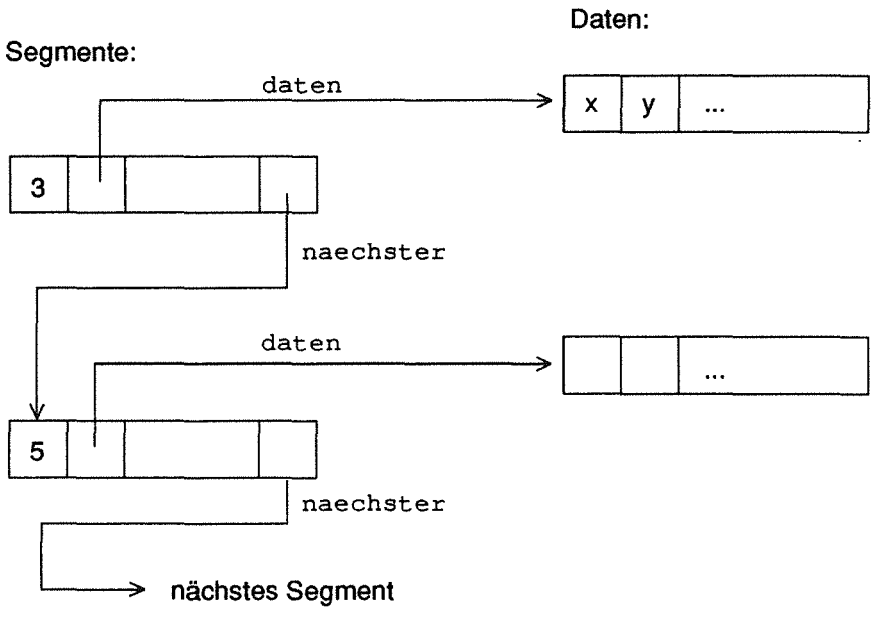
Abb. 6-7
Struktur segment_typ

```
typedef struct {
                    int             modus;
                    int             *daten;
                    int             anzahl_daten;
                    Bool            geloescht;
                    int             umriss_farbe;
                    int             fuell_farbe;
                    int             kanten_breite;
                    int             strichart;
                    Region          region;
  struct segment_typ_deklaration  *naechster;
                } segment_typ_deklaration;

typedef segment_typ_deklaration   *segment_typ_zeiger,
                                   segment_typ;
```

- Speicher für die Integer-Liste `daten` wird bei Bedarf angefordert und enthält die Koordinatenwerte (s. Abb. 6-8 und 6-10).
- In `anzahl_daten` steht, wieviele Integerwerte gespeichert sind.
- Wenn `geloescht` gleich TRUE, wird das Segment nicht mehr ausgegeben.

Abb. 6-8
Aufbau der Segmentliste



- Eine `Region` in X ist eine beliebige Menge von Pixel auf dem Bildschirm, normalerweise entweder eine rechteckige Fläche, mehrere überlappende oder benachbarte rechteckige Flächen oder ein Polygon (aus [Nye 1990-1]).

Regionen werden für Strecken, Quadrate, Rechtecke und Polylines (dort erst beim Suchen von Segmenten) erzeugt. Wenn der Benutzer im Editor ein Segment anklicken möchte, kann mit Hilfe dieser Region festgestellt werden, ob es getroffen wurde.

```
#define     modus_auswahl          0
            modus_ea               1
            modus_strecke          2
            modus_polyline         3
            modus_quadrat          4
            modus_rechteck         5
            modus_kreis            6
            modus_kreisbogen       7
            modus_text             8

#define     farbe_weiss            0
            farbe_schwarz          1
            farbe_blau             2
            farbe_rot              3
            farbe_gruen            4

#define     breite_schmal          0
            breite_mittel          1
            breite_fett            2

#define     strichart_normal       0
            strichart_gestrichelt  1
            strichart_gepunktet    2
```

Abb. 6-9
Konstanten für Modus, Farbe, Kantenbreite und Kantenart

- `naechster` gibt das nächste Element in der Liste an (s. Abb. 6-8).

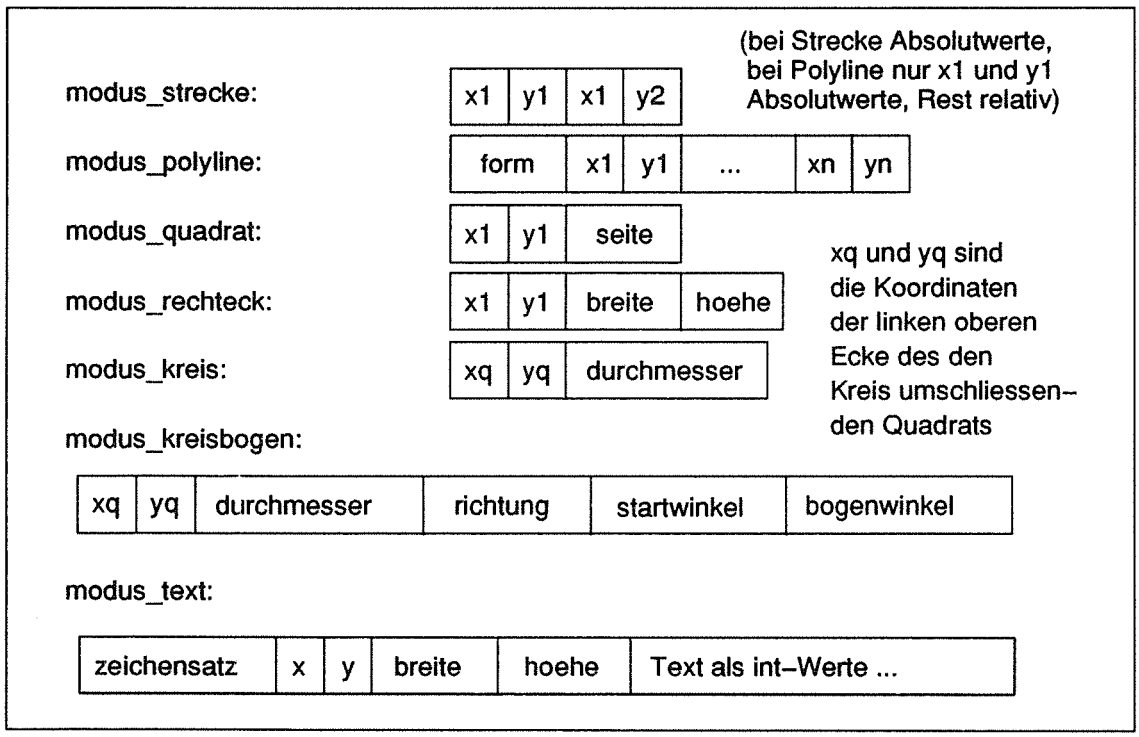


Abb. 6-10
Inhalt der Datenliste

6.1.3 E/A-Objekte

Für die aus mehreren Polylines zusammengesetzten E/A-Objekte werden zwei Strukturen benötigt, einmal `ea_typ` (s. Abb. 6-11) und zum zweiten `ea_objekt_typ` (s. Abb. 6-14) mit Vorlageobjekten.

Abb. 6-11
Struktur ea_typ

```
typedef struct {
            char                        *name;
            char                        *text;
            int                         typ;
            int                         richtung;
            int                         groesse;
            int                         umriss_farbe;
            int                         fuell_farbe;
            ea_objekt_deklaration       objekt;
            Bool                        geloescht;
            Region                      region;
      struct ea_typ_deklaration         *naechster;
            } ea_typ_deklaration;

typedef ea_typ_deklaration   *ea_typ_zeiger, ea_typ;
```

Mit der Struktur `ea_typ` (s. Abb. 6-11) wird eine einfach verkettete Liste geschaffen, um die Eingangs- und Ausgangsobjekte zu speichern (s. Abb. 6-13). Auch hier wird der Speicher dynamisch angefordert. Die Struktur enthält folgende Variablen:

- `name` ist ein Zeiger auf eine Zeichenkette mit dem Namen des E/A-Objekts.
- `text` ist eine Zeichenkette mit optionalem Text (Zusatzinformationen) zum Objekt.
- `typ` gibt an, ob es sich um ein Eingangs-, Ausgangs- oder Durchgangsobjekt handelt (Durchgangsobjekte können als Eingangs- und als Ausgangsobjekte verwendet werden)
- `richtung`, ob das Objekt nach links oder rechts zeigt (s. Abb. 6-12).
- `groesse` ist eine von drei Größen (s. Abb. 6-12).

Abb. 6-12
Konstanten für E/A-Typ, Richtung und Größe

```
#define     eingang     0
            ausgang     1
            durchgang   2

#define     links       0
            rechts      1

#define     ea_klein    0
            ea_mittel   1
            ea_gross    2
```

- `umriss_farbe` und `fuell_farbe` können wie bei den Segmenten gewählt werden.
- In `objekt` sind die Daten der das E/A-Objekt darstellenden Polyline gespeichert.
- `geloescht` ist die Löschkennzeichnung.
- in `region` werden die Umrisse des E/A-Objektes gespeichert, um beim Anklikken einfach feststellen zu können, ob es getroffen wurde.
- `naechster` verweist auf das folgende Listenelement.

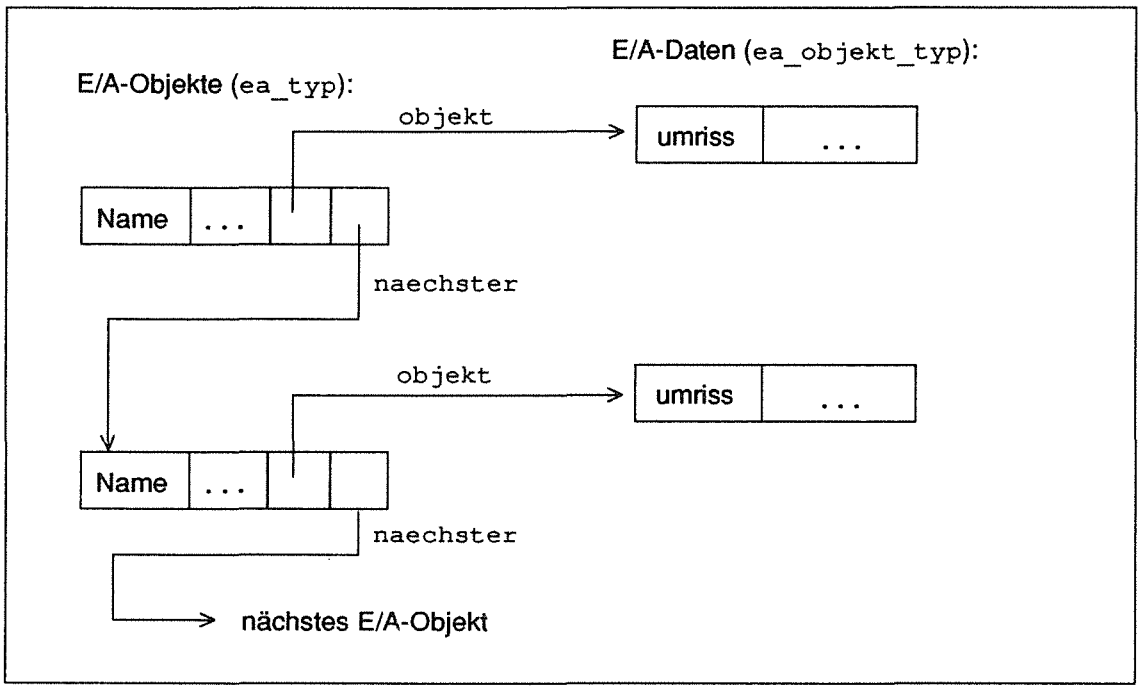


Abb. 6-13
Aufbau der E/A-Liste

Die E/A-Objekte bestehen aus zwei geschlossenen Polylines mit jeweils sieben Punkten. Die Variable `ea_objekt` (s. Abb. 6-14) dient dazu, 18 verschiedene E/A-Objekte als Vorlage zu speichern (3 Typen mal 2 Richtungen mal 3 Größen); die Daten dieser vorgegebenen Objekte werden in der Funktion `erzeuge_ea_vorlagen` (Modul `EDIT_OBJ`) zugewiesen. Die einzelnen Variablen haben folgende Bedeutung:

- In `umriss` sind die Koordinaten der ersten Polyline gespeichert.
- `markierung` speichert die zweite Polyline. Damit wird markiert, ob es sich um ein Eingangs- , Ausgangs- oder Durchgangsobjekt handelt.
- `pkt_links_oben` ist die linke obere Ecke des umschließenden Rechtecks.
- `breite` und `hoehe` sind die Daten des das Objekt umgebende Rechtecks.
- `pkt_eingang` und `pkt_ausgang` sind die Koordinaten, die beim späteren Zeichnen von Verbindungen gebraucht werden.

Abb. 6-14

Struktur ea_objekt

```
typedef struct {
            XPoint      umriss[7];
            XPoint      markierung[7];
            XPoint      pkt_links_oben;
               int      breite;
               int      hoehe;
            XPoint      pkt_eingang;
            XPoint      pkt_ausgang;
                } ea_objekt_deklaration;

ea_objekt_deklaration    ea_objekt[anzahl_typen]
                                  [anzahl_richtungen]
                                  [anzahl_groessen];
```

6.1.4 Verbindungen

Mit der Datenstruktur `verbindungs_typ` wird eine einfach verkettete Liste von Verbindungen erzeugt; die einzelnen Komponenten sind in Abb. 6-15 aufgeführt. Jede Verbindung besteht aus einer Polyline mit vier Punkten (d.h. drei Teilstrecken), die ein Ausgangs- mit einem Eingangsobjekt verbinden. Die Variablen haben folgenden Zweck:

- `name` speichert den Namen der Verbindung (jeder Name ist eindeutig).
- `von_exemplar` und `nach_exemplar` speichern die Namen von Quell- und Zielexemplaren.
- `von_ausgang` und `nach_eingang` enthalten die Namen der E/A-Objekte.
- `farbe`, `kanten_breite` und `strichart` sind Attribute der Verbindung, es werden die auch im Editor verwendeten Konstanten benutzt.
- Wenn `name_ausgeben` `TRUE` ist, wird der Name der Verbindung an der Position `pos_name` ausgegeben.
- `punkte` speichert die Koordinaten der Polyline.
- `ausgeben` wird auf `TRUE` gesetzt, wenn die Verbindung ausgegeben werden soll. Damit ist, genauso wie bei den Exemplaren, ein logisches Löschen möglich.
- `naechster` zeigt auf die nächste Verbindung.

```
typedef struct {
        char            name[max_laenge_text+1];
        char            von_exemplar[max_laenge_text+1];
        char            nach_exemplar[max_laenge_text+1];
        char            von_ausgang[max_laenge_text+1];
        char            nach_eingang[max_laenge_text+1];
        int             farbe;
        int             kanten_breite;
        int             strichart;
        Bool            name_ausgeben;
        XPoint          pos_name;
        XPoint          punkte[4];
        Bool            ausgeben;
        struct verbindungs_typ_deklaration    naechster;
        } verbindungs_typ_deklaration;

typedef verbindungs_typ_deklaration   *verbindungs_typ_zeiger,
                                      verbindungs_typ;
```

Abb. 6-15
Struktur verbindungs_typ

6.2 Arbeitsfenster

Auf einem Arbeitsfenster können, genauso wie im Standardfenster, zugehörige Exemplare und Verbindungen dargestellt werden. Die Unterschiede zum Standardfenster sind, daß die Größe des Arbeitsfensters beim Erzeugen (`erzeuge_fenster`) bestimmt werden kann und daß das Fensters keine weiteren Felder hat. Zur Widget-Hierarchie von Abb. 6-1 kommt nur ein weiteres hinzu.
Die Größe wird mit einer `float`-Zahl angegeben, bezogen auf die Arbeitsfläche des Standardfensters. Der Faktor 1.2 bedeutet die 1,2-fache Größe.
In der folgenden Abb. 6-16 ist die Datenstruktur dargestellt:

Abb. 6-16
Struktur arbeits_ fenster_typ

```
typedef struct {
            Widget                      arbeits_flaeche;
            exemplar_typ_zeiger         beginn_exemplare;
            verbindungs_typ_zeiger      beginn_verbindung;
            Bool                        exemplar_vorhanden;
            Window                      ausgabe_window;
            float                       skalierung;
        } arbeits_fenster_typ;
```

6.3 Menüfenster

Ein Menüfenster dient, genauso wie das Menüfeld oder Popup-Menü des Standardfensters, zur Ausgabe von Wahlmöglichkeiten, die angeklickt und abgefragt werden können.

Abb. 6-17
Struktur menu fenster_typ

```
typedef struct {
            Widget      box;
            Widget      list_box;
            int         anzahl;
           } menu_feld_typ;

typedef struct {
            menu_feld_typ menu_feld;
           } menu_fenster_typ;
```

In Abb. 6-16 ist die Datenstruktur dargestellt, die dafür benötigt wird. Für die Einträge wird eine Listbox erzeugt, in `anzahl` steht die aktuelle Anzahl der Einträge. Dieser Wert wird zum Löschen der Einträge benötigt. Die Komponente `box` wird im Fenster nicht benutzt (nur im Menüfeld des Standardfensters).

6.4 Eingabefenster

Abb. 6-18 zeigt ein Eingabefenster, der OK-Button ruft die Callback-Funktion auf, um die Eingabe zu speichern, der Lösch-Button löscht die aktuelle Eingabe.
Die zugehörigen Callback-Funktionen werden, genauso wie die Callback-Funktionen

der anderen Fenster und Felder, erst aufgerufen, wenn die Event-Warteschlange abgefragt wird. Dies geschieht durch eine synchrone oder asynchrone Abfrage.

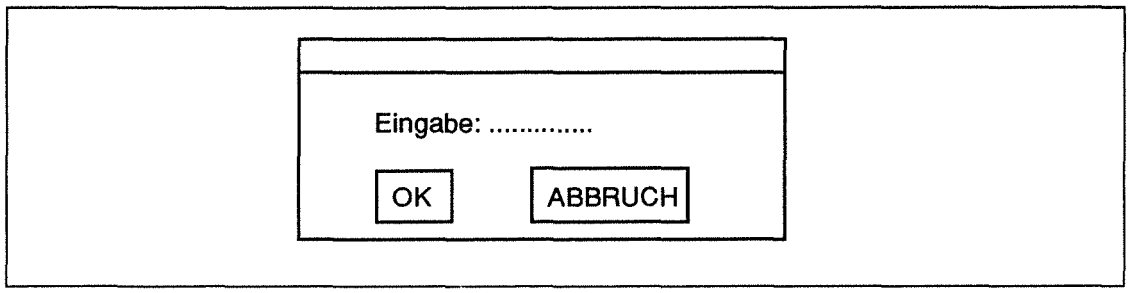


Abb. 6-18
Eingabefenster-

Die Struktur `eingabe_fenster_typ` besteht aus vier Widgets (s. Abb. 6-19):

```
typedef struct {
            Widget      text_label;
            Widget      text_eingabe;
            Widget      button_ok;
            Widget      button_loeschen;
          } eingabe_fenster_typ;
```

Abb. 6-19
Struktur eingabe fenster_typ

- `text_label` für den Text 'Eingabe',
- `text_eingabe` ist ein Simple-Text-Widget des DEC-Toolkits,
- `button_ok` und `button_loeschen` sind zwei Push-Buttons.

6.5 Textfenster

Ein Textfenster hat die gleiche Aufgabe wie ein Textfeld des Standardfensters, nämlich die Ausgabe von Textdateien. Der Unterschied ist, wie schon beim Menüfenster, daß es sich um eine unabhängige Widget-Hierarchie handelt. Die Datenstruktur ist in Abb. 6-20 dargestellt:

```
typedef struct {
            Widget      box;
            Widget      text;
            Widget      button;
            int         anzahl_zeilen;
          } text_feld_typ;

typedef struct {
            text_feld_typ text_feld;
          } text_fenster_typ;
```

Abb. 6-20
Struktur text_ fenster_typ

Die Komponenten `box` und `button` werden nur im Textfeld des Standardfensters benutzt, nicht in den eigenständigen Textfenstem. In `anzahl_zeilen` wird gespeichert, wieviele Textzeilen ausgegeben sind.

7 Fensterfunktionen

7.1 Exemplare und Verbindungen

In den folgenden Kapiteln werden Funktionen zum Erzeugen, Löschen, Manipulieren, Laden und Speichern von Exemplaren und Verbindungen beschrieben (Modul `XGAP_BIB`):

- `erzeuge_exemplar`
- `ausgabe_exemplar`
- `verbinde_exemplare`
- `ausgabe_verbindung`
- `verschiebe_exemplar`
- `speichere_exemplare`
- `lade_exemplare`
- `loesche_exemplar`
- `loesche_verbindung`

Alle aufgeführten Funktionen beziehen sich auf ein (Standard- oder Arbeits-) Fenster, der erste Parameter ist deswegen immer der Fenstername.
In jeder Funktion wird zu Beginn geprüft, ob der Fenstername in der Fensterliste vorhanden ist. Wenn ja, wird getestet, ob der Fenstertyp zu der aufrufenden Funktion paßt (z.B. kann ein Befehl `ausgabe_exemplar` nicht bei einem Menüfenster aufgerufen werden). Ist einer der beiden Tests nicht erfolgreich, wird die Funktion mit dem `return`-Wert `FALSE` beendet.

Die synchrone Abfrage eines Auswahl-Events geschieht mit `auswahl_objekt`, eines Positions-Events mit `ermittle_koordinaten`.

Weitere Funktionen sind in [Keller 1991] aufgeführt. Sie dienen zum

- Ändern von E/A-Namen und E/A-Texten,
- Ändern von Segment-Texten und
- Ausgeben von Verbindungsnamen.

7.1.1 erzeuge_exemplar

Diese Funktion erzeugt ein Exemplar, ohne es auszugeben. Die Ausgabe erfolgt mit `ausgabe_exemplar` (s. Kap. 7.1.2).
Zu Beginn der Prozedur wird geprüft, ob ein Exemplar mit dem übergebenen Namen bereits in der Liste vorhanden ist. Falls das der Fall ist, wird eine Fehlermeldung ausgegeben und die Funktion beendet (`return`-Wert `FALSE`).

Nach dem Öffnen der Vorlagedatei (vom XGAP-Editor) wird die Kennung überprüft, um festzustellen, ob es sich um eine XGAP-Datei handelt. Falls die richtige Kennung gelesen wird, wird in der Exemplarliste des Fensters das letzte Exemplar gesucht. Es wird, wie auch bei den anderen Listen, immer im voraus erzeugt und als leer gekenn-

zeichnet. Bei Exemplaren und Verbindungen geschieht dies, indem dem ersten Zeichen des Namens (`exemplar_name[0]`) das Zeichen '\0' zugewiesen wird.
Danach wird der erste Speicherplatz für die Segment- und E/A-Liste angefordert und die Adressen gespeichert.
Nach dem Lesen weiterer Exemplarattribute wie Anzahl der Segmente und der Umrisskoordinaten wird mit

```
objekt_laden
```

ein mit dem Editor erzeugtes Objekt geladen.

Wenn das Vorlageobjekt erfolgreich geladen wurde, wird der Schalter `exemplar_vorhanden` auf `TRUE` gesetzt (jetzt können Exemplare gespeichert werden).
Mit `umrechnung_exemplar_daten` werden alle absoluten Koordinatenwerte in relative Werte umgerechnet, die sich auf die linke obere Ecke (`exemplar->min`) des zugehörigen Exemplares beziehen. Die Eckkoordinaten wurden bereits im Editor ermittelt und in der Objekt-Datei mitgespeichert.
Zuletzt wird Speicher für das nächste Exemplar angefordert und die Funktion mit dem Rückgabewert `TRUE` beendet.

7.1.2 ausgabe_exemplar

Damit wird ein mit der Funktion `erzeuge_exemplar` erzeugtes Objekt auf der Arbeitsfläche ausgegeben.
Nach dem Testen des Fensternamens wird mit `suche_exemplar` die Adresse des gewünschten Exemplares festgestellt.
Wenn der Exemplarname in der Liste gefunden wurde, wird der Ausgabeschalter auf `TRUE` gesetzt. Danach wird mit

```
lese_xgap_wert (xgap_positions_art);
```

festgestellt, welche Art der Positionierung gewählt wurde. Wenn die Positionsart `pos_koordinaten` ist, werden die mit

```
setze_show_wert (xgap_x, x-Koordinate);
```
und
```
setze_show_wert (xgap_y, y-Koordinate);
```

gesetzten Koordinaten verwendet, bei `pos_default` wird das Exemplar in der linken oberen Ecke ausgegeben.
Danach werden mit den geänderten Bezugskoordinaten (`lese_exemplar_min`) Regionen erzeugt, eine für das gesamte Exemplar (`erzeuge_region_exemplar`), weitere für die E/A-Objekte (`erzeuge_regionen_ea`).
Die Regionen werden erst bei der Ausgabe erzeugt, weil erst jetzt die aktuellen Bezugskoordinaten bestimmt sind und damit die Position des Objekts auf der Arbeitsfläche.
Zuletzt wird das Exemplare mit `ausgabe_objekt` im zum Fenster gehörenden X-Window (die Adresse ist in den Fenster-Daten gespeichert) ausgegeben .
Falls es bereits ausgegeben ist, werden mit `ausgabe_xgap_objekte` alle Objekte dieses Fensters neu ausgegeben, um das Exemplar an der alten Position zu löschen.

7.1.3 verbinde_exemplare

Die Funktion verbindet zwei Exemplare, die zwei Parameter sind der Fenster- und Verbindungsname. Nach der Überprüfung, ob bereits eine Verbindung mit solch einem Namen existiert, werden die Objektnamen gelesen, die an die Prozedur mit `setze_xgap_text` übergeben wurden (s. [Keller 1991]) und die Existenz der zugehörigen Objekte überprüft (`suche_exemplar` und `suche_ea`).
Wenn alle Objekte vorhanden sind, werden die Namen und Attribute gespeichert. Wie bei `erzeuge_exemplar` wird ein freies Element der Verbindungsliste gesucht.
Danach wird neuer Speicherplatz für die nächste Verbindung angefordert (`anfordern_verbindungs_speicher`).
In dieser Funktion werden noch keine Polyline-Punkte berechnet, dies geschieht erst bei `ausgabe_verbindung`.

7.1.4 ausgabe_verbindung

Wie bei den Exemplaren werden auch die Verbindungen nach ihrer Erzeugung nicht sofort ausgegeben. Erst in `ausgabe_verbindung` wird zunächst geprüft, ob die zugehörigen Exemplare ausgegeben sind. Wenn ja, werden die Verbindungskoordinaten mit `ermittle_verbindungs_punkte` berechnet. Jede Verbindung besteht aus zwei horizontalen und einer vertikalen Strecke. Der arithmetische Mittelwert zwischen der x-Koordinate des Ausgangs- und Eingangspunktes wird als x-Koordinate der vertikalen Strecke verwendet (`mittel_x`). Die weiteren Koordinaten lassen sich leicht bestimmen, die beiden horizontalen Strecken haben die y-Koordinaten des zugehörigen Objektpunktes.
Nach der Koordinatenberechnung wird die Verbindung ausgegeben.

7.1.5 verschiebe_exemplar

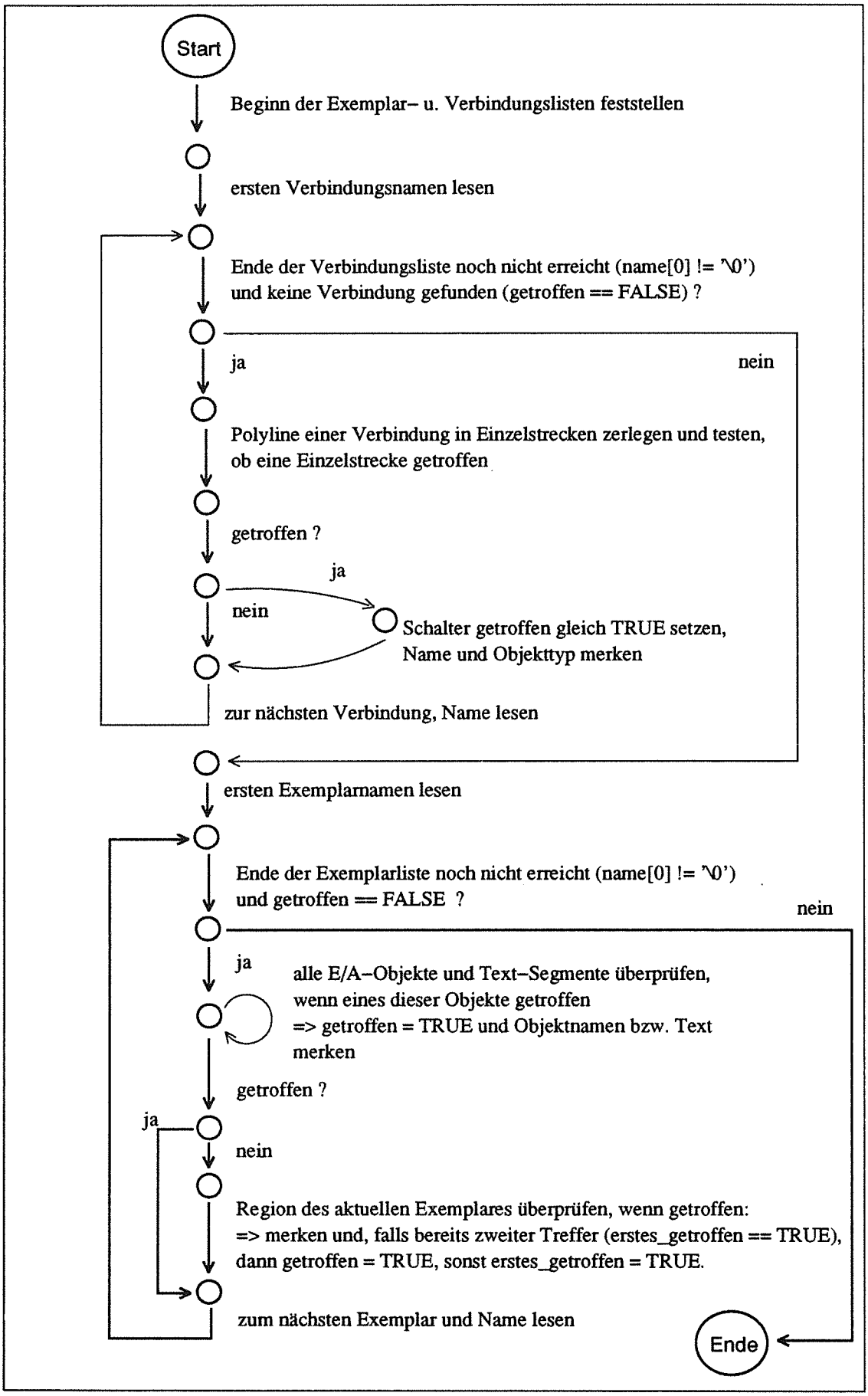
Mit dieser Funktion können Exemplare, die auf der Arbeitsfläche ausgegeben sind, interaktiv verschoben werden. Eventuell vorhandene Verbindungen werden automatisch neu berechnet.
Zunächst wird die Funktion `ermittle_koordinaten` aufgerufen; hier wird gewartet, bis der Benutzer die Arbeitsfläche anklickt und es werden die Cursorkoordinaten gespeichert.
Diese Werte werden von der Verschiebe-Funktion gelesen und an `teste_exemplare` übergeben. Dort werden alle Verbindungen, Exemplare, E/A-Objekte und Text-Segmente überprüft. In Abb. 7-1 ist ein Ablaufdiagramm dieser Funktion dargestellt. Mit

```
setze_xgap_wert (xgap_typ, typ_...)
```

wird dort gespeichert, ob ein Exemplar (`typ_exemplar`), eine Verbindung (`typ_verbindung`) oder nichts (`kein_wert`) getroffen wurde. Beim Verschieben sind nur die Exemplare von Bedeutung; ein Treffer wird mit Hilfe ihrer Regionen festgestellt.
Da sich innerhalb eines Exemplares ein weiteres befinden kann (d.h. es können gleichzeitig zwei Betrachtungsstufen dargestellt werden), muß nach dem ersten Treffer noch weitergesucht werden. Erst ein zweiter Treffer oder das Listenende beenden die Suche.

Abb. 7-1

Funktion teste_exemplare



Falls ein Exemplar getroffen wurde, wird in `verschiebe_exemplar` mit den Umrisskoordinaten des Exemplares das umgebende Rechteck bestimmt und dieser Bereich aus dem Ausgabefenster in einen Hintergrundspeicher (`pixmap`) kopiert. Der Bereich wird danach verschoben (bzw. vom Hintergrundspeicher an neue Positionen im Ausgabefenster kopiert), bis die Maustaste losgelassen wird.

Damit die Verschiebefunktion nicht sofort beendet wird, wenn ein Benutzer den Mausbutton drückt und unabsichtlich gleich wieder losläßt, wird bei der Ende-Abfrage zusätzlich überprüft, ob sich eine Koordinate verändert hat (und damit der Bereich verschoben wurde).
Nach dem Verschieben werden die Bezugskoordinaten des Exemplares geändert, neue Regionen erzeugt, betroffene Verbindungen geändert und die gesamte Arbeitsfläche neu ausgegeben. Die als Hintergrundspeicher verwendete Pixmap wird vor Beendigung der Funktion wieder gelöscht (`XFreePixmap`).

7.1.6 speichere_exemplare

Mit dieser Funktion werden Exemplare und Verbindungen in einer Datei gespeichert. Die Daten werden in folgender Reihenfolge gesichert:

- Dateikennung (Konstante `datei_kennung_xgap`)
- Anzahl Exemplare
- Anzahl Verbindungen
- Exemplar-Daten, jeweils mit
  - Länge Name u. Name
  - linke obere Ecke
  - Breite und Höhe
  - Anzahl Segmente und E/A-Objekte
  - Segment- und EA-Daten
- Verbindungen mit
  - Länge Verbindungsname u Name
  - Start- u. Zielexemplarname, jeweils mit Länge
  - Name Ausgangs- u. Eingangsobjekt mit Länge
  - Farbe, Kantenbreite u. Strichart

7.1.7 lade_exemplare

Beim Laden von Exemplaren und Verbindungen werden im Fenster vorhandene Objektlisten gelöscht und mit

```
anfordern_exemplar_speicher ()
```

und

```
spchr_exemplar_bgn_std ()
```

neuer Speicherplatz für das erste Exemplar angefordert und die Adresse gespeichert (std = Standardfenster). Das Gleiche geschieht für die Verbindungsliste. Danach werden die Objekte geladen in der Reihenfolge, wie sie in `speichere_exemplare` beschrieben ist.

7.1.8 loesche_exemplar

Mit dieser Funktion wird ein Exemplar vom Bildschirm (logisch) oder im Speicher (physisch) gelöscht. Die Löschart wird vor Aufruf der Funktion mit

```
setze_xgap_wert (xgap_loesch_art, loeschen_...)
```

bestimmt.
In `loesche_exemplar` wird zunächst geprüft, ob sich am zu löschenden Exemplar Verbindungen befinden. Wenn ja, werden diese zuerst gelöscht mit `loesche_verbindung`. Danach wird die Exemplar-Liste durchsucht und, wenn ein Exemplar gefunden wurde, dieses Element mit dem gleichen Algorithmus wie in `edit_loeschen_callback` (Kap. 8.9.2) entfernt. Falls es nur logisch gelöscht werden soll, wird nur der Ausgabeschalter des Exemplars auf `FALSE` gesetzt (`exemplar_nicht_ausgeben`).

7.1.9 loesche_verbindung

Der Algorithmus zum Entfernen eines Elementes aus der Verbindungsliste ist der gleiche wie in `loesche_exemplar`. Auch hier ist ein logisches (`xgap_loesch_art == loeschen_logisch`) oder physisches Löschen möglich.

7.1.10 auswahl_objekt

Nach dem Fenstertest wird mit

- `ermittle_koordinaten` (s. Kap. 7.1.11) gewartet, bis eine Position auf der Arbeitsfläche des Fensters angeklickt wird und
- mit `teste_exemplare` (s. Abb. 7-1) geprüft, ob ein Objekt getroffen wurde.

Die Ausgabeparameter werden von `teste_exemplare` gesetzt, sie sind in [Keller 1991] beschrieben.

7.1.11 ermittle_koordinaten

Für die synchrone Abfrage wird die asynchrone Funktion

```
abfrage_xgap_event
```

verwendet. Sie wird so lange aufgerufen, bis im gewünschten Fenster der gewünschte Event (hier `xgap_event_position`) eintrifft.
Danach wird die Position gespeichert.

7.2 Aktionsfeld

Mit den Funktionen für das Aktionsfeld können Aktions-Buttons belegt, gelöscht und abgefragt werden.
Die synchrone Funktion `auswahl_aktion` wartet auf ein Event mit dem Typ `xgap_event_aktion` (der Algorithmus ist der Gleiche wie bei `auswahl_objekt`).

7.3 Menüfeld und Menüfenster

Die Einträge im Menüfeld (und -Fenster) werden, im Gegensatz zum Aktionsfeld, nur im Widget gespeichert und nicht in der zugehörigen Variablen `menu_feld` (Struktur `menu_feld_typ`). Das Widget `menu_feld.list_box` ist vom Typ *listbox*, die Einträge werden vom Widget verwaltet und können gescrollt werden. Wenn z.B der erste Eintrag gelöscht wird, rücken die anderen Einträge automatisch nach. Das Listbox-Widget bietet keine Möglichkeit, einen Eintrag zu aktivieren oder zu deaktivieren.

Die Funktionen

- `neuer_menu_eintrag`,
- `loesche_menu_eintrag`,
- `loesche_menu_feld` (Löschen der Einträge, nicht des Feldes)
- und `auswahl_menu`

können sowohl für das Menüfeld im Standardfenster als auch für eigenständige Menüfenster verwendet werden.
In den jeweiligen Funktionen werden mit dem Fenstertyp die entsprechenden Widget-Adressen ermittelt.

In `neuer_menu_eintrag` wird die übergebene Zeichenkette dem Widget weitergegeben, mit `loesche_menu_eintrag` wird ein Eintrag gelöscht.
Die Funktion `loesche_menu_feld` löscht alle Einträge. Die Anzahl der Einträge wird in `menu_feld.anzahl` gespeichert; dieser Variablenwert wird von den vorherigen Funktionen erhöht oder erniedrigt.

Die vollständigen Parameter sind in [Keller 1991] aufgeführt.

7.4 Popup-Menü

Popup-Menüs gibt es nur bei Standardfenstern, deswegen können die folgenden Funktionen nur dafür verwendet werden.
Zum Eintragen und Löschen werden die gleichen Möglichkeiten wie beim Menüfeld angeboten:

- `neuer_popup_eintrag`
- `loesche_popup_eintrag`
- `loesche_popup_feld.`

Zusätzlich wird bei `loesche_popup_feld` auch die Popup-Box vom Bildschirm gelöscht. Nachdem Texte an das Popup-Menü übergeben wurden, muß es mit der Funktion `ausgabe_popup_menufeld` sichtbar gemacht werden. In der Ausgabe-Funktion wird mit

```
lese_xgap_wert (xgap_positions_art)
```

überprüft, welche Positionierungsart vorgewählt wurde. Ist die Art `pos_koordinaten`, werden die zuvor gesetzten Koordinaten gelesen und mit `XtSetArg` und `XtSetValues` dem Widget übergeben.
Falls die Position automatisch ermittelt werden soll (`pos_automatisch`), wird ein

vor dem Aufruf der Funktion gespeicherter Exemplarname gelesen, die Adresse des zugehörigen Exemplares festgestellt und das Popup-Menü rechts neben dem Exemplar positioniert:

```
x = exemplar_min.x + exemplar_breite + 5;
y = exemplar_min.y;
```

Falls sich das Exemplar am rechten Rand befindet, wird das Popup-Menü auf dem Exemplar ausgegeben.

7.5 Eingabefeld und Eingabefenster

Mit lese_eingabe kann sowohl das Eingabefeld eines Standardfensters, als auch die Eingabe von einem Eingabefenster gelesen werden.

Bei der asynchronen Abfrage mit `abfrage_xgap_event` und `xgap_event_eingabe` ist folgendes zu beachten:
Sobald der Benutzer das Eingabe-Widget anklickt und damit aktiviert, blockiert `XtNextEvent` (d.h. die Abfrage-Funktion `XtPending` meldet, daß noch weitere Events vorhanden sind). Erst nachdem die Callback-Funktion für die Eingabe aufgerufen wurde, kann der *Input-Focus* wieder zurückgesetzt werden und damit weitere Funktionen asynchron weiterlaufen.

7.6 Textfeld und Textfenster

Zur Ausgabe einer Textdatei wird die gleiche Widget-Art wie im Menüfeld, eine zeilenorientierte Listbox, verwendet. Die DEC-Widget-Toolbox bietet kein Widget an, um einen beliebigen Text direkt auszugeben.
Die Textdatei wird in der Funktion `textdatei_ausgeben` zum Lesen geöffnet, zeichenweise gelesen und in die Variable `zeile` kopiert. Die maximale Zeichenanzahl ist `text_feld_zeichen`. An das Ende jeder Zeile wird ein Endezeichen (`'\0'`) angefügt, danach wird die Zeile an das Listbox-Widget übergeben. Wenn das Ende der Datei ereicht ist, wird sie geschlossen und das Textfenster angezeigt.
Das Löschen des Textfeldes im Standardfenster geschieht mit der Callback-Funktion, die dem Ende-Button zugewiesen ist (`text_feld_callback`).

Die Ausgabe von neuem Text löscht bestehenden alten.

8 Der graphische Editor

8.1 Zielsetzung

Aufgabe war die Entwicklung eines graphischen Editors zur Erzeugung von Objekten, die nach ihrer Erstellung vom Editor weiterbearbeitet, aber auch unabhängig davon benutzt werden können. Mit dem graphischen Editor sollen graphische Primitive erzeugt werden können (ähnlich DEC-Paint). Diese Grundelemente haben mehrere Eigenschaften wie beispielsweise eine Farbe und eine bestimmte Kantenbreite. Diese Eigenschaften können verändert werden, indem man das Element anklickt und danach einen der Buttons im Auswahlfenster (s. Abb. 8-1) aktiviert.

8.2 Funktionalität

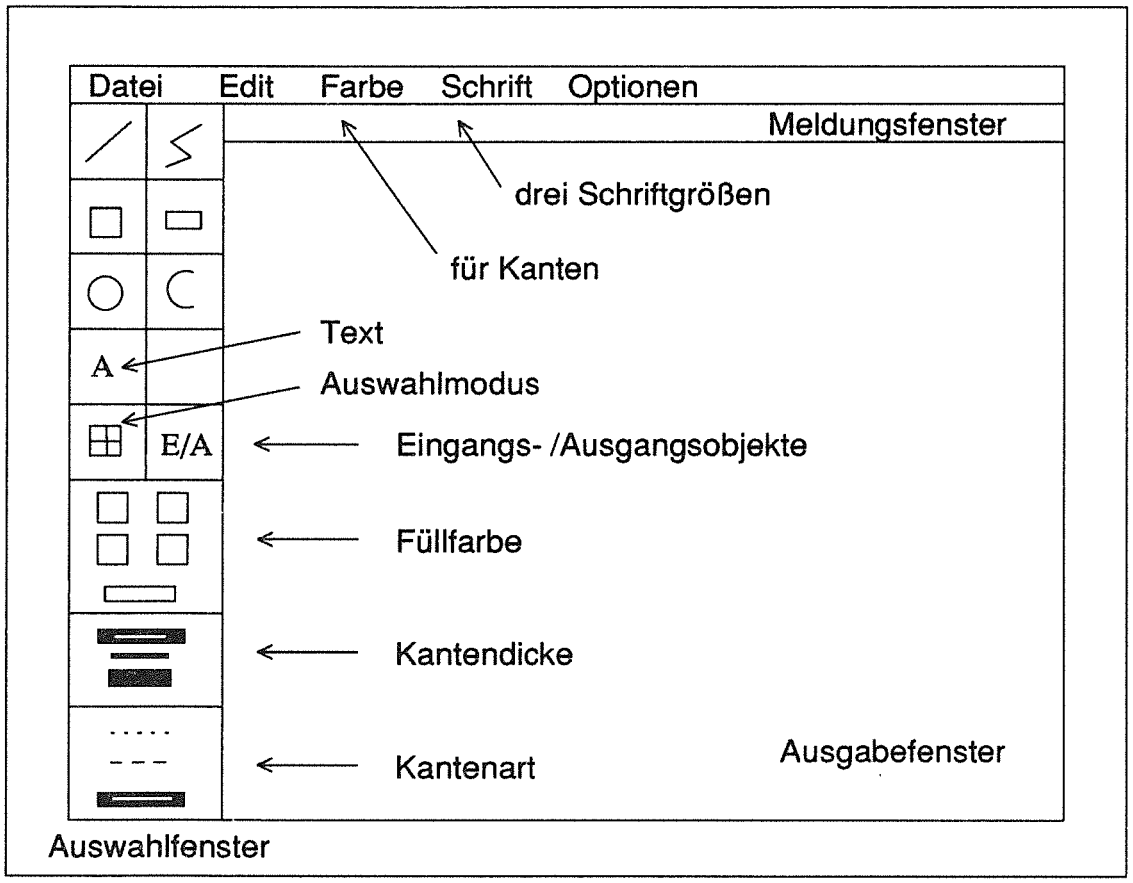


Abb. 8-1
Der XGAP-Editor

Mit dem graphischen Editor kann man Objekte

- erzeugen,
- verändern (Position, Attribute) und löschen,
- speichern und laden.

Ein graphisches Objekt kann aus mehreren Grundelementen und Eingangs- bzw. Ausgangsobjekten (E/A-Objekten) bestehen. Ein E/A-Objekt ist ein geschlossener Poly-

gonzug.
Um Teilobjekte verändern zu können, wird für jedes Grundelement und E/A-Objekt ein Segment erzeugt. Ein Segment in der graphischen Datenverarbeitung ist eine Zusammenfassung von graphischen Primitiven, die als eine abgeschlossene Einheit manipuliert werden kann [Newman 1986]. Zur Erzeugung der graphischen Objekte werden folgende Grundelemente angeboten (sie werden durch Anklicken von einem der Buttons im Auswahlfenster aktiviert, s. Abb. 8-1):

- Strecken
- verbundene Strecken (auch als Polyline bezeichnet, geschlossen oder offen)
- Rechteck und Quadrat
- Kreis und Kreisbogen
- Text mit verschiedenen Größen
- E/A-Objekte mit verschiedenen Größen.

Jedes der gezeichneten Grundelemente hat folgende Attribute:

- Farbe der Kanten
- Füllfarbe (nur geschlossene Objekte)
- Kantendicke
- Kantenart (gepunktet, gestrichelt oder durchgezogen)

Es wird gleich bei Programmstart ein Objektname vergeben, der während der Sitzung nicht geändert werden kann. Man kann, während der Sitzung, bereits bestehende Objekte (als Vorlage) laden, die auch den zu Beginn vergebenen Objektnamen erhalten, wenn man sie speichert.

In Anhang A wird beschrieben, wie Objekte mit dem Editor erstellt und manipuliert werden.

8.3 Aufbau des Programmes

In Abb. 8-2 ist die "benutzt"-Hierarchie der Editor-Module dargestellt. Gestrichelte Pfeile bedeuten Zuweisungen von Callback-Funktionen, die Ziffern in Klammern sind die Modulnummern.

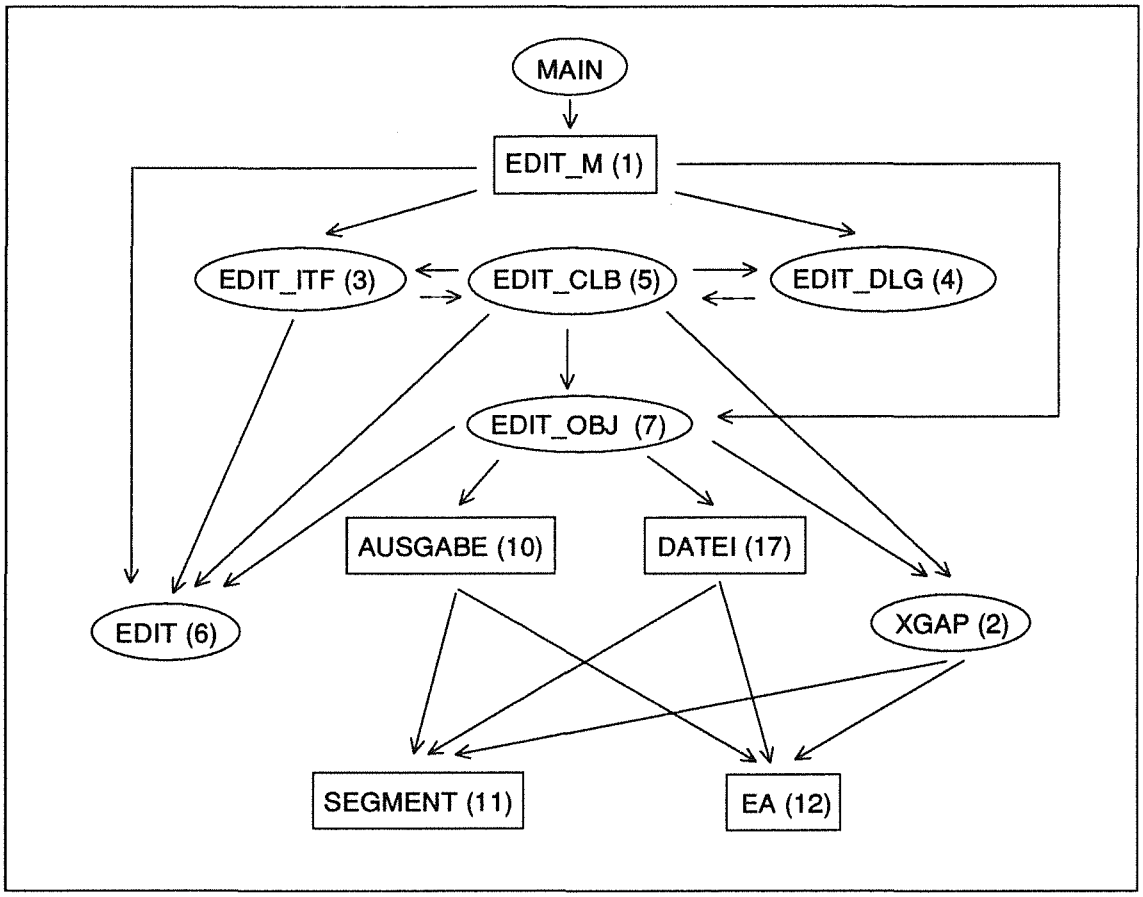


Abb. 8-2
Modul-Hierarchie Editor

8.3.1 Module EDIT_M, EDIT_ITF und EDIT_DLG

In diesen Modulen werden

- die Widgets und damit die Benutzeroberfläche erzeugt,
- die Variablen initialisiert
- und auf X-Events gewartet.

Das Modul `EDIT_M` besteht nur aus der Funktion `starte_xgap_editor`. Wird diese Funktion zum ersten Mal aufgerufen, werden die Editor-Widgets neu erzeugt, bei allen weiteren Aufrufen werden sie nur noch neu angezeigt (`XtRealize`). Mit

- `init_editor` wird die Zustandsvariable des Editors initialisiert,
- `init_objekt` die ersten Elemente der Segment- und E/A-Liste erzeugt,
- `init_dialoge` die Dialogboxen initialisiert.

Der Editor läuft synchron, d.h. solange vom Editor Events verarbeitet werden, können keine weiteren Funktionen des Hauptprogramms ausgeführt werden.

Mit einem Schalter wird festgestellt, ob der Editor beendet werden soll (bzw. es wird dann mit der Event-Verarbeitung aufgehört).

8.3.2 Modul EDIT_CLB

Sämtliche Callback-Funktionen des Editors befinden sich im Modul `EDIT_CLB`, die zugehörigen Teile der Benutzeroberfläche lassen sich in vier Gruppen einteilen:

- Pulldown-Menü (`Datei, Edit, Farbe, Schrift, Optionen`),
- Auswahlfenster (mit Buttons zur Auswahl der Zeichenelemente, Füllfarben, Kantenbreiten und Stricharten),
- Ausgabefenster (Drücken der linken und mittleren Maustaste, `expose`-Events),
- Dialogfenster (Dateinamen, E/A-Objekte, Text, Hinweis).

Sobald man einen der Auswahl-Buttons drückt, wird eine zugehörige Callback-Funktion aufgerufen, die dann weitere Funktionen startet:

- Die Buttons der Zeichenelemente ändern den Zeichenmodus (s. Kap. 8.5.1, `editor_typ`) und rufen bei Text und E/A-Objekten Dialogfenster auf.
- Füllfarben, Kantenbreiten und Stricharten rufen Funktionen auf, die die neue Attributwahl speichern und diese Auswahl markieren (durch Änderung der Label in den Buttons). Wenn der Benutzer ein Objekt markiert hat, werden auch Objekt-Attribute geändert.

Die Pulldown-Menü-Einträge bieten weitere Möglichkeiten, das Programm zu steuern:

- In `Datei`, `Edit` und `Optionen` werden Callback-Funktionen aufgerufen, die in den Kapiteln 8.9 - 8.11 beschrieben werden.
- `Farbe` und `Schrift` dienen, wie die Attribut-Buttons, zur Vorwahl von Editor-Einstellungen, hier der Kantenfarbe und der Schriftgröße.

8.4 Widget-Hierarchie und zugehörige Datenstrukturen

Die Widget-Hierarchie wird mit den Modulen EDIT_ITF und EDIT_DLG erzeugt. Die oberste Ebene im Editor ist das `toplevel`-Window, danach kommt das `main`-Window (s. Abb. 8-3). Dieses Widget erhält vom Window-Manager das charakteristische Aussehen der DEC-Windows-Oberfläche. Positionswünsche für dieses Fenster werden vom Window-Manager ignoriert, 'er' bestimmt die Position. Mit diesem Widget wird außerdem die Gesamtgröße des Objekt-Editors bestimmt. Das nächste Widget ist das `haupt_fenster`, davon stammen alle weiteren Widgets (außer dem Pulldown-Menü) direkt oder indirekt ab. Die Widget-Hierarchie des Editors ist unabhängig von den Hierarchien der anderen Fenster.
Ein Child des `haupt_fenster` ist `ausgabe_fenster`, dies ist die Zeichenfläche. Damit sofort bestimmte Funktionen aufgerufen werden, wenn der Benutzer den linken oder mittleren Mausbutton innerhalb dieses Fensters drückt, erhält die Aktionstabelle dieses Widgets zwei neue Einträge. Dazu werden zunächst die Events und die Funktionen bestimmt, die aufgrund der Events ausgeführt werden:

```
static char ausgabe_fenster_transl_table[] =
"<Btn1Down>: ausgabe_fenster_callback ()\n\
 <Btn2Down>: ea_aenderung_callback ()";
```

`<Btn1Down>` ist der Event, der beim Drücken der linken Maustaste gesendet wird. Danach werden die Adressen der Funktionen festgestellt:

```
static XtActionsRec ausgabe_fenster_action_table[] =
{
    {"ausgabe_fenster_callback",
     (caddr_t)ausgabe_fenster_callback},
    {"ea_aenderung_callback",
     (caddr_t)ea_aenderung_callback},
};
```

Mit

```
XtSetArg (arglist[], DwtNtranslations,
 XtParseTranslationTable (ausgabe_fenster_transl_table));
```

werden die neuen Events der Übersetzungstabelle des Widgets hinzugefügt und mit

```
XtAddActions (ausgabe_fenster_action_table, 2);
```

wird eine benutzerspezifische Aktionstabelle - hier Zeiger auf Funktionen - erzeugt. Mit dieser Methode lassen sich auch kompliziertere Events, wie z.B. Doppelklicks oder Tastenkombinationen, mit Callback-Funktionen verbinden.
Damit Objekte automatisch neu ausgegeben werden, wenn das Ausgabefenster verdeckt war, wird die Funktion `expose_callback` aufgerufen, sobald das Widget `ausgabe_fenster` einen Expose-Event erhält. In `expose_callback` werden dann die Objekte neu ausgegeben.

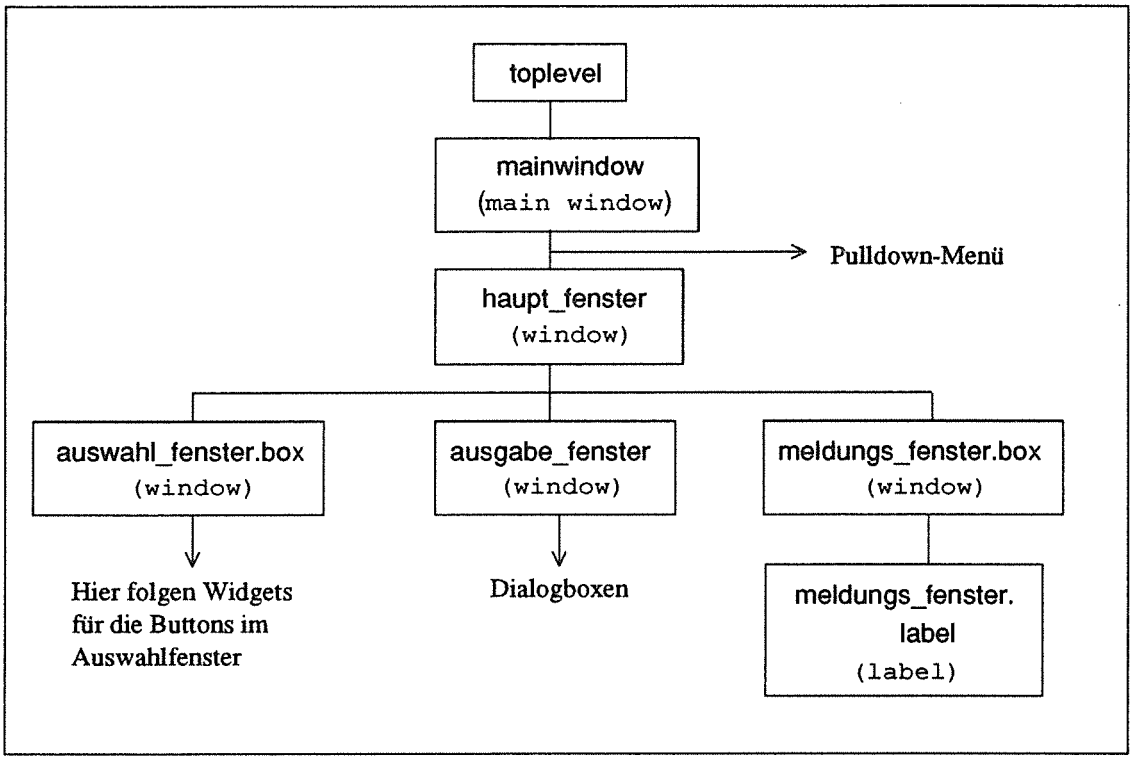


Abb. 8-3

Widget-Hierarchie Editor

In Abb. 8-3 ist ein Teil der Widget-Hierarchie des Editors dargestellt. Das Widget zur Aufnahme der Auswahl-Buttons am linken Rand heißt `auswahl_fenster.box`. Meldungen werden in `meldungs_fenster.label` ausgegeben.
Das Pulldownmenü-Widget heißt `pulldown_menu.menu_leiste` (Struktur `pulldown_menu`). Die Pulldownmenü-Einträge werden mit der Funktion `erzeuge_pulldown_menu` erzeugt.
Das Besondere daran ist, daß für jeden Haupteintrag eine Box erzeugt werden muß (Funktion `DwtMenuPulldownCreate`) und zu jeder von diesen Boxen ein Eintrag (Funktion `DwtPullDownMenuEntryCreate`) mit einem Argument, welches auf die Boxen verweist (`DwtsubMenuId`). Die Einträge, die mit Buttons realisiert werden, beziehen sich dann auf die mit `DwtMenuPulldownCreate` erzeugten Boxen.

Das Auswahlfenster (bzw. die zugehörige Struktur `auswahl_fenster`, s. Abb. 8-4 und Modul `EDIT_ITF`) besteht unter anderem aus einigen Buttons, denen jeweils ein Widget und zwei Pixmaps zugeordnet sind (Struktur `button_typ`). Eine der Pixmaps ist für den nichtaktiven (`na`) Button, das andere für den aktiven (`a`).
Innerhalb des Auswahlfensters gibt es vier Button-Gruppen:

- `button_typ button_element []`
  zur Auswahl des Zeichenelements,
- `button_typ button_farbe []`
  für die Füllfarben,
- `button_typ button_kante []`, `button_strichart []`
  für die Kantendicke und Stricharten.

Zur Indizierung dieser als Widget-Listen realisierten Gruppen dienen die im Konstantenteil von `XGAP_EDITOR_KONST.h` vereinbarten Namen. Diese Indizierung mit

Namen (die vom Preprozessor durch Konstanten ersetzt werden) wird im gesamten Programm verwendet, um es lesbarer zu machen.

```
typedef struct {
                Widget          widget;
                Pixmap          pix_na;
                Pixmap          pix_a;
                } button_typ;
typedef struct {
                Widget          box;
                button_typ      button_element[...];
                Widget          trenn_balken_1;
                button_typ      button_farbe[...];
                Widget          farb_label;
                Widget          trenn_balken_2;
                button_typ      button_kante[...];
                Widget          trenn_balken_3;
                button_typ      button_strichart[...];
                } auswahl_fenster_typ;

auswahl_fenster_typ auswahl_fenster;
```

Abb. 8-4 Struktur auswahl_fenster

Das Button-Widget für den Kreis im Auswahlfenster wird z.B. folgendermaßen erzeugt (s. Modul `INIT: erzeuge_buttons`):

```
auswahl_fenster.button_element[modus_kreis].widget =
    DwtPushButtonCreate (auswahl_fenster.box,
                         "bk", arglist, 7);
```

Nach der Variablen `auswahl_fenster` kommt die Button-Gruppe, hier `button_element`. Die dann folgende Konstante `modus_kreis` bestimmt den Button, `widget` ist der Zugriff auf den Widget-Teil der `button_typ` - Struktur. Die Funktion `DwtPushButtonCreate` ist eine Funktion der DEC-Widget-Bibliothek (deren Namen mit `Dwt` beginnen). Der erste Parameter gibt das in der Hierarchie darüberstehende Widget an, hier `auswahl_fenster.box`. Der Parameter `arglist` und die Ziffer 7 verweisen auf eine Argumentliste, die zuvor mit `XtSetArg` gesetzt wurde.
Weitere Widgets im Auswahlfenster dienen zur Trennung der Button-Gruppen (sog. *separator*-Widgets), damit werden waagrechte Balken und ein Label (`farb_label`) zur Anzeige der gewählten Füllfarbe erzeugt.
Sämtliche Widgets im Auswahlfenster werden mit der Funktion `erzeuge_buttons` realisiert.

Eine weitere Funktion, die im Modul EDIT_M aufgerufen wird, ist erzeuge_editor_dialoge. Hier werden alle Dialogboxen erzeugt, damit sie später schneller angezeigt werden können. Das komplexeste Dialogfenster erscheint, wenn der Benutzer den E/A-Button anklickt. Die zugehörige Struktur heißt ea_dialog_typ (s. Abb. 8-5 und Modul EDIT_DLG). Sie besteht aus mehreren Labels zur Beschriftung (label n), Text-Widgets zur Texteingabe (name zur Eingabe E/A-Name, text für zusätzliche Erläuterungen) und aus *toggle*-Buttons (z.B. toggle_links zur Auswahl eines E/A-Objektes, das nach links zeigt) in *radio*-Boxen. Ein Toggle-Button ist ein Button, der zwei Zustände haben kann. Mit Radio-Boxen werden mehrere Toggle-Buttons zusammengefaßt, von denen immer nur ein Button aktiviert sein kann (z.B. haben die beiden Toggle-Button-Widgets toggle_links und toggle_rechts den gleichen Vater in der Widget-Hierarchie).

Abb. 8-5
Struktur ea_dialog

```
typedef struct {
                Widget     box;
                Widget     label1;
                Widget     label2;
                Widget     name;
                Widget     label3;
                Widget     text;
                Widget     label4;
                Widget     radiobox_typ;
                Widget     toggle_eingang;
                Widget     toggle_ausgang;
                ...
                Widget     radiobox_richtung;
                Widget     toggle_links;
                Widget     toggle_rechts;
                ...
                Widget     ok;
                Widget     abbruch;
                int        typ;
                int        richtung;
                int        groesse;
                } ea_dialog_typ;

ea_dialog_typ ea_dialog;
```

In weiteren Komponenten der Variablen ea_dialog (typ, richtung und groesse) wird die momentane Auswahl gespeichert; diese Daten werden nach Drükken des OK-Buttons (Widget ok) verarbeitet.

8.5 Datenstrukturen für Editor und Objekte

Die folgenden Datenstrukturen werden im Modul EDIT verwendet.

8.5.1 editor_typ

```
typedef struct {
              int              modus;
              int              umriss_farbe;
              int              fuell_farbe;
              int              kanten_breite;
              int              strichart;
              int              zeichensatz;
              Bool             raster_ein;
              int              raster_groesse;
              Bool             datei_gespeichert;
              Bool             undo_moeglich;
              Bool             ea_namen_ausgeben;
              Bool             widgets_erzeugt;
              Bool             editor_ein;
              Window           button_window;
              Window           ausgabe_window;
              Pixmap           pixmap
          } editor_typ;

editor_typ   editor;
```

Abb. 8-6
Struktur editor_typ

Die Editor-Variable speichert den aktuellen Zustand des Editors, dazu gehören Modus, Farben usw. Im folgenden werden die einzelnen Variablen aus Abb. 8-6 erläutert:

- `modus:` Der Modus wird in `init_var` auf `modus_auswahl` gesetzt und durch Drücken der Buttons im Auswahlfenster geändert. Mit dem Modus wird bestimmt, was gezeichnet wird (die möglichen Modi und Werte für die Integer-Variablen sind als Konstanten vordefiniert, s. Abb. 6-9).
- `umriss_farbe` und `fuell_farbe` sind die Farben der Objektkanten und die Farben zum Füllen des geschlossenen Objekts. Fünf Farben werden angeboten (s. Abb. 6-9).
- Auch die `kanten_breite` und `strichart` werden vom Benutzer gewählt, es gibt jeweils drei Varianten (s. Abb. 6-9).
- `zeichensatz` speichert die gewählte Schriftgröße (s. Abb. 8-7).
- `raster_ein` wird `TRUE` gesetzt, sobald das Fangraster des Editors eingeschalten wird.
- `raster_groesse` speichert eine von zwei möglichen Grössen (s. Abb. 8-7).

Abb. 8-7

Konstanten für Zeichensätze und Raster

```
#define     font_helvetica_14       0
            font_helvetica_18       1
            font_helvetica_24       2

#define     raster_klein            1
            raster_gross            2
```

- `datei_gespeichert` gibt an, ob eine neueste Version des Objekts gespeichert ist. Diese Variable wird in der aktuellen Programmversion nicht verwendet.
- Auch `undo_moeglich` wird nicht verwendet. Das Zurücknehmen einer bereits ausgeführten Funktion ist nur möglich, wenn man mit `Edit:Loeschen` ein Objekt gelöscht hat und dieses wieder sichtbar machen möchte.
- `ea_namen_ausgeben` wird durch `Optionen:E/A-Namen` aktiviert bzw. deaktiviert. Ist diese Variable auf TRUE gesetzt, werden die Namen der E/A-Objekte mit ausgegeben.
- in `widgets_erzeugt` wird gespeichert, ob die Funktion `starte_xgap_editor` bereits einmal aufgerufen wurde und damit die Widgets erzeugt sind.
- `editor_ein` bleibt `TRUE`, solange Events für den Editor verarbeitet werden.
- in `button_window` und `ausgabe_window` werden Xlib-Fenster-Adressen gespeichert.
- Die `pixmap` ist ein unsichtbarer Hintergrundspeicher im Server. Sie hat die gleiche Größe und Tiefe (Anzahl Farbebenen) wie das `ausgabe_window`. Damit kann man die Objekte in die Pixmap ausgeben, bevor man sie in das sichtbare Fenster kopiert. Dies vermeidet ein Flackern der Ausgabefläche. Außerdem wird diese Pixmap beim Verschieben von Objekten verwendet.

8.5.2 objekt_typ

Abb. 8-8

Struktur objekt_typ

```
typedef struct {
            char            *dateiname;
            Bool            objekt_vorhanden;
            XPoint          min;
            XPoint          max;
            segment_typ     *aktives_segment;
            ea_typ          *aktives_ea;
            Bool            bereich_aktiviert;
            int             bereich_x, bereich_y;
            int             bereich_breite;
            int             bereich_hoehe;
        } objekt_typ;

objekt_typ  objekt;
```

In dieser Variable (s. Abb. 8-8) sind alle Daten gespeichert, die sowohl die Segmente als auch die E/A-Objekte betreffen. Dazu gehören:

- `dateiname`, er gibt den VMS-Dateinamen an, mit dem das Objekt gespeichert wird.
- `objekt_vorhanden` wird auf `TRUE` gesetzt, sobald ein Segment oder E/A-Objekt gezeichnet wurde. Erst danach ist `Datei:Speichern` aktiv, und das Objekt kann gespeichert und (neu) ausgegeben werden.
- `min` und `max` sind Koordinaten der linken oberen und rechten unteren Ecke des Gesamtobjekts, d.h. aller Segmente und E/A-Objekte (`XPoint` ist eine Struktur mit x- und y-Komponente). Die Berechnung erfolgt vor jedem Speichern des Objekts neu. Aus `min` und `max` wird die Breite und Höhe berechnet und mit den Objektdaten gespeichert. Im Anwendungsteil werden die Objektdaten relativ zu `min` umgerechnet. Beim Verschieben des Objekts muß dann nur die `min`-Koordinate geändert werden.
- in `aktives_segment` und `aktives_ea` werden die Adressen von Unterobjekten gespeichert, wenn eines selektiert ist.
- `bereich_aktiviert` wird gesetzt, sobald ein Objekt getroffen und damit aktiviert wurde.
- in `bereich_x`, `bereich_y`, `bereich_breite` und `bereich_hoehe` werden die Umrisskoordinaten der aktivierten Objekte gespeichert.

8.6 Zeichnen der Grundelemente und E/A-Objekte

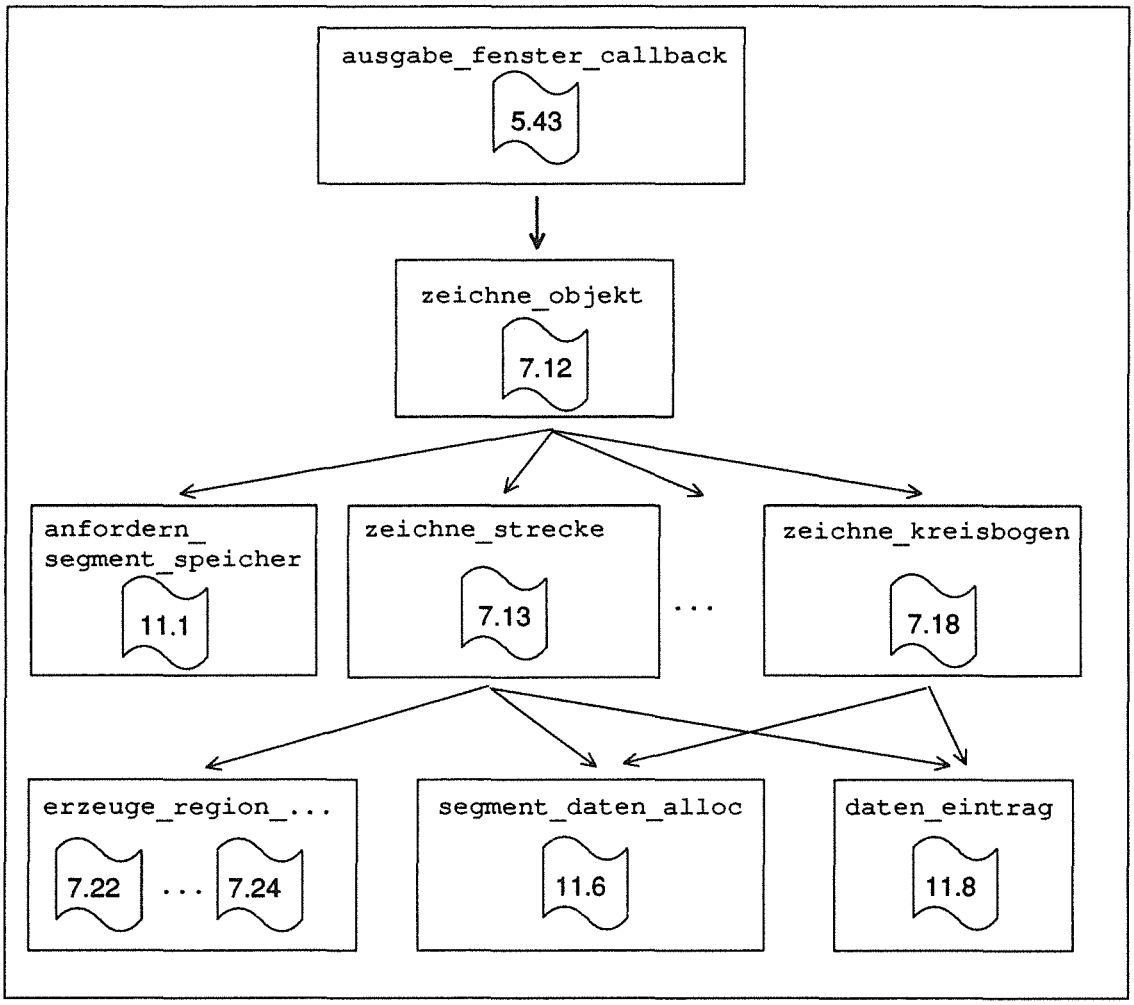


Abb. 8-9

Ausschnitt aus Funktionshierarchie zum Zeichnen von Objekten

Alle Segmente werden interaktiv erstellt, d.h. während der Benutzer den linken Mausbutton drückt und die Maus verschiebt, verändert sich die Größe des zu zeichnenden Objekts. Der Startpunkt des Objekts ist der Punkt der Ausgabefläche, an dem der Benutzer das Zeichnen beginnt. Dieser Punkt ist während des Zeichnens fest.
Nach Loslassen des Mausbuttons wird das Objekt endgültig gezeichnet (vorher ist es schwarz gestrichelt). Jede der in den folgenden Abschnitten beschriebenen Funktionen hat eine `while`-Schleife (`zeichne_polyline` und `zeichne_kreisbogen` haben zwei), die das interaktive Zeichnen ermöglicht. Sie funktionieren immer nach dem gleichen Prinzip:

- Zunächst werden die neuen Koordinatenwerte ermittelt (`event.motion...`).
- Falls das Objekt schon einmal gezeichnet wurde, wird die alte Zeichnung gelöscht. Dies wird ermöglicht, indem man das Objekt mit einer XOR-Verknüpfung im graphischen Kontext ausgibt. Verwendet man die XOR-Verknüpfung ein zweites Mal, wird das Element wieder gelöscht (und evtl. darunterliegende Objekte sind wieder im Originalzustand).
- Danach wird das Objekt mit den neuen Koordinaten gezeichnet, die neuen Koordinaten werden zu den alten.

- Es wird geprüft, ob der Benutzer die Maustaste losgelassen hat.

Nach der `while`-Schleife wird in jeder Zeichenfunktion getestet, ob die Anfangs- und Endkoordinaten mehr als fünf Pixel voneinander entfernt sind. Wenn nicht, wird davon ausgegangen, daß die Maustaste versehentlich gedrückt wurde und die Daten werden nicht gespeichert. Falls die Bedingung erfüllt ist,

- wird das Objekt farbig ausgegeben,
- wird Speicherplatz für die Koordinatenwerte angefordert, (`segment_daten_alloc`),
- werden die Attribute gespeichert (Modus, Umriss- u. Füllfarbe, Kantenbreite und Strichart)
- und die Koordinaten gespeichert (`speichere_segment_daten_auto`).

Für Strecke, Quadrat und Rechteck wird eine Region erzeugt.

In den folgenden Funktionsbeschreibungen werden nur noch Besonderheiten jeder Funktion erläutert. Vor den Zeichenfunktionen wird die Funktion beschrieben, die diese startet (s. Abb. 8-9).

8.6.1 zeichne_objekt

In dieser Funktion wird der Startpunkt bestimmt, je nach Editormodus eine Zeichenfunktion aufgerufen und danach Speicher für das nächste Segment angefordert (`erzeuge_neues_segment`). Damit steht zu Beginn dieser Funktion immer ein leerer Speicherplatz zur Verfügung, und das letzte Element der Segment- und E/A-Liste ist leer (d.h. `modus == kein_wert`). Mit dieser *eins-mehr*-Technik kann ohne Zähler das Ende der Liste festgestellt werden.
Die Funktion `editor_objekt_vorhanden` aktiviert den Pulldown-Menü-Eintrag `Speichern` und erlaubt durch das Setzen von `objekt.objekt_vorhanden`, Objekte auszugeben.

8.6.2 zeichne_strecke

Es werden in der `daten`-Liste vier Absolutwerte gespeichert; jeweils ein x- und y-Wert für Anfangs- und Endpunkt. Falls die beiden y-Koordinaten verschiedene Werte haben, wird eine Region erzeugt (`erzeuge_region_strecke`). In dieser Funktion werden die bereits gespeicherten Koordinatenwerte in eine Punktliste eingetragen und aus dieser Liste eine Region erzeugt. Falls bereits eine Region vorhanden ist, wird die alte vorher gelöscht. Wenn die y-Koordinaten gleich sind, d.h. die Strecke waagrecht ist, kann keine Region erzeugt werden, da die Xlib-Funktion `XPolygonRegion` sonst abstürzt. Dieser Fall wird beim Suchen von Segmenten gesondert behandelt.

8.6.3 zeichne_polyline

Es werden mehrere aneinanderhängende Strecken gezeichnet, bis der Benutzer zweimal kurz hintereinander die Maustaste drückt (Doppelklick). Dieser Doppelklick wird durch den Zeitunterschied zwischen dem ersten und zweiten Drücken der Maustaste ermittelt (der Unterschied muß kleiner als 500 Millisekunden sein).
Die Koordinatenwerte werden während des Zeichnens in einer Punktliste (`punkte_abs`) in Absolutwerten gespeichert. Falls der Abstand zwischen Anfangs- und

Endpunkt kleiner gleich 10 Pixel ist, wird Endpunkt gleich Anfangspunkt gesetzt und damit das Polygon geschlossen. Nun kann, falls als Füllfarbe eine Farbe ungleich Weiß gewählt wurde, die Polyline gefüllt werden. Ob die Polyline geschlossen oder offen ist, wird als Konstante (`polyline_offen` oder `polyline_geschlossen`) in der Daten-Liste an erster Position gespeichert.
Die Maximalanzahl der möglichen Teilstrecken einer Polyline ist in der Konstanten `max_anzahl_poly_punkte` gespeichert.
Vor dem Speichern werden alle Daten, bis auf den ersten Punkt, mit `umrechnung_absolut_relativ` in relative Werte umgerechnet. Beim Verschieben des Segments muß dann nur die erste Koordinate geändert werden.

8.6.4 zeichne_rechteck

Geschlossene Elemente (wie z.B. Rechtecke) werden nur gefüllt, wenn die Füllfarbe nicht Weiß ist. Damit kann in der Anwendung ein großes (mit der Farbe Weiß gefülltes) Objekt ein kleineres enthalten (s. Anhang B).
Falls die Füllfarbe ungleich Weiß ist, sind die Startkoordinaten der zu füllenden Fläche unterschiedlich, je nach Breite der Kanten. Falls die Kantenbreite mehr als ein Pixel beträgt, ist die zu füllende Fläche kleiner und wird entsprechend verringert.

8.6.5 zeichne_quadrat

Besonderheiten wie bei zeichne_rechteck.

8.6.6 zeichne_kreis

Auch hier Besonderheiten wie bei zeichne_rechteck. Außerdem muß hier, mit den neuen Cursorkoordinaten, jedesmal der Durchmesser neu berechnet werden und damit die linke obere Ecke des umgebenden Quadrats. Im Xlib-Aufruf zum Zeichnen eines Kreises (`XDrawArc`) wird nicht der Mittelpunkt des Kreises angegeben, sondern diese linke obere Ecke. Der zu zeichnende Bogen wird nicht in Grad angegeben, sondern in Grad * 64. Ein Vollkreis hat somit den Wert 360 * 64 = 23040.

8.6.7 zeichne_kreisbogen

Das Zeichnen eines Kreisbogens erfolgt in zwei Phasen. In der ersten Phase wird der Mittelpunkt des Kreises, auf dem der Bogen liegt, bestimmt und der Startpunkt des Kreisbogens (damit der Radius). Diese Phase entspricht dem Zeichnen einer Strecke.
Mit der Funktion `ermittle_winkel` wird der Winkel eines Punktes (hier der aktuellen Cursorposition) innerhalb eines x-y-Koordinatensystems bestimmt, dessen Ursprung der Mittelpunkt eines Kreises ist. Der Winkel wird, ausgehend von der positiven x-Achse (y=0), gegen den Uhrzeigersinn ermittelt.
Der Winkel wird mit der Arcustangens-Funktion bestimmt, und es wird unterschieden, in welchem Quadrant der Punkt liegt, damit die Umwandlung Radiant in Grad richtig erfolgen kann (s. Abb. 8-10).
In der Xlib-Funktion zum Zeichnen eines Kreisbogens (`XDrawArc`) gibt man zwei Winkel an. Der erste ist der Winkel, bei dem der Bogen beginnt (`start_winkel`), der zweite ist der Winkel, der den Bogen bestimmt, ausgehend vom Startwinkel. Der Startwinkel beginnt auf der positiven x-Achse (y=0). Positive Werte für den zweiten Winkel bedeuten ein Zeichnen entgegen des Uhrzeigersinns.

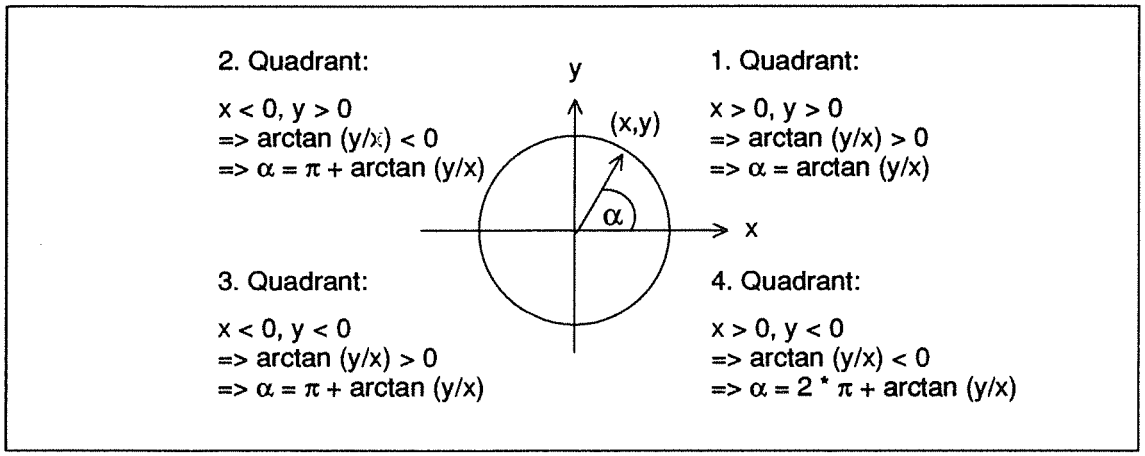


Abb. 8-10
Winkelberechnung Kreisbogen

Ein Kreisbogen kann sowohl im Uhrzeigersinn als auch entgegen erstellt werden. Deshalb wird zu Beginn, wenn der Bogenwinkel (der hier von der positiven x-Achse ausgeht) noch nahe am Startwinkel liegt, die Richtung des Bogens festgestellt:

```
if (abs (bogen_winkel_neu - start_winkel) <= 300)
{
    if (bogen_winkel_neu < start_winkel)
        richtung = uhrzeigersinn
    else
        richtung = gegenuhrzeigersinn;
}
```

Mit dieser Richtung kann dann geprüft werden, ob der Kreisbogen die x-Achse durchläuft. Sobald sich Richtung und Bogenwinkel widersprechen, z.B. die Richtung im Uhrzeigersinn verläuft und der Bogenwinkel größer als der Startwinkel ist, wird daraus ein Durchgang durch die x-Achse geschlossen (s. Abb. 8-11).

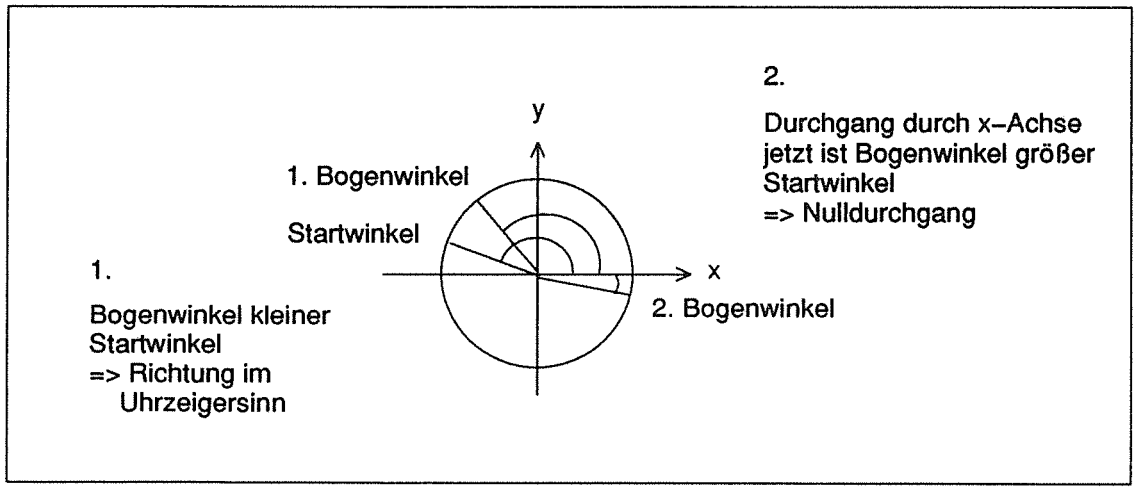


Abb. 8-11
Erstellung Kreisbogen

Diese Unterscheidung wird genauso beim Löschen des alten Bogens gebraucht wie bei der endgültigen Ausgabe in Farbe. Die Richtung wird mit den weiteren Daten gespeichert, um auch bei einer erneuten Ausgabe den Sonderfall feststellen zu können.

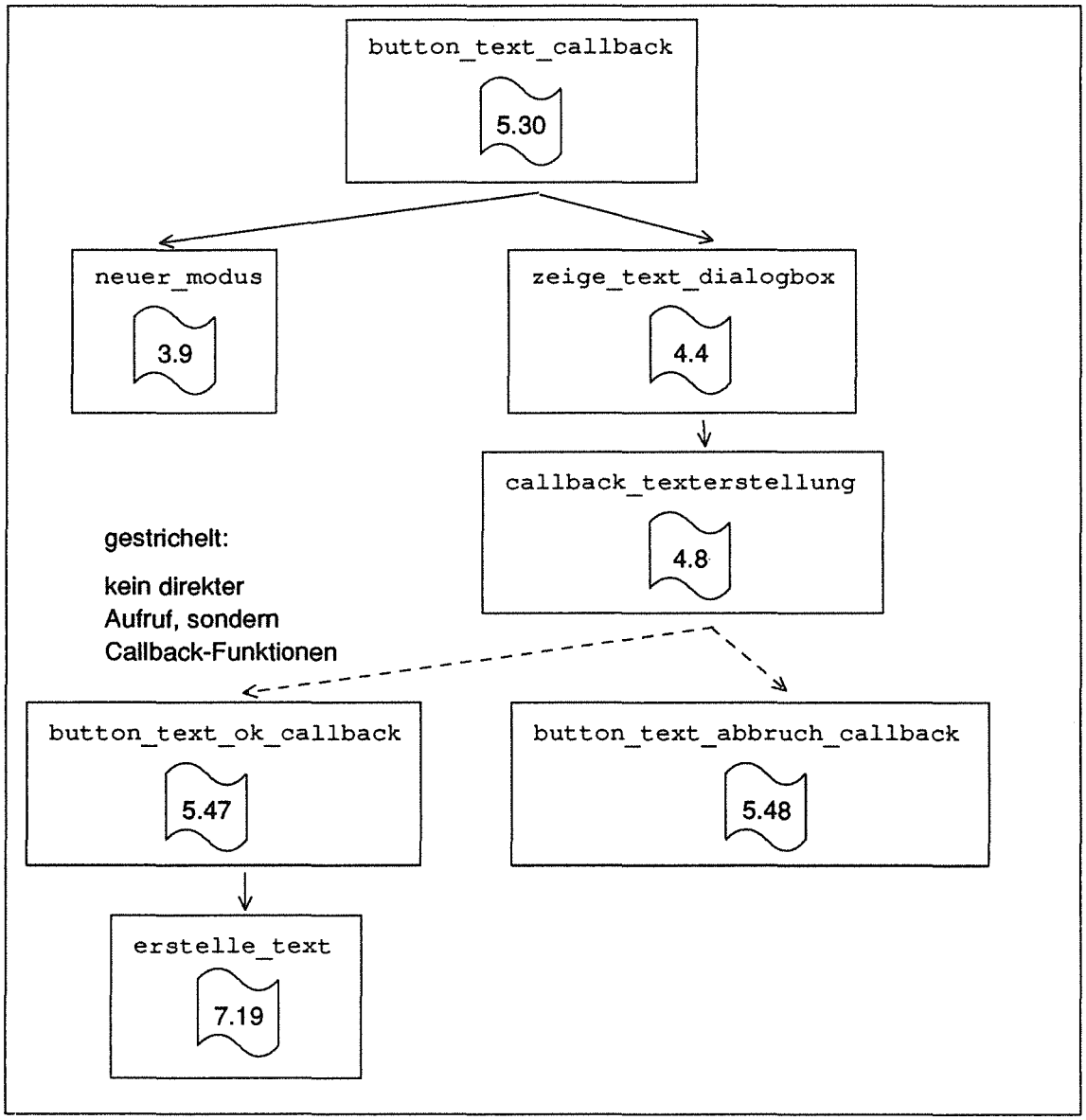
8.6.8 Eingabe von Text

Da Text und E/A-Objekte mit Hilfe von Dialogboxen erstellt werden, funktioniert der Aufruf in anderer Weise als bei den Zeichenfunktionen (und die Funktionsnamen sind geändert, anstatt `zeichne_...` heißen sie `erstelle_...`).

Mit den Callback-Funktionen der Button-Widgets werden die Dialogboxen angezeigt (erzeugt wurden sie in `erzeuge_dialogboxen`). Sobald der Benutzer den `OK`-Button drückt, werden die Ausgabefunktionen aufgerufen; beim Drücken des `Abbruch`-Buttons verschwindet die Dialogbox vom Bildschirm, ohne daß die Eingaben gespeichert werden.

Abb. 8-12

Ausschnitt aus der Funktionshierarchie für Text-Segmente



Nun zum Ablauf der Texteingabe. Wenn der mit dem Buchstaben 'A' gekennzeichnete Button im Kontrollfeld geklickt wird, wird die damit verbundene Callback-Funktion aufgerufen (`button_text_callback`, s. Abb. 8-12). Diese ändert den Zeichenmodus (`neuer_modus`) und ruft `zeige_text_dialogbox` auf. Mit einer weiteren Funktion (`callback_texterstellung`) werden die Callback-Funktionen für die Button-Widgets `text_dialog.ok` und `text_dialog.abbruch` diesen zu-

gewiesen. Die Callback-Funktionen werden hier nicht gleich bei der Erzeugung des Widgets zugewiesen, damit das Widget in verschiedenen Situationen gebraucht werden kann, jeweils mit anderen Callback-Funktionen. Außerdem wird in `zeige_text_dialogbox` die Box für die Eingabe von Text angezeigt:

```
XtManageChild (text_dialog.box).
```

Danach ist diese Funktion beendet, es werden jetzt, solange in der Dialogbox kein Button gedrückt und damit die Box gelöscht wird, nur Events innerhalb dieses Fensters verarbeitet (*modaler* Modus).
Sobald der Benutzer den Abbruch-Button drückt, verschwindet die Box, und der Editor befindet sich im Auswahlmodus. Wenn er `OK` drückt und nichts eingegeben hat, wird eine Meldungsbox (Widget: `meldungs_dialog.box`) ausgegeben, die mit zugehörigem `OK`-Button bestätigt werden muß. Sobald Text eingegeben und der `OK`-Button gedrückt ist, wird die Funktion

```
erstelle_text
```

aufgerufen. In dieser Funktion wird zunächst die Textdialogbox gelöscht, danach die Attribute gesetzt (bei Text ist nur Umrissfarbe von Bedeutung). Mit

```
text = DwtSTextGetString (Text-Widget)
```

wird ermittelt, was in das Text-Widget eingegeben wurde. Danach wird der Text im Widget gelöscht. Damit später abgefragt werden kann, wie breit und hoch der Text ist, wird dies in Abhängigkeit vom gewählten Zeichensatz festgestellt. Nach Anforderung von Speicherplatz wird die Nummer des Zeichensatzes gespeichert, danach eine vorgegebene Position, die Breite und die Höhe. Die Buchstaben werden in Integerwerte umgerechnet und gespeichert.
Zum Schluß wird Speicher für das nächste Segment angefordert und der Modus in `modus_auswahl` geändert. Danach kann der Benutzer den Text an die von ihm gewünschte Position verschieben.

8.6.9 Erstellung E/A-Objekte

Sobald der E/A-Button gedrückt wird, wird die Funktion `button_ea_callback` aufgerufen (s. Abb. 8-13). In dieser Funktion wird das Widget `ea_dialog.box` angezeigt. Diese Dialogbox enthält mehrere Eingabefelder und Toggle-Buttons, zum Beenden einen `OK`- und `Abbruch`-Button. Alle Buttons sind mit festen Callback-Funktionen verbunden (zugewiesen in `erzeuge_dialog_boxen`). Eine Änderung von Typ (Eingang/Ausgang), Richtung oder Größe führt zum Aufruf der in der obigen Abb. gezeigten Funktionen. In der Datenstruktur `ea_dialog` wird die neue Auswahl gespeichert, mit der Funktion `neue_ea_wahl` eine andere Pixmap angezeigt (um die aktuelle Auswahl zu verdeutlichen). Sobald `OK` gedrückt wird, testet die Funktion `button_ea_ok_callback`, ob im Namensfeld etwas eingegeben wurde. Wenn nicht, wird, wie beim Textdialog, eine Meldung "Bitte Namen eingeben oder <Abbruch>" ausgegeben, die bestätigt werden muß. Sobald im Namensfeld etwas eingegeben und OK gedrückt ist, wird die Funktion

```
erstelle_ea
```

aufgerufen. Die Box wird vom Bildschirm gelöscht, die eingegebenen Daten gelesen (Name, Text, Typ, Richtung und Größe) und die Attribute gespeichert (nur Füllfarbe und Umrissfarbe). Mit `kopiere_ea_struktur` wird das gewünschte Objekt aus

einem der in `init_var` vordefinierten Objekte kopiert. Danach wird neuer Speicherplatz angefordert und der Modus geändert, damit das Objekt positioniert werden kann.

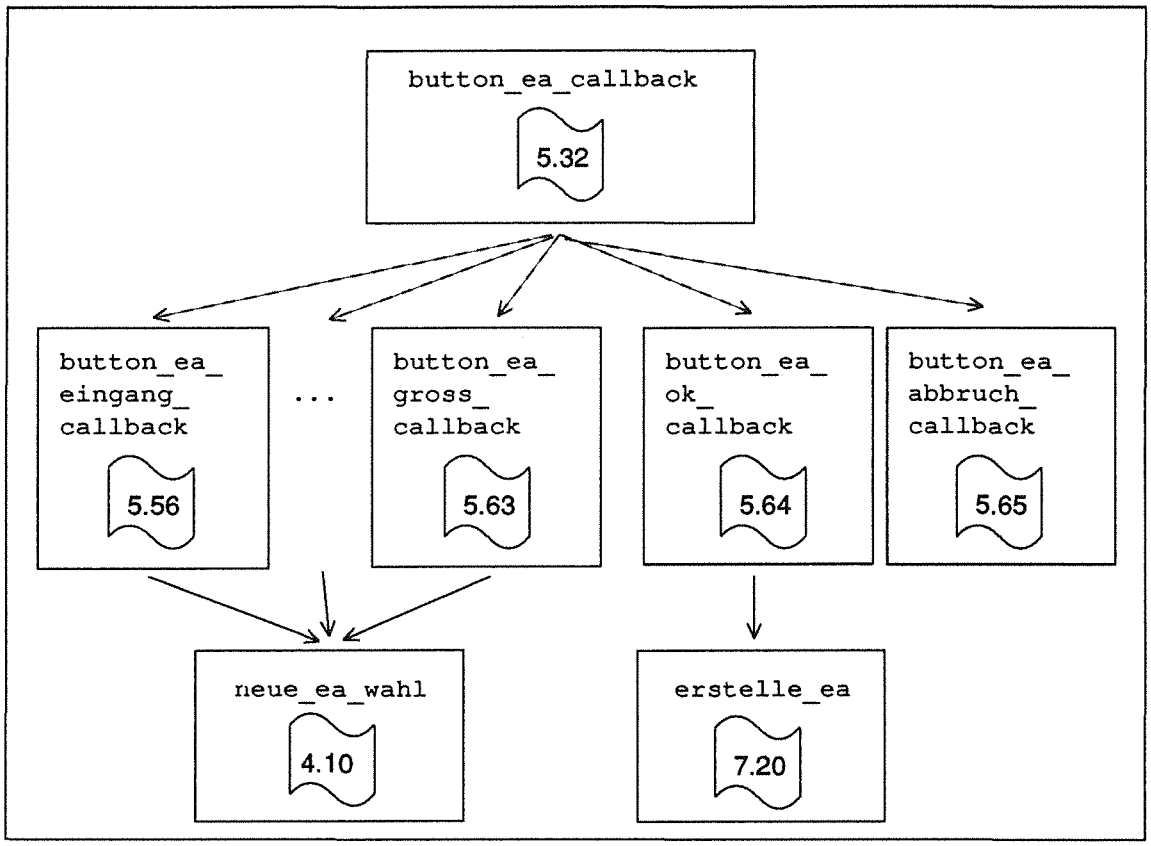


Abb. 8-13

Ausschnitt aus der Funktionshierarchie für E/A-Objekte

8.7 Ausgabe der Objekte

Sobald ein Objekt-Attribut geändert, ein Objekt verschoben oder ein Expose-Event gesendet wurde, wird die Funktion `ausgabe_editor_objekt` (Modul `EDIT_OBJ`) mit ihren Unterfunktionen aufgerufen (s. Abb. 8-14).

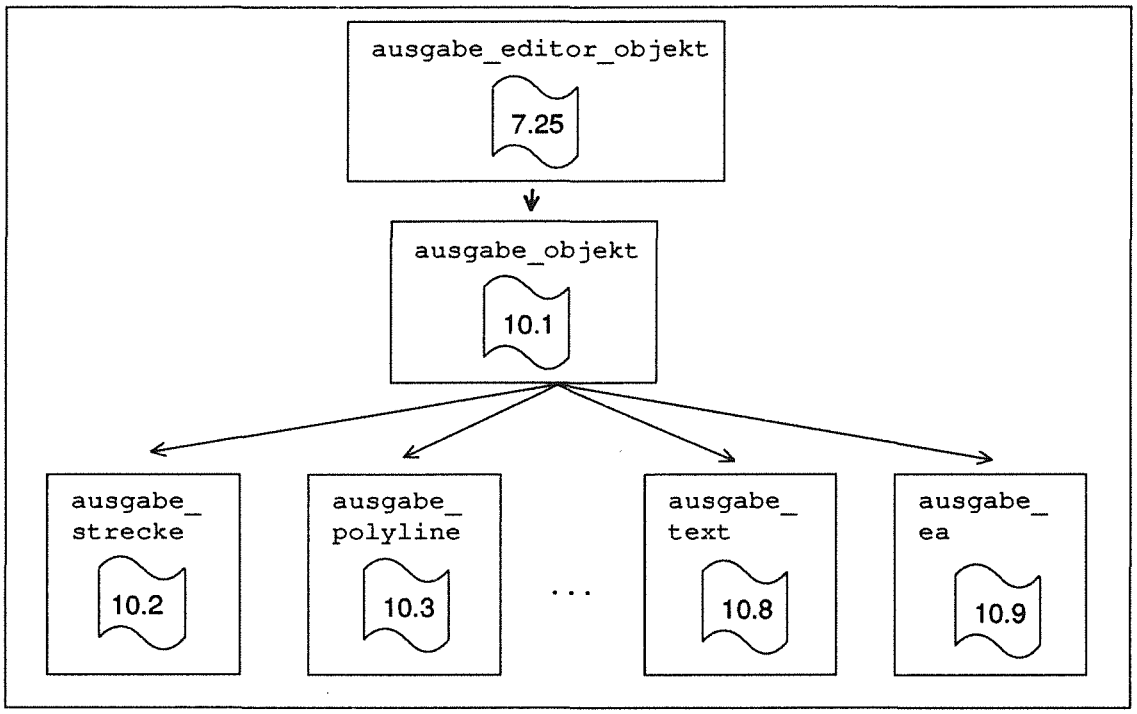


Abb. 8-14

Ausschnitt aus der Funktionshierarchie zur Objekt-Ausgabe

Die Funktion `ausgabe_editor_objekt` ermittelt das Ausgabefenster und den Beginn der Objektlisten, um diese Daten an `ausgabe_objekt` zu übergeben. In `ausgabe_objekt` werden zunächst, getrennt nach Modus, die Segmente ausgegeben, danach die E/A-Objekte. Die einzelnen Ausgabefunktionen zeichnen das Bild in den Hintergrundspeicher (`editor.pixmap`); erst nachdem alle Objekte ausgegeben sind, werden sie mit dem `XCopyArea`-Befehl in das Ausgabefenster kopiert.
Die einzelnen Ausgabefunktionen lesen die Koordinatenwerte aus der `daten`-Liste, ändern, je nach Attributwerten des auszugebenden Objekts, den graphischen Kontext (`aendere_gc_segment`) und geben das Objekt aus.
In `ausgabe_polyline` werden die Koordinatenwerte erst in eine Punktliste kopiert, bevor mit `XDrawLines` alle Strecken auf einmal ausgegeben werden.
Bei geschlossenen Polylines, Rechtecken, Quadraten und Kreisen werden die Objekte, falls die Füllfarbe ungleich Weiß ist, abhängig von der Kantenbreite gefüllt (`XFillRectangle`).
In `ausgabe_text` werden die als Integerwerte gespeicherten Buchstaben wieder in `char`-Zeichen umgewandelt und in einer Zeichenkette zusammengefaßt. Mit `XSetFont` wird eine der drei möglichen Schriften dem graphischen Kontext `copy_gc` zugewiesen und mit `XDrawString` der Text ausgegeben.
In `ausgabe_ea` wird, nach dem Füllen und Ausgeben der Polyline, der Name des Objekts ausgegeben, falls der Schalter `editor.ea_namen_ausgeben` gleich `TRUE` ist.

8.8 Änderung der Grafikattribute

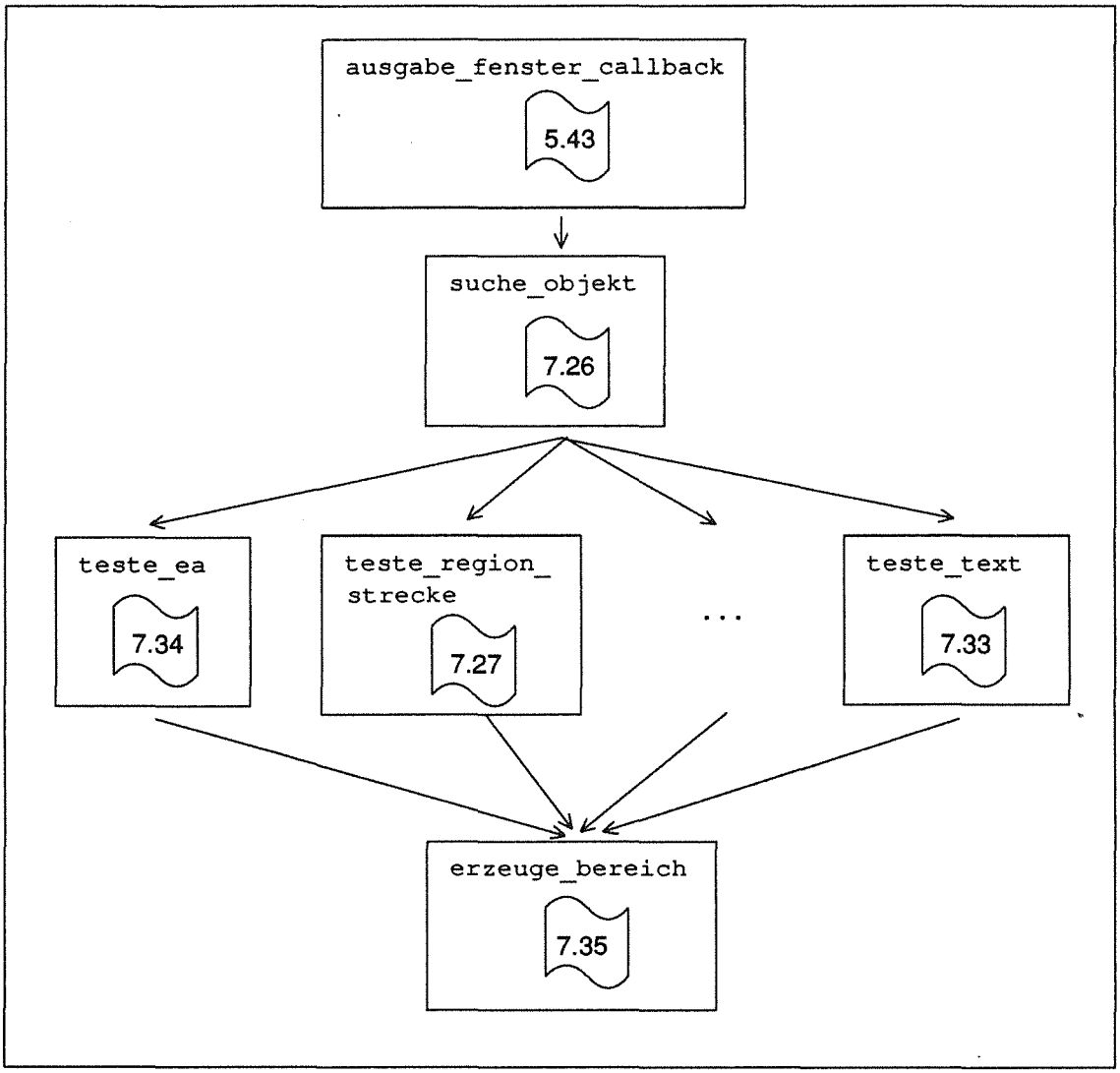


Abb. 8-15

Ausschnitt aus der Funktionshierarchie zur Manipulation von Objekten

Wenn der Auswahlmodus gewählt ist (damit ist im Kontrollfeld keiner der Zeichenelement-Buttons aktiviert) und der Benutzer im Ausgabefenster die linke Maustaste drückt, wird über die Funktion `ausgabe_fenster_callback` `suche_objekt` (s. Abb. 8-15) gestartet. Diese Funktion mit ihren Unterfunktionen testet, ob der Benutzer ein Objekt (Segment oder E/A-Objekt) getroffen hat und wenn ja, welches.
Eine bereits vorhandene Markierung (= Aktivierung eines Bereichs) wird gelöscht, in der dann folgenden ersten Schleife werden die E/A-Objekte überprüft. Dieser Test kommt zuerst, da die E/A-Objekte auf den Segmenten liegen.
Wenn nach dem Testen der E/A-Objekte noch kein Objekt getroffen wurde, werden in der nächsten `while`-Schleife die Segmente geprüft, getrennt nach Modus. Wenn in einer der Test-Funktionen ein Treffer bemerkt wurde, wird das Segment gestrichelt markiert (`markiere_bereich`), und es werden die Adresse und die Umrissdaten des markierten Segments gespeichert.

Wenn der Benutzer ein Segment getroffen hat und weiterhin die Maustaste drückt, wird die Funktion `verschiebe_bereich` aufgerufen. Mit

```
XCheckMaskEvent (disp, ButtonReleaseMask, &event)
```

wird geprüft, ob ein ButtonRelease-Event eingetroffen ist und damit festgestellt, ob die Maustaste losgelassen wurde.

8.8.1 teste_ea

Der Funktion werden die aktuelle Cursorposition und ein Zeiger auf das zu untersuchende Objekt übergeben. Es wird geprüft, ob diese Cursorkoordinaten in dem das Objekt umgebenden Bereich liegen. Der Bereich ist etwas größer als das Objekt, damit man nicht genau treffen muß. Wenn der Bereich getroffen ist, wird

- `erzeuge_bereich` aufgerufen, in dieser Funktion werden die Koordinaten in der `objekt`-Struktur gespeichert. Außerdem werden die Pulldown-Einträge `Kopieren` und `Löschen` im Menü `Edit` aktiviert, da jetzt ein Bereich vorhanden ist.
- in der `objekt`-Struktur die Adresse des getroffenen E/A-Objektes gespeichert. Damit festgestellt werden kann, daß ein E/A-Objekt und kein Segment aktiviert ist, wird `editor.aktives_segment` auf `NULL` gesetzt (bei getroffenen Segmenten wird `editor.aktives_ea` gleich `NULL`)

8.8.2 teste_region_strecke

Die Koordinaten der Strecke werden gelesen, dann der kleinste (x,y)-Wert festgestellt (um den Rahmen um die zwei in absoluten Koordinaten gespeicherten Punkte zeichnen zu können).
Falls die beiden y-Koordinaten verschiedene Werte haben, wird mit der Funktion `XRectInRegion` festgestellt, ob der Cursor auf der Strecke liegt. Der Cursor wird dabei auf ein 10x10-Pixel-Feld vergrößert, um nicht genau treffen zu müssen. Mit `XRectInRegion` kann überprüft werden, ob ein Rechteck (hier der vergrößerte Cursor) außerhalb, innerhalb oder auf der Grenze einer Region liegt. Erhält man als Ergebnis `RectanglePart`, liegt das Cursor-Quadrat zum Teil innerhalb der Region und damit auf der Strecke.

Wenn y1 und y2 gleich sind, findet eine Spezialbehandlung statt, da hierfür keine Region erzeugt werden kann. Es wird, wie bei den E/A-Objekten, ein Rechteck um die Strecke gelegt und getestet, ob der Cursor sich innerhalb dieses Bereichs befindet.

Wenn getroffen wurde, wird, mit der gleichen Funktion wie in `teste_ea`, ein Bereich erzeugt.

8.8.3 teste_region_polyline

Die Koordinatenwerte der Polyline-Segmente sind relativ zum ersten Punkt der Strekken gespeichert. Sie müssen deswegen erst in absolute Koordinaten umgewandelt werden. Dies geschieht mit folgendem Algorithmus:

absolut-Wert (n) = absolut-Wert (n-1) + relativ-Wert (n)

Gleichzeitig mit der Umwandlung werden die Koordinaten für das Rechteck ermittelt, das die Polyline einschließt. Damit wird festgestellt, ob der Cursor sich innerhalb die-

ses Rechtecks befindet.
Wenn der Cursor sich innerhalb befindet, wird dann jeweils mit zwei Punkten eine Region erzeugt und, wie in `teste_region_strecke`, überprüft, ob der Cursor auf dem Rand der Region ist. Dies wird wiederholt, bis alle Punkte getestet sind oder eine Strecke getroffen wurde. Falls zwei y-Koordinaten gleich sind, findet eine Ausnahmebehandlung statt (wie in `teste_region_strecke`).
Wenn eine der Polyline-Strecken getroffen wurde, wird, wie in `teste_ea`, ein Bereich erzeugt.

8.8.4 teste_region_strecke und teste_region_quadrat

Für diese Objekte sind immer Regionen gespeichert, sie können mit der `XRectInRegion`-Funktion getestet werden.
Es wird unterschieden, ob die Kanten ein Pixel, zwei oder drei Pixel breit sind. Wenn die Breite größer ist, muß der Bereich entsprechend vergrößert werden, damit die gesamte Fläche eingerahmt wird.

8.8.5 teste_kreis

Mit den Koordinaten der linken, oberen Ecke des den Kreis umgebenden Quadrats und dem Durchmesser werden Mittelpunkt und Radius berechnet. Mit Hilfe des euklidschen Abstandes wird bestimmt, ob der Cursor innerhalb eines Toleranzbereiches (+/-5) liegt.
Berechnung des Abstandes eines Punktes vom Koordinatenursprung:

$$a = \sqrt{x^2 + y^2}$$

bzw.

$$a^2 = x^2 + y^2$$

Um die Wurzelberechnung zu vermeiden, wird das Quadrat der Abstände verwendet.
In der Funktion wird dann das Quadrat des minimal und maximal erlaubten Radius berechnet (`radius_min_quad` ...). Danach das Quadrat des Abstandes der Cursorkoordinaten (`abstand_quad`). Diese Abstände werden miteinander verglichen und, wie beim Rechteck, je nach Kantenbreite ein Bereich erzeugt, falls `abstand_quad` innerhalb der tolerierten Werte liegt.

8.8.6 teste_kreisbogen

Der Algorithmus ist der gleiche wie beim Kreis. Erschwerend kommt hinzu, daß man zunächst feststellen muß, ob der Cursor innerhalb des Bogens ist (`in_bogen`). Dazu wird mit der Richtung des Kreisbogens sein Verlauf festgestellt und überprüft, ob der durch den Cursor bestimmte aktuelle Winkel innerhalb des Bogens liegt.
Wenn ja, findet die gleiche Überprüfung wie bei `teste_kreis` statt, und es wird, wenn getroffen, ein Bereich erzeugt.

8.8.7 teste_text

Nach dem Lesen der Koordinaten des den Text umgebenden Rechtecks wird getestet, ob der Cursor innerhalb dieses Rechtecks (plus einer Toleranz) liegt. Wenn ja, wird ein Bereich erzeugt.

8.8.8 verschiebe_bereich

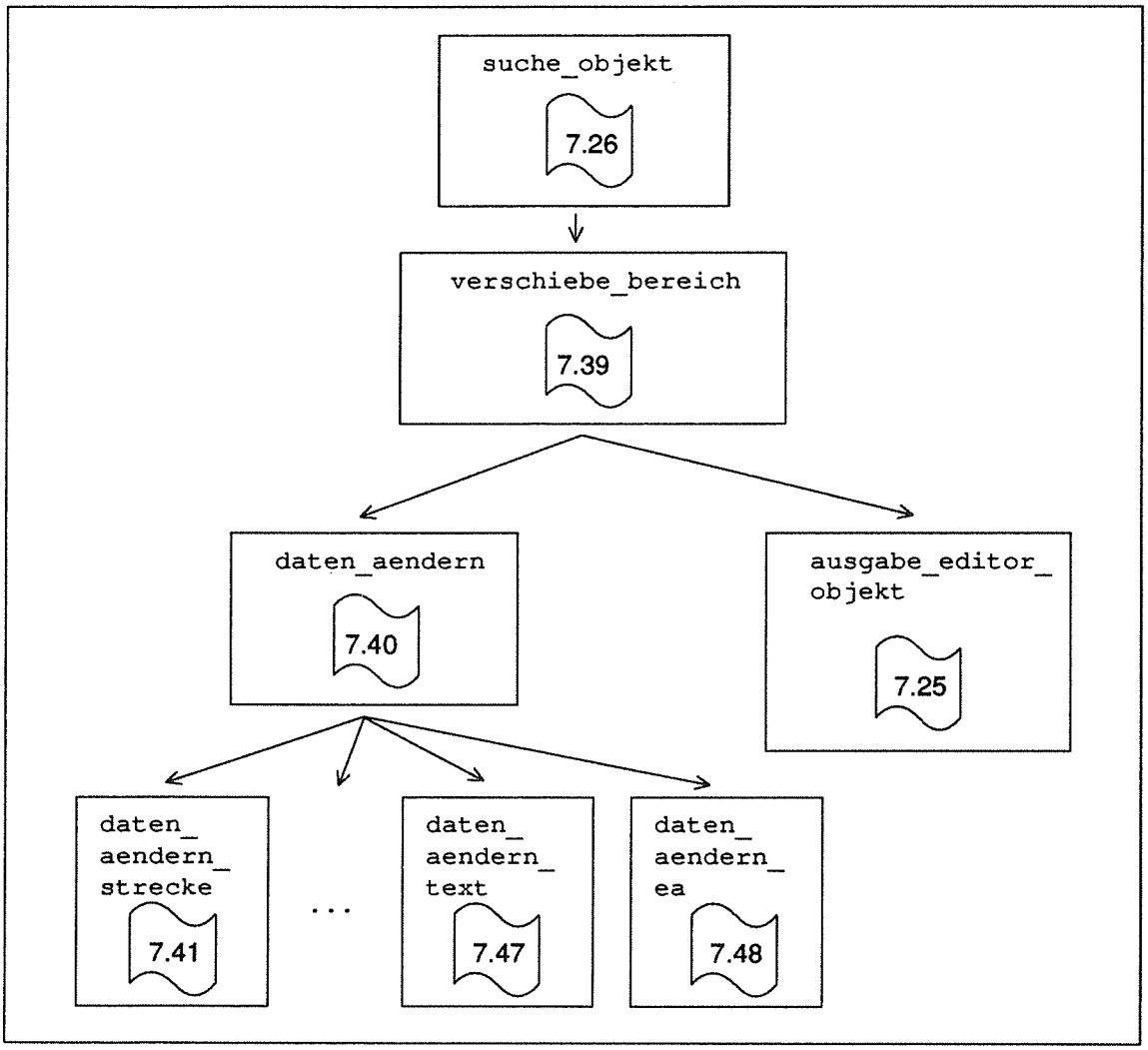


Abb. 8-16

Ausschnitt aus der Funktionshierarchie zum Verschieben von Segmenten

Mit dieser Funktion wird ein markierter Bereich, in dem sich ein Segment oder E/A-Objekt befindet, verschoben.
In `markiere_bereich` wurde die zu verschiebende Pixmap mit `XCopyArea` in einen Hintergrundspeicher kopiert.
Mit der gleichen Xlib-Anweisung wird in `verschiebe_bereich` (s. Abb. 8-16) die Pixmap an die neue Position im Ausgabefenster kopiert.
Dies geschieht in einer `while`-Schleife, die beendet wird, sobald die Maustaste losgelassen wird. Mit `XWindowEvent` wird der nächste Event abgefragt, um die neue Cursorposition feststellen zu können.
Im ersten Schleifendurchlauf (neu == TRUE) wird die erste gestrichelte Markierung gelöscht durch einen (zweiten) `XDrawRectangle`-Aufruf mit der XOR-

Verknüpfung. Außerdem wird der erste Bereich gelöscht (`XClearArea`). In allen weiteren Durchläufen wird mit `XCopyArea` und der `XOR`-Verknüpfung das zuvor Gezeichnete wieder gelöscht, bevor der neue Bereich gezeichnet wird.

Die jeweils neuen Koordinaten werden mit dem Algorithmus:

neue_koordinate = editor.bereichs_koordinate + cursor_koordinate
- anfangs_koordinate

berechnet.

Wenn die Schleife beendet ist, wird die Gesamtverschiebung festgestellt:

delta = cursor_aktuell - anfangs_koordinaten,

und diese Werte an `daten_aendern` übergeben.

8.8.9 daten_aendern

Zunächst wird festgestellt, ob die Daten eines Segments oder eines E/A-Objekts geändert werden sollen. Falls es sich um ein verschobenes Segment handelt, zeigt die Variable `objekt.aktives_segment` auf das zu verändernde Segment. Je nach Modus wird dann die entsprechende Änderungsroutine gestartet. In den Änderungsfunktionen wird auf die absoluten Koordinatenwerte zugegriffen und diese um den `delta`-Wert geändert. Bei Strecken, Rechtecken und Quadraten werden neue Regionen erzeugt (Ausnahmebehandlung in `daten_aendern_strecke` wie in `zeichne_strecke`).
In `daten_aendern_ea` werden neben dem linken oberen Punkt des umschließenden Rahmens auch der Startpunkt der Polyline, der Eingangs- und der Ausgangspunkt geändert.

Die Änderung von Attributen der markierten Objekte geschieht mit Callback-Funktionen, die von den Buttons des Kontrollfensters (Widgets innerhalb `auswahl_fenster.box`) aufgerufen werden. Werden die Buttons im Auswahl-Modus gedrückt, wird damit ein gewünschtes Attribut vorgewählt.
Zunächst die Änderung der Füllfarben. Die Callback-Funktionen rufen nur

8.8.10 neue_fuell_farbe

auf und übergeben die zugehörige Farbkonstante. In dieser Funktion (s. Abb. 8-17) wird der `editor`-Variablen die neu gewählte Farbe zugewiesen, danach die Farbkonstante in eine Xlib-Farbe umgewandelt (`ermittle_farbe`). Die Xlib-Farbe wird an das Label-Widget übergeben, um die neu gewählte Farbe anzuzeigen.
Falls ein Bereich markiert ist, wird auf den Attributwert `fuell_farbe` des aktiven Segments oder E/A-Objekts zugegriffen, dieser Wert aktualisiert und danach das gesamte Objekt neu ausgegeben (`ausgabe_editor_objekt`).

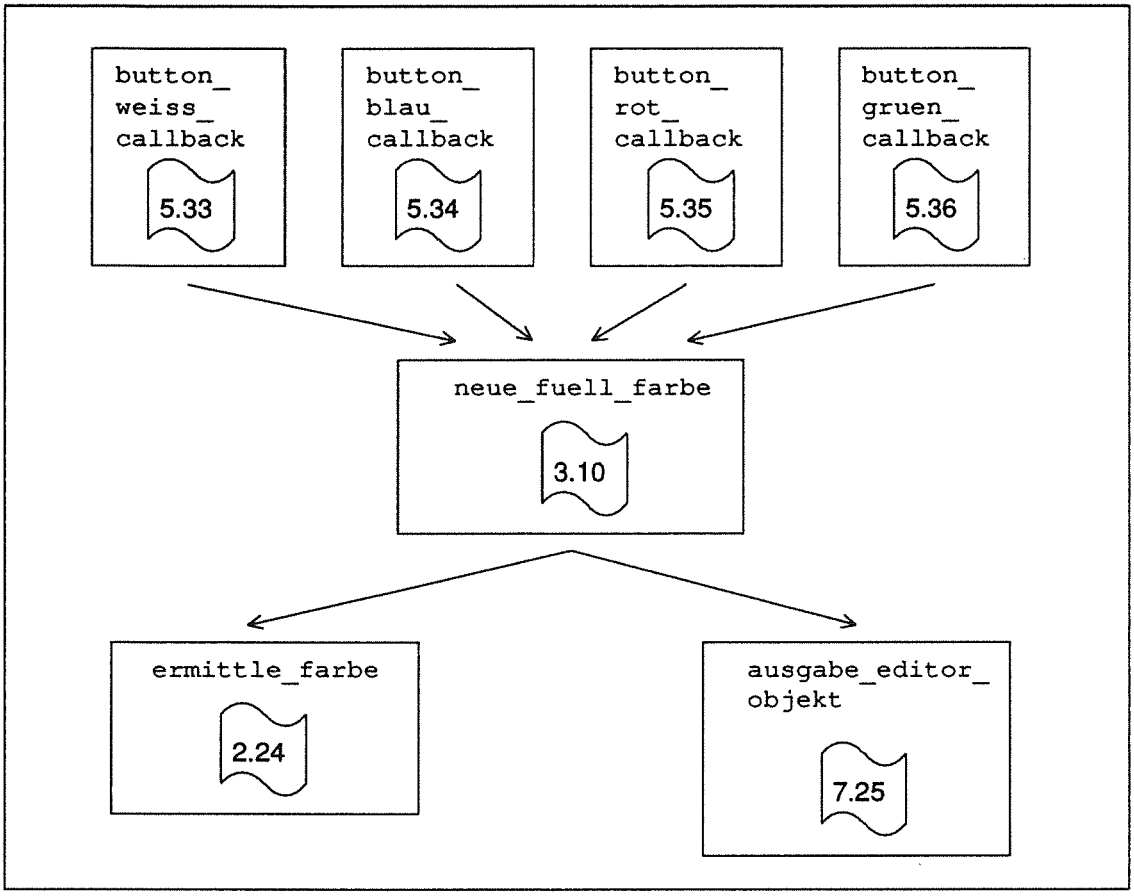


Abb. 8-17

Ausschnitt aus der Funktionshierarchie zum Ändern der Füllfarbe

8.8.11 neue_kantenbreite

wird von den drei Callback-Funktionen der Kantenbreite-Buttons aufgerufen (s. Abb. 8-18); übergeben wird die entsprechende Konstante.

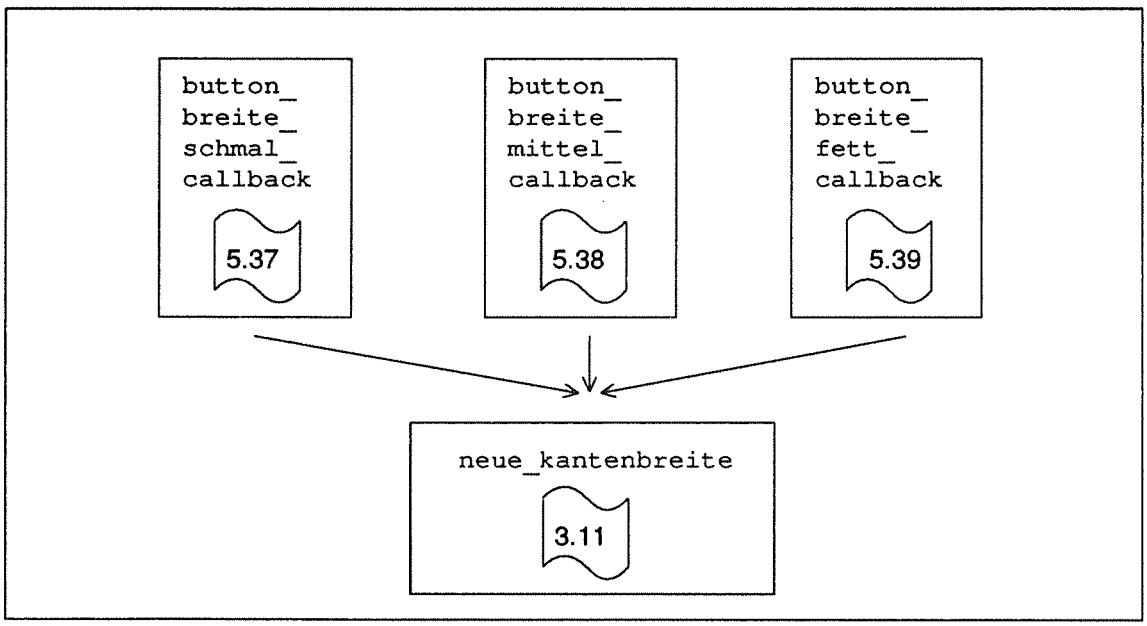


Abb. 8-18

Ausschnitt aus der Funktionshierarchie zum Ändern der Kantenbreite

In `neue_kantenbreite` wird zuerst der markierte Button deaktiviert, d.h. es wird eine andere Pixmap (`pix_na`) angezeigt.

Nachdem die Variable `editor.kanten_breite` geändert wurde, wird der soeben angeklickte Button aktiviert, d.h. es wird die inverse Pixmap ausgegeben. Falls ein Objekt markiert ist, wird, wie in `neue_fuell_farbe`, das entsprechende Attribut geändert.

8.8.12 neue_strichart

wird auch über Callback-Funktionen von Buttons (s. Abb. 8-19) aufgerufen, hier durch die drei untersten zur Auswahl der Strichart. Funktionalität wie `neue_kantenbreite`.

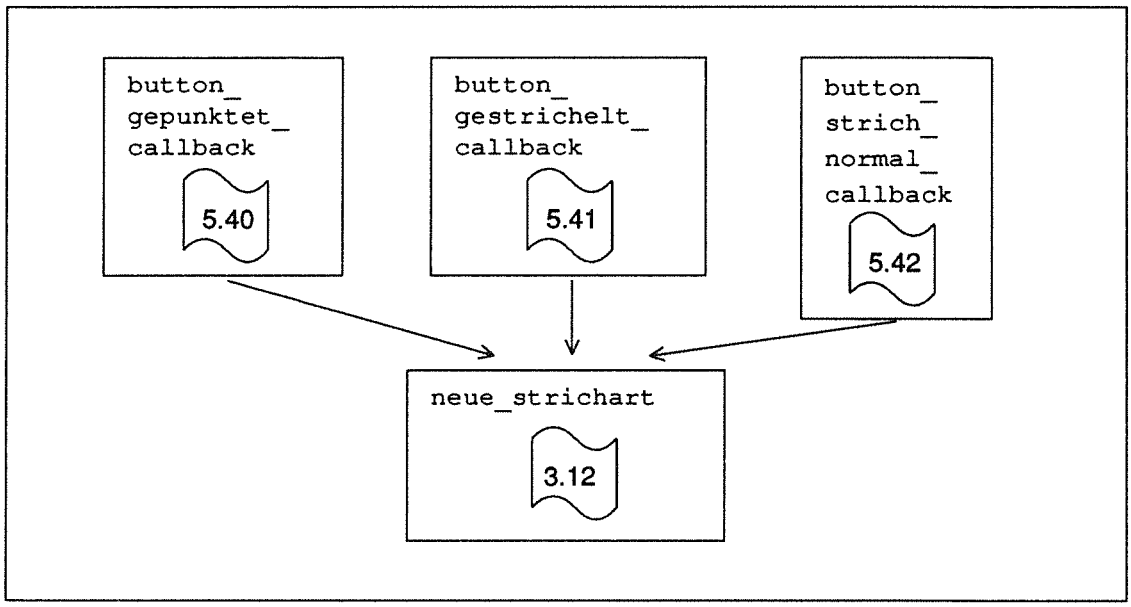


Abb. 8-19

Ausschnitt aus der Funktionshierarchie zum Ändern der Strichart

8.8.13 neue_kanten_farbe

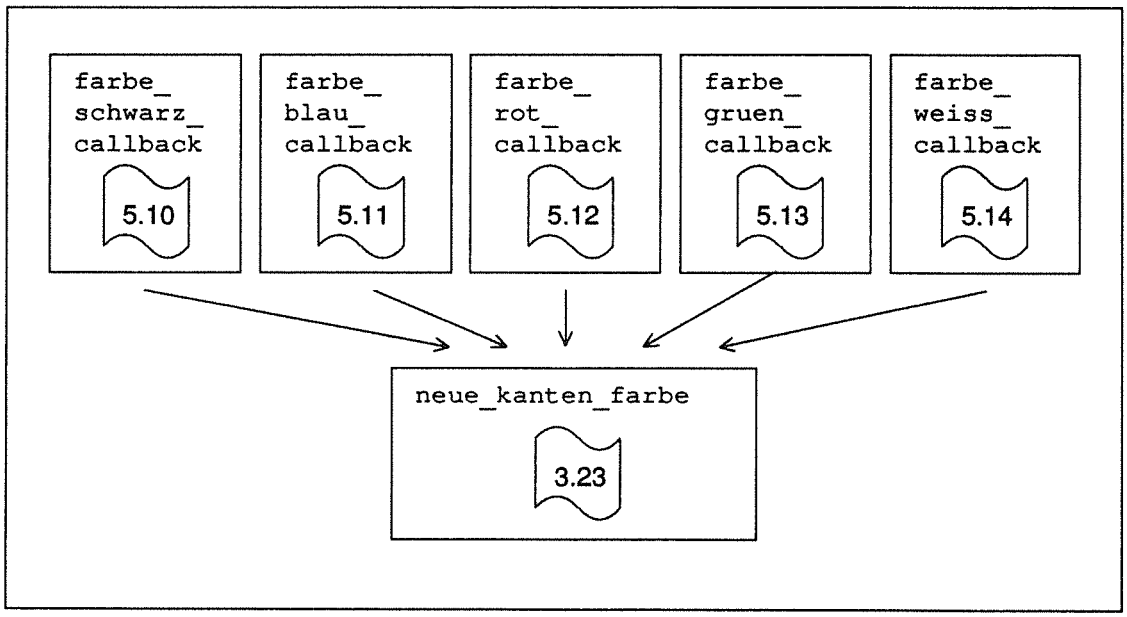


Abb. 8-20

Ausschnitt aus der Funktionshierarchie zum Ändern der Kantenfarbe

Die Kantenfarbe wird über das Pulldown-Menü `Farbe` geändert. Die Buttons dieses Menüs rufen `neue_kanten_farbe` (s. Abb. 8-20) auf. Die Funktion ist größtenteils die Gleiche wie `neue_fuell_farbe`. Im Pulldown-Menü ist der Button für die Farbe Weiß schwarz markiert.

8.8.14 ea_aenderung_callback

Um Attribute eines E/A-Objektes zu ändern, markiert man zuerst ein E/A-Objekt und drückt dann die mittlere Maustaste (`MB2`). Mit diesem Event wird `ea_anderung_callback` aufgerufen (s. Abb. 8-21); diese Funktion gibt eine Dialogbox zur Änderung aus.

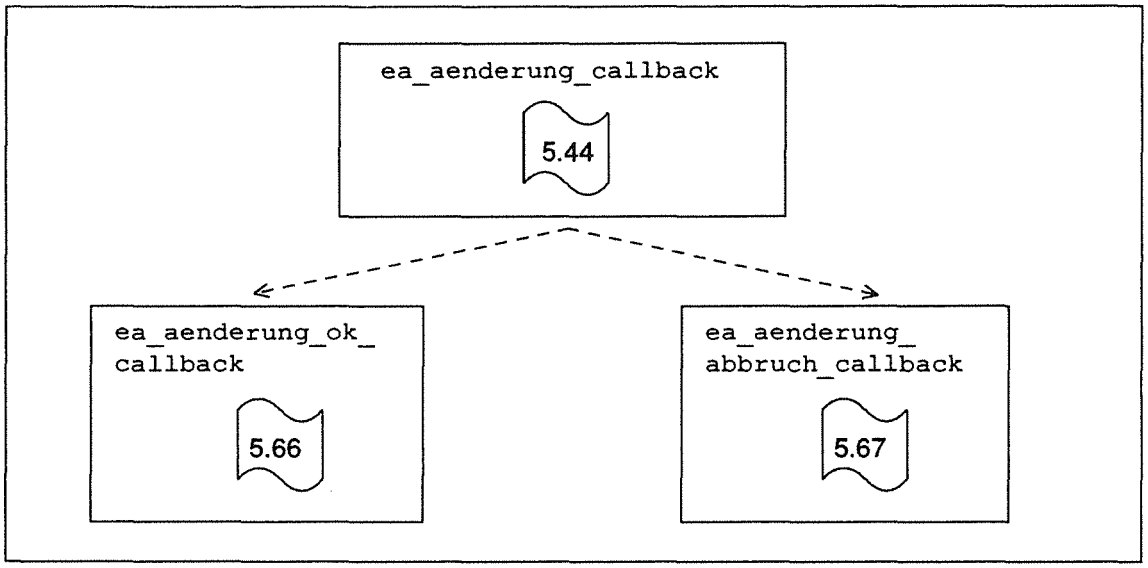


Abb. 8-21

Ausschnitt aus der Funktionshierarchie zum Ändern von E/A-Objekten

In dieser Box wird der gewählte Typ angezeigt (der Typ kann nicht geändert werden) und die Textfelder Name und Text.
Wenn man Namen bzw. Text geändert hat und den OK-Button drückt, wird `ea_aenderung_ok_callback` aufgerufen. Die Funktion prüft zuerst, ob die Länge des Textes im Namensfeld größer Null ist. Wenn die Namenszeile leer ist, wird, wie beim Erzeugen der E/A-Objekte, ein Fenster mit Warnhinweis ausgegeben. Wenn ein Name eingegeben wurde, werden die geänderten Texte gespeichert.
Bei Abbruch wird nur die Dialogbox vom Bildschirm gelöscht (`ea_aenderung_abbruch_callback`).

8.9 Kopieren und Löschen

Sobald ein Objekt aktiviert wurde, werden in `erzeuge_bereich` die Einträge `KOPIEREN` und `LOESCHEN` des Pulldown-Menüs `EDIT` aktiviert (Widgets: `pulldown_menu.edit_kopieren_eintrag, .._edit_loeschen_eintrag`). Die zuständigen Callback-Funktionen heißen `edit_kopieren_callback` und `edit_loeschen_callback`.
Das zuletzt gelöschte Objekt kann mit `EDIT:UNDO` wieder sichtbar gemacht werden (Widget: `pulldown_menu.edit_undo_eintrag`).

8.9.1 edit_kopieren_callback

Mit dieser Funktion können sowohl Segmente als auch E/A-Objekte kopiert werden. Um welche Objektart es sich handelt, wird durch die Zeiger `objekt.aktives_segment` bzw. `objekt.aktives_ea` festgestellt. Der entsprechende Zeiger muß ungleich `NULL` sein. Es werden danach die Attribute und Koordinatenwerte des neuen Objekts im letzten (noch leeren) Element der `segment`- oder `ea`-Liste gespeichert.

Ein kopiertes Segment wird neben dem Quellsegment ausgegeben, ein neues E/A-Objekt in der linken oberen Ecke. Die Objektkoordinaten eines neuen E/A-Objekts werden nicht aus dem Vorlageobjekt kopiert, sondern mit `kopiere_ea_struktur` aus den vordefinierten Objekten. Deswegen liegt ein kopiertes E/A-Objekt nicht neben dem Quellobjekt.

8.9.2 edit_loeschen_callback

Zu Beginn wird die Funktion `loesche_markiertes_objekt` aufgerufen. Diese Funktion durchläuft die Segment- und E/A-Listen und löscht ein früher logisch gelöschtes Objekt physikalisch aus der entsprechenden Liste. Das heißt, daß immer maximal ein Element in den Objekt-Listen logisch gelöscht sein kann. Falls das erste Objekt gelöscht werden soll, wird der Zeiger, der auf den Beginn der Liste zeigt, umgehängt. Wenn nicht, werden alle weiteren Objekte geprüft. Dazu werden zwei Zeiger auf die Listen gelesen und gespeichert (`erstes_.., zweites_..`). Beim zweiten gespeicherten Element wird geprüft, ob das Löschkennzeichen gesetzt ist.
Wenn die Kennung gesetzt ist, wird das Element dadurch entfernt, indem man dem Nachfolger des ersten Elementes die Adresse des Nachfolgers des zweiten Elements zuweist, zum Beispiel bei Segmenten:

```
erstes_segment->naechster = zweites_segment->naechster;
```

Danach wird in `edit_loeschen_callback` der Löschschalter des aktiven Objekts gesetzt. Mit `loesche_aktivierung` wird der aktivierte Bereich deaktiviert. Danach werden die Objekte neu ausgegeben, nun ohne das soeben (logisch) gelöschte Element.

8.9.3 edit_undo_callback

Die Funktion durchsucht die beiden Listen nach einem gelöschten Objekt, um den `geloescht`-Schalter auf `FALSE` zu setzen. Danach werden die Objekte neu ausgegeben und der `UNDO`-Eintrag deaktiviert.

8.10 Speichern und Laden

Dazu wird das Pulldown-Menü DATEI verwendet. Der Eintrag SPEICHERN wird erst aktiviert, wenn sich ein Objekt auf dem Bildschirm befindet. Die zum Eintrag SPEICHERN gehörende Callback-Funktion datei_speichern_callback ruft editor_objekt_speichern auf.

8.10.1 editor_objekt_speichern

Mit der Funktion suche_min_und_max wird das Minimum und Maximum aller Objekt-Koordinaten berechnet. Diese Funktion wurde geschrieben, um im Anwendungsteil Bezugskoordinaten zu haben (linke obere Ecke). Alle Koordinaten werden dort relativ zu dieser Koordinate umgerechnet. Beim Verschieben muß nur der Bezugspunkt geändert werden.
In suche_min_und_max werden der minimale und maximale Punkt eines Elements gelesen, danach mit neues_min_und_max geprüft, ob die übergebenen Koordinaten neue Minima oder Maxima sind. Das Ergebnis wird in der objekt-Variablen gespeichert. Für Polylines gibt es eine eigene Funktion (neues_min_und_max_in_liste); sie wandelt die Punkte erst in Absolutwerte um.

Aus objekt.min und objekt.max werden dann in editor_objekt_speichern Breite und Höhe bestimmt. Diese Werte werden gespeichert, um im Anwendungsteil den das Objekt umschließenden Bereich feststellen zu können.

Danach beginnt das Speichern; die Datei mit dem vorgewählten Dateinamen wird zum Schreiben geöffnet, danach

- wird eine Dateikennung geschrieben. Damit kann man beim Lesen feststellen, ob es sich um eine mit dem Editor gespeicherte Datei handelt, und man kann Dateiversionen unterscheiden (Konstante datei_kennung_editor).
- wird Dateispeicher dadurch reserviert, daß die Anzahl der Segmente und E/A-Objekte geschrieben wird (mit dem Anfangswert 0). Wenn alle Objekte gespeichert sind und die Anzahl der Objekte feststeht, wird der Dateizeiger an den Dateianfang zurückgesetzt, und die richtigen Werte werden geschrieben.
- werden Daten des Gesamtobjekts geschrieben, d.h. Minimum, Maximum, Breite und Höhe.
- wird objekt_speichern augerufen. Dort wird die Segmentliste durchlaufen, um Attribute und Daten zu speichern. Die Attribute werden zunächst in eine Struktur vom Typ segment_kopie kopiert, bevor sie zusammen in die Datei geschrieben werden.
  Außerdem werden alle E/A-Objekte gespeichert, auch hier werden die Attribute zuerst kopiert. Vor den Zeichenketten wird deren Länge gespeichert.
- werden zuletzt (wieder in editor_objekt_speichern), wie beschrieben, die Anzahl der Objekte gespeichert und die Datei geschlossen.

8.10.2 objekt_laden

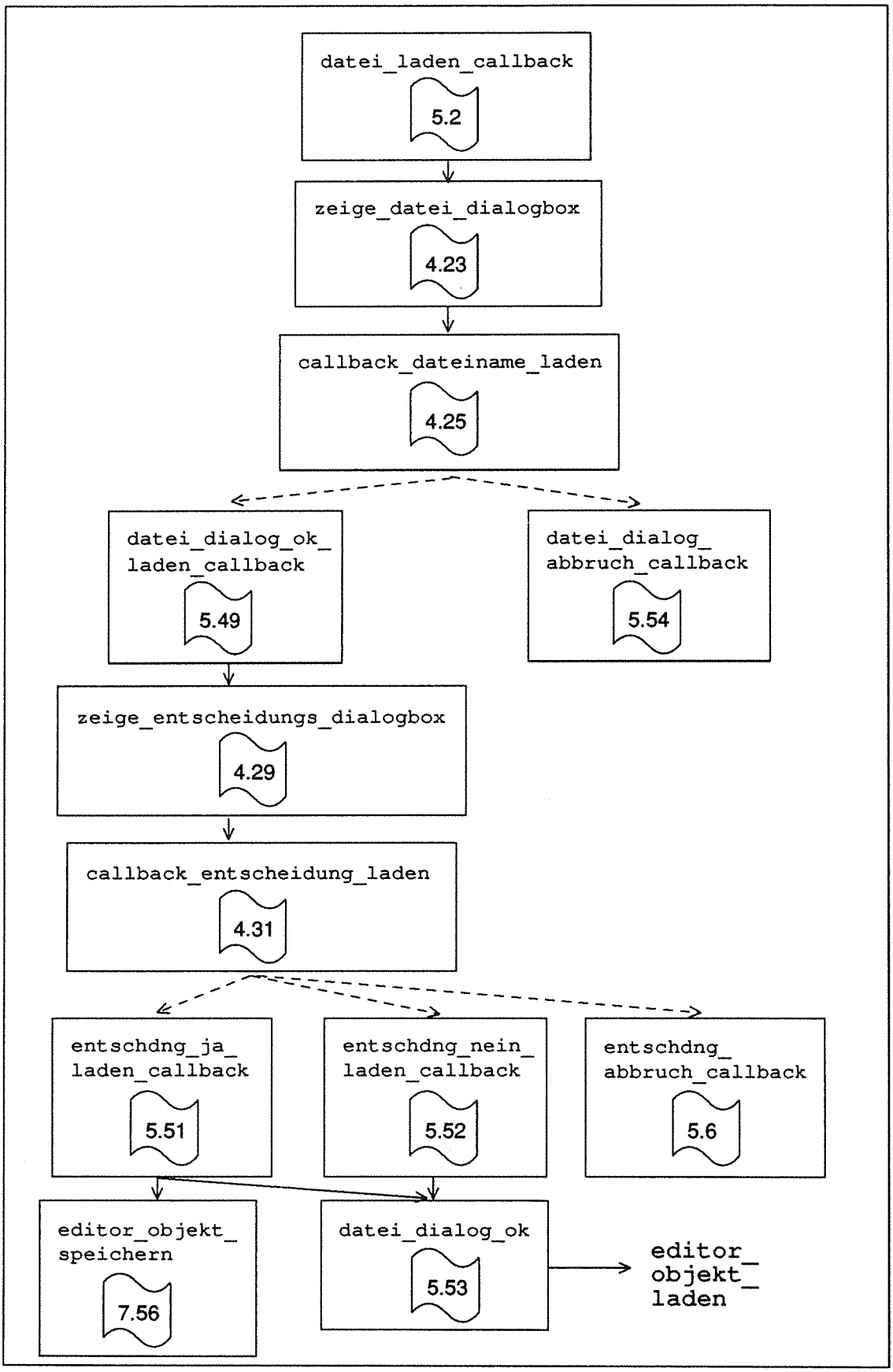


Abb. 8-22

Ausschnitt aus der Funktionshierarchie zum Laden von Objekten

Wenn man das Menü `Edit:Laden` anklickt, wird zunächst die Callback-Funktion `datei_laden_callback` aufgerufen, zusammen mit einem Dialogparameter (s. Abb. 8-22). Dieser Parameter gibt an, welche Callback-Funktionen dem `OK`- und `Abbruch`-Button in der Dateiauswahl-Box zugewiesen werden. Die Funktion `zeige_datei_dialogbox` wird auch beim Start des Programms verwendet (mit einem anderen Dialogparameter), ein Drücken des `Abbruch`-Buttons führt dort zur Beendigung des Programms. Beim Dateidialog während des Programmlaufs wird nach Drükken des `Abbruch`-Buttons nur das Dateidialog-Widget vom Bildschirm gelöscht.
In der Dateidialog-Box kann nun eine Datei ausgewählt werden, nach der Auswahl wird mit dem `OK`-Button die Funktion `datei_dialog_ok_laden_callback` augerufen. Zur Sicherheit wird nun, mit `zeige_entscheidungs_dialogbox`, nachgefragt, ob das bereits gezeichnete Objekt gesichert werden soll. Auch die hier verwendete Entscheidungsbox wird mehrfach verwendet mit unterschiedlichen Texten und Callback-Funktionen.
Falls der Benutzer sichern möchte, wird in `entschdng_ja_laden_callback` das Objekt gespeichert und danach `datei_dialog_ok` aufgerufen.
Dort wird

- der Dateiname aus der `datei_dialog`-Struktur gelesen (in die er nach Beendigung des Dateidialogs, d.h. Drücken des `OK`-Buttons, geschrieben wurde).
- das Suffix `'*.dat'` angehängt, wenn noch nicht vorhanden (`ergaenze_suffix_in_name`).
- der Dateiname gespeichert, wenn der Aufruf beim Start erfolgt. Dazu wird die Versionsnummer gelöscht (`loesche_version_in_name`), da diese beim Speichern nicht benötigt wird.
- getestet, ob die Datei vorhanden ist und wenn ja, `editor_objekt_laden` aufgerufen.

Danach ist ein Objekt vorhanden (`objekt_vorhanden`), eine evtl. bestehende Markierung wird gelöscht (`loesche_aktivierung`), und es werden Regionen erzeugt (`erzeuge_regionen`).
In `editor_objekt_laden` wird der bisherige Speicherplatz für die Objekte freigegeben und neuer angefordert. Danach

- wird die zu lesende Datei geöffnet,
- die Dateikennung überprüft,
- es werden die Anzahl der Objekte,
- die Objektattribute,
- in einer Schleife die Segmente und
- die E/A-Objekte gelesen.

Das Lesen ist im Prinzip das Gleiche wie das Schreiben, nur mit `fread` anstatt `fwrite` und dem Anfordern von Speicherplatz, wenn notwendig.

8.11 Weitere Funktionen

8.11.1 Optionen-Menü

Hier werden vier Buttons angeboten:

- `E/A-Namen EIN/AUS` mit der Callback-Funktion `optionen_ea_namen_callback`. Diese Funktion ändert den `editor.ea_namen_ausgeben`-Schalter und den Text im Widget. Die Funktion `ausgabe_objekt` wird automatisch aufgerufen, da das Pulldown-Menü das Ausgabefenster verdeckt und damit ein `expose`-Event gesendet wird. Dieser Expose-Event startet die Funktion `expose_callback`, in der dann `ausgabe_objekt` aufgerufen wird.
- mit `Raster ein/aus` kann im Ausgabefenster ein Raster ausgegeben werden, um graphische Objekte "einzufangen". Dazu wird in der Funktion zur Erzeugung der Objekte die Funktion `naechster_rasterpkt` aufgerufen, um die Koordinatenwerte zu ändern.
- `Raster gross/klein` bietet eine Auswahl von zwei verschiedenen Rastern.
- Ausgabe startet über `optionen_ausgabe_callback` die Funktion `ausgabe_objekt`.
- Dateiname gibt mit `optionen_meldung_namen_callback` und der Funktion `meldung` den Objektnamen im Meldungsfenster aus.
- Notepad: `optionen_notepad_callback` startet den Mini-Editor mit einem `system`-Aufruf.

8.11.2 Programmende

Wenn man das Programm beenden möchte (`Datei:Ende`) wird, wie vor dem Laden eines neuen Objektes, die Funktion `zeige_entscheidungs_dialogbox` (s. Abb. 8-23) aufgerufen, diesmal mit dem Parameter `dialog_entscheidung_ende`.
In `callback_entscheidung_ende` werden der Entscheidungsbox Callback-Funktionen zugewiesen, bevor sie angezeigt wird. Mit der Box wird gefragt, ob vor dem Programmende das Objekt gesichert werden soll. Drückt der Benutzer `'Ja'`, werden die Daten mit `objekt_speichern` gesichert, bevor das Programm mit `exit` verlassen wird.

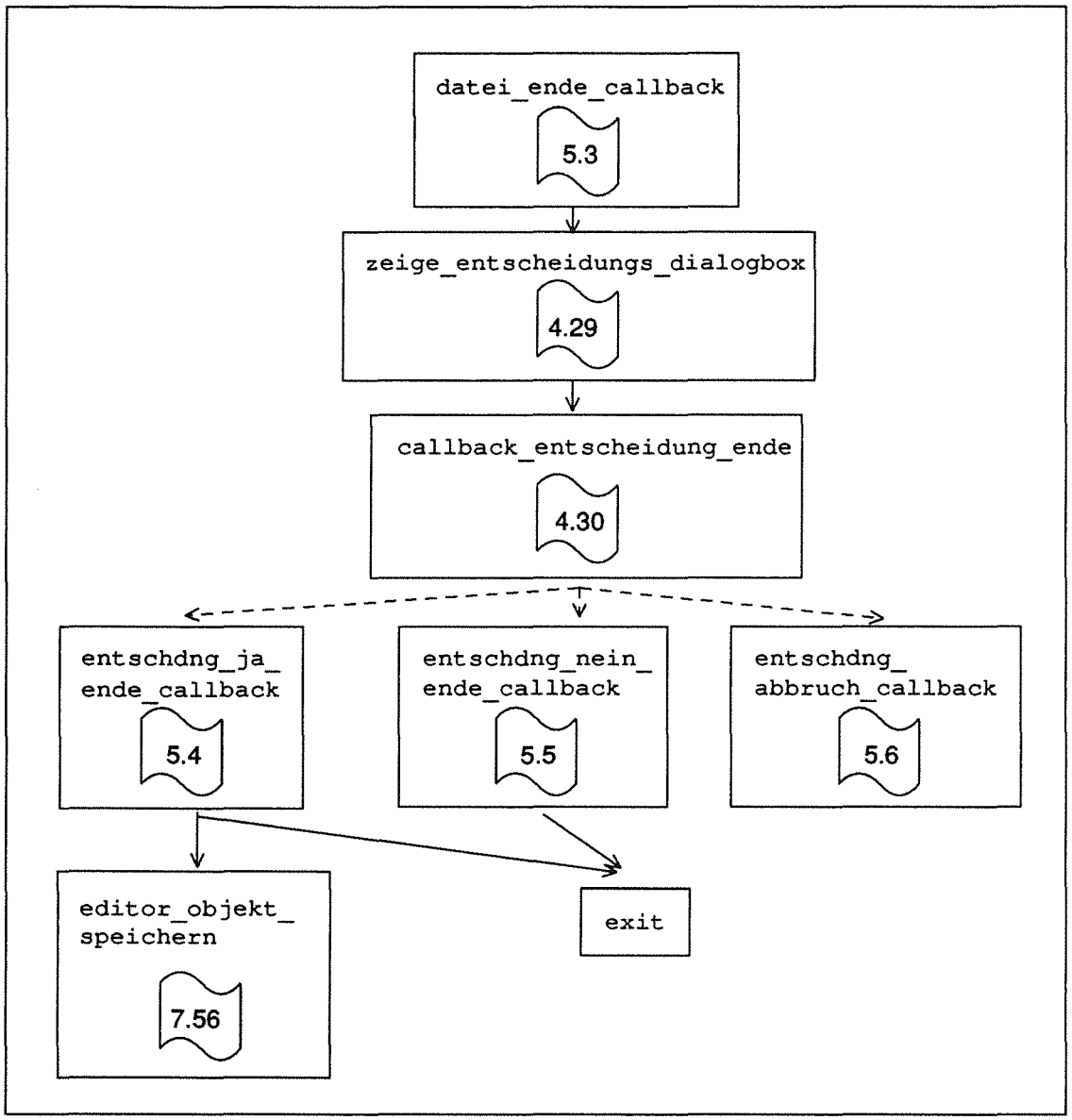


Abb. 8-23

Ausschnitt aus der Funktionshierarchie zur Beendigung des Editors

9 Literaturverzeichnis

Die aufgeführte Literatur wurde im Text zitiert, oder diente als Unterlage bei der Systementwicklung.

[Byte 1989] D. Pountain
The X Window System
Byte, Januar 1989, S. 353 - 360

[Keller 1988] H. B. Keller
Echtzeitsimulation zur Prozeßführung komplexer Systeme.
Berlin: Springer 1988

[Keller 1988] H. B. Keller, M. Gauges
XGAP - Programmbeschreibung,
unveröffentlichter Bericht, 1988

[Kernighan 1983] B. W. Kernighan / D. M. Ritchie
Programmieren in C
Carl Hanser Verlag, 1983

[Newman 1986] W. M. Newman / R. F. Sproull
Grundzüge der interaktiven Computergrafik
McGraw-Hill, 1986

[Nye 1989] A. Nye (ed.)
X Protocol Reference Manual
O'Reilly & Associates, 1989

[Nye 1990-1] A. Nye
Xlib Programming Manual
O'Reilly & Associates, 1990

[Nye 1990-2] A. Nye (ed.)
Xlib Reference Manual
O'Reilly & Associates, 1990

[Nye 1990-3] A. Nye / T. O'Reilly
X Toolkit Intrinsics Programming Manual
O'Reilly & Associates, 1990

[Reilly 1990] T. O'Reilly (ed.)
X Toolkit Intrinsics Reference Manual
O'Reilly & Associates, 1990

[Scheifler 1986] R. W. Scheifler / J. Gettys
The X Window System
ACM Transactions on Graphics,
Vol. 5, No. 2, April 1986, S. 79 - 109

[XUI 1988] XUI Programming Manuals
Vol. 1 - Vol. 5
Digital Equipment Corporation, 1988

10 Anhang A: Benutzungsanleitung Editor

10.1 Start

Beim Start des Editors erscheint zunächst ein Dateiauswahlfenster (s. Abb. 10-1).

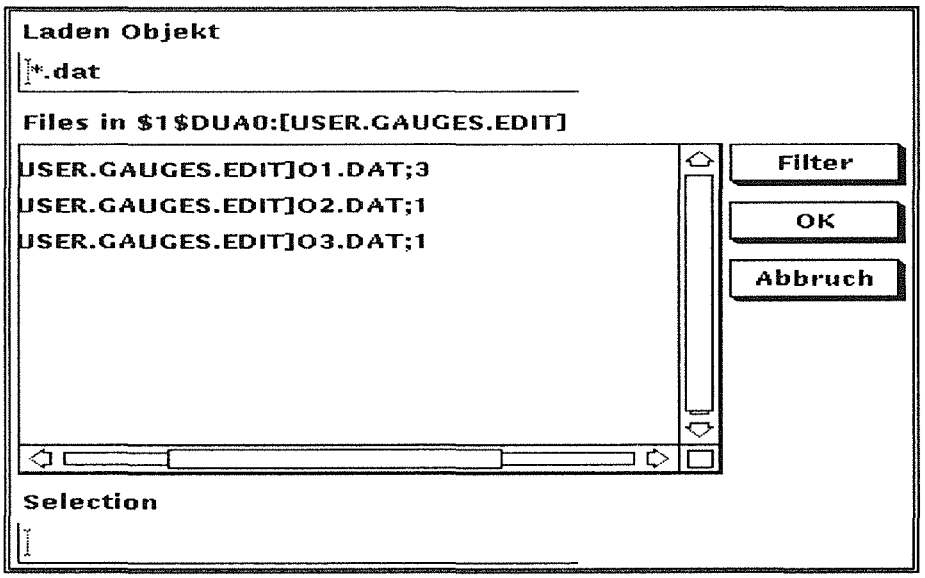


Abb. 10-1
Dateiauswahl-Fenster

Zu Beginn werden aus dem lokalen Verzeichnis alle Dateien mit der Endung `.dat` angezeigt. Diese Endung wird vom XGAP-Editor für die graphischen Objekte verwendet. Möchte man die Dateien eines anderen Verzeichnisses anzeigen, klickt man mit der Maus das Filter-Feld unter `Laden Objekt` an, gibt dort das neue Verzeichnis ein und drückt `<Return>` (oder klickt den Button `Filter` an). Mit dem Filter-Feld werden Dateien vorgewählt (in Abb. 10-1 die Dateien mit der Endung `dat`).
Ein vorhandenes Objekt lädt man durch einen Doppelklick auf die zugehörige Datei im Auswahlfenster oder man klickt die Objektdatei einmal an und drückt den `OK`-Button.
Möchte man ein neues Objekt erstellen, gibt man den Objektnamen (= Dateinamen) im Feld `Selection` ein und drückt `<Return>`.
Das Suffix `dat` wird automatisch angehängt. Der gewählte Objektname bleibt während der gesamten Sitzung gleich.
Mit dem `Abbruch`-Button wird der Editor sofort wieder beendet.

Die Benutzeroberfläche des Editors besteht aus drei Fenstern (s. Abb. 10-2):

- einem Auswahlfenster am linken Rand,
- einem Ausgabefenster zum Zeichnen der Objekte
- und einem Meldungsfenster über dem Ausgabefenster; hier wird der Objektname angezeigt.

Am oberen Rand befindet sich eine Menüleiste mit fünf Pulldown-Menüs.

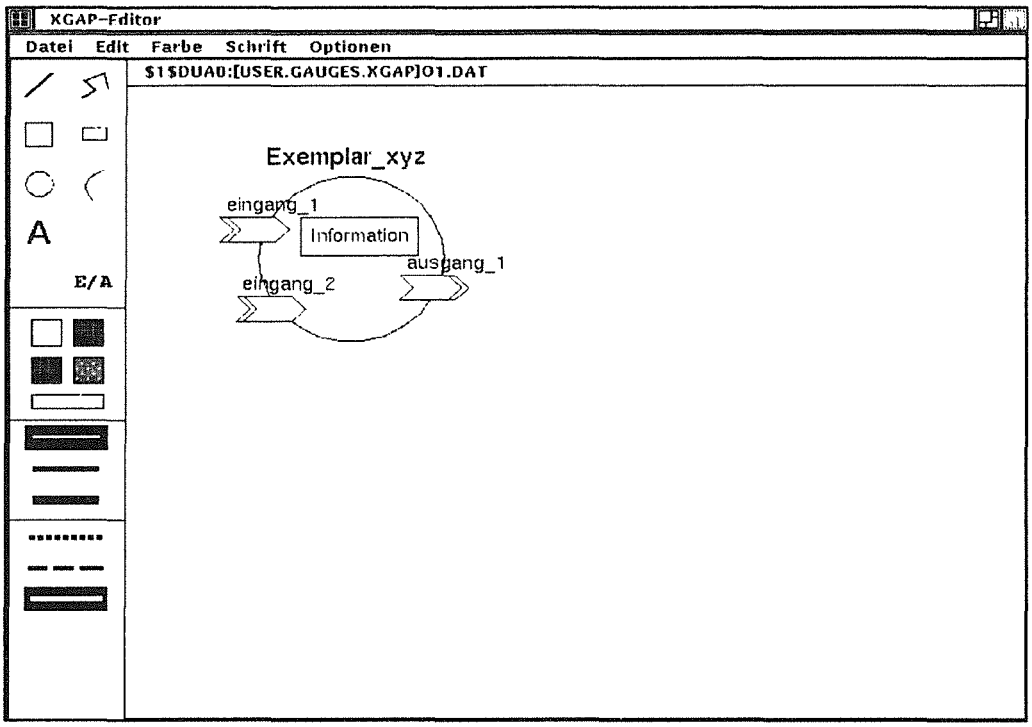


Abb. 10-2

Benutzeroberfläche Editor

10.2 Auswahl Zeichenelement und Attribute

Vor dem Zeichnen wählt man aus dem Auswahlfenster eines der folgenden Grundelemente aus und klickt es an:

- Strecke
- Polyline
- Quadrat
- Rechteck
- Kreis
- Kreisbogen
- Text (A)
- E/A-Objekt (E/A).

Sobald man einen der Buttons angeklickt hat, erscheint der Auswahlbutton (⊞), mit dem man wieder in den Auswahlmodus zurückkommt. Der Auswahlmodus dient dazu, gezeichnete Elemente zu manipulieren (s. Kap. 8.4).
Das Zeichenobjekt kann durch Anklicken eines anderen Zeichenbuttons direkt gewechselt werden.

Folgende Füllfarben werden angeboten:

- Rot
- Blau
- Grün.

Die gewünschte Füllfarbe wird durch Anklicken eines der Farbbuttons im Auswahlfenster gewählt und unterhalb der Farbbuttons angezeigt.

Unterhalb der Farbbuttons befinden sich Buttons zur Auswahl der Kantendicke und der Kantenart. Folgende Kantendicken werden angeboten, der aktivierte Button wird invers dargestellt:

- schmal
- mittel
- breit.

Außerdem gibt es die Kantenarten:

- gepunktet
- gestrichelt
- durchgezogen.

Die Attribute Kantendicke und Kantenart werden bei E/A-Objekten nicht verwendet.
Die Kantenfarbe kann mit dem Pulldown-Menü `Farbe` für alle Zeichenelemente vorgewählt werden. Folgende Kantenfarben sind möglich:

- Schwarz
- Blau
- Rot
- Grün
- Weiß.

Bevor man den Text-Button (`A`) anklickt, kann eine Schriftgröße aus dem Pulldown-Menü Schrift gewählt werden:

- 14 Punkt
- 18 Punkt
- 24 Punkt.

10.3 Zeichnen Objekt

Alle Elemente werden interaktiv erstellt; d.h., ihre Größe kann mit der Maus verändert werden, solange der Mausbutton gedrückt bleibt. Das Element wird in der Erstellungsphase schwarz gestrichelt gezeichnet. Erst nach dem Loslassen des Mausbuttons wird es farbig ausgegeben.

Strecke:	Zunächst fährt man (mit der Maus) im Ausgabefenster an den Startpunkt der Strecke. Der Startpunkt hat die Position, an der die linke Maustaste gedrückt wird. Nun hält man die Taste gedrückt und verschiebt die Maus; die Strecke verändert sich mit dem Verschieben. Sobald man die Taste losläßt, wird die endgültige Strekke ausgegeben.
Polyline:	Erste Teilstrecke wie bei Strecke; läßt man die Taste los, wird die erste Linie ausgegeben, und die zweite kann durch Veränderung der Mausposition erzeugt werden. Drückt man die Maustaste wieder kurz, wird die nächste Strecke gezeichnet usw. Beendet wird der Polygonzug mit einem doppelten Mausklick. Befindet sich der letzte Punkt nahe am Anfangspunkt (Abstand kleiner 5 Pixel), wird der Polygonzug geschlossen (d.h der Endpunkt wird gleich dem Anfangspunkt gesetzt).
Quadrat und Rechteck:	Der Start ist wieder gleich wie bei einer Strecke; der Startpunkt ist der linke obere Eckpunkt des Quadrats bzw. Rechtecks. Das Objekt verändert seine Größe, wenn man die Maus nach rechts unten bewegt.
Kreis:	Wie bei Quadrat; Startpunkt ist der linke obere Eckpunkt des umgebenden Quadrats.
Kreisbogen:	Die Erstellung eines Kreisbogens verläuft in zwei Phasen (s. Abb. 10-3): Phase 1: Bestimmung Kreismittel- und Startpunkt als Anfangs- und Endpunkt einer Strecke (Radius). Phase 2: Nach Loslassen des Mausbuttons wird der Kreisbogen erstellt; durch Bewegen der Maus (entweder im Uhrzeigersinn oder entgegen) verändert sich die Länge des Bogens, Startpunkt und Radius sind jetzt fest. Nach nochmaligen Drücken der Maustaste wird der Bogen farbig gezeichnet.

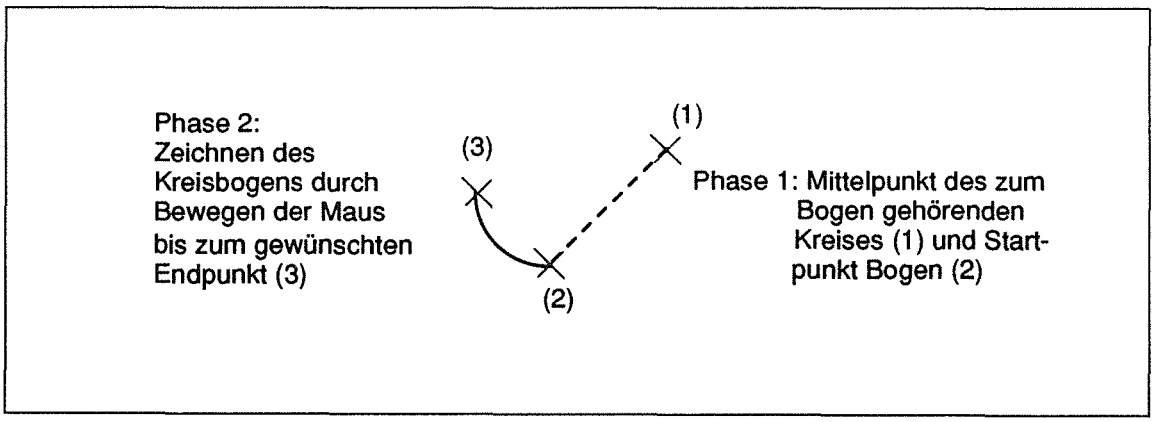


Abb. 10-3
Zeichnen eines Kreisbogens

Text und E/A-Objekte erstellt man mit Dialogfenstern:

Text: Nach Drücken des Buttons `A` erscheint ein Dialogfenster (s. Abb. 10-4), in das der Text eingegeben wird. Drückt man dann den `OK`-Button oder `<Return>`, verschwindet das Dialogfenster, und der eingegebene Text erscheint an einer Defaultposition im Ausgabefenster. Man kann nun mit der Maus den Text anklicken und an die gewünschte Position schieben.
Drückt man den `Abbruch`-Button, wird kein Text erzeugt.
Die Schriftgröße und der Text können nach dem Erstellen nicht mehr geändert werden.

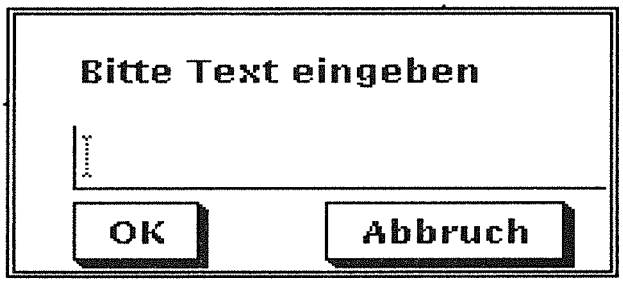


Abb. 10-4
Dialogfenster zur Texteingabe

E/A-Objekt: Im E/A-Dialogfenster (s. Abb. 10-5) wird folgendes eingegeben:

- Name des Objekts (wird in der Anwendung zur Identifizierung benötigt; falls der `OK`-Button gedrückt wird, bevor hier Text eingegeben ist, erscheint ein Hinweisfenster mit der Aufforderung, einen Namen einzugeben).
- Text (beliebig, als zusätzliche Information).

Außerdem können folgende Attribute bestimmt werden:

- Typ des Objekts (Eingang, Ausgang oder Durchgang, Kennzeichnung durch Doppelkante)
- Richtung (nach rechts oder nach links)
- Größe des Objekts (klein, mittel, groß).

Die gewählte Größe und Richtung wird angezeigt.
Wird der `Abbruch`-Button gedrückt, verschwindet die Dialogbox, ohne daß ein E/A-Objekt ausgegeben wird.
Nach Drücken des `OK`-Buttons wird das neue E/A-Objekt zunächst in der linken oberen Ecke des Ausgabefensters gezeichnet. Es kann danach genauso wie ein Textelement verschoben werden.

Nach dem Zeichnen wird aus den Elementen ein sog. Segment erzeugt, das manipuliert werden kann.

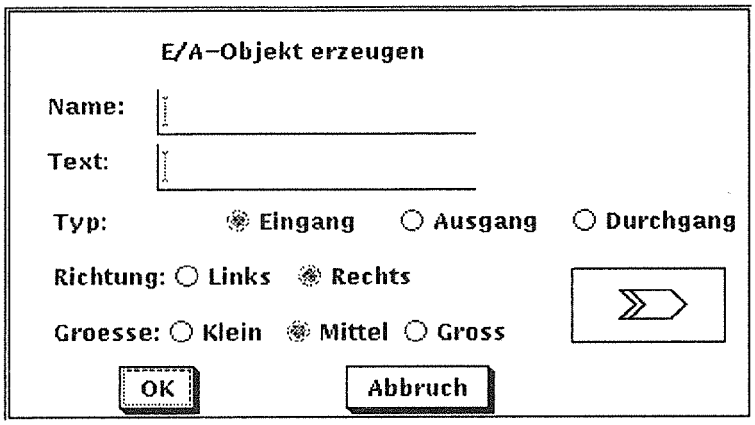


Abb. 10-5

Dialogfenster für E/A-Objekte

10.4 Ändern eines Elements

Um ein Element-Attribut zu verändern oder das Element zu verschieben, muß zunächst durch Anklicken des Auswahlbuttons der Auswahlmodus aktiviert werden. (Sowohl nach der Eingabe von Text als auch nach der Erzeugung eines E/A-Objektes befindet sich der Editor, nachdem die jeweilige Dialogbox verschwunden ist, automatisch im Auswahlmodus.)

In diesem Modus kann man, neben dem bereits erläuterten Auswählen eines Zeichenmodus, die Kante eines Elements anklicken, um das zugehörige Segment zu aktivieren. Ist ein Segment aktiviert, wird das graphische Teilobjekt gestrichelt markiert. Da das Erkennen von Segmenten mit Hilfe von Elementkanten erfolgt, kann man innerhalb eines Elements (z.B. eines Quadrats) ein weiteres zeichnen (z.B. einen Kreis), das unabhängig vom äußeren manipuliert werden kann.
Innerhalb von Textsegmenten und E/A-Objekten können keine weiteren Segmente liegen. Bei diesen Elementen ist auch ein Anklicken inmitten der Segmente möglich.

Wenn ein Element aktiviert (bzw. markiert) ist, kann man

- es verschieben durch Drücken des linken Mausbuttons und Verschieben der Maus (wenn Segmente übereinanderliegen wird beim Verschieben evtl. ein Teil eines anderen Segmentes mit verschoben. Sobald die Maustaste losgelassen wird, wird das Originalelement wieder restauriert),
- ein Attribut verändern durch Anklicken eines der Farb- oder Kantenbuttons im Auswahlfenster oder mit dem Pulldown-Menü `Farbe` (für Kantenfarbe),
- ein E/A-Objekt dadurch verändern, daß man, nachdem man es mit der linken Maustaste aktiviert hat, mit der mittleren Maustaste eine Dialogbox aufruft, in der die Daten des aktivierten E/A-Objekts geändert werden können,
- es löschen mit dem Pulldown-Eintrag `Edit:Loeschen`,
- es kopieren durch `Edit:Kopieren`; danach erscheint ein weiteres Exemplar neben dem Quellsegment, das verschoben werden kann.

Falls ein Segment mit `Edit:Loeschen` unsichtbar gemacht wurde, kann man das letzte Löschen mit `Edit:Undo` wieder rückgängig machen.

10.5 Speichern und Laden

Möchte man während einer Sitzung das Objekt speichern, geschieht dies mit dem Pulldown-Menü `Datei:Speichern`. Das gesamte Objekt wird mit dem zu Beginn vergebenen Namen gesichert.

Mit `Datei:Laden` kann ein anderes Objekt als Vorlage geladen werden. Dazu erscheint die gleiche Dateiauswahl-Box wie beim Programmstart (s. Abb. 10-1). Der Objektname im Editor bleibt bestehen, auch wenn ein neues Objekt geladen wird.
Vor dem Laden eines neuen Objektes wird gefragt, ob das alte Objekt gespeichert werden soll.

10.6 Optionen

Das Menü `Optionen` bietet folgende Aktionen an:

- `E/A-Namen EIN/AUS`
  Ein Button, der zwei Zustände haben kann (Ein oder Aus). Wenn er eingeschaltet ist, werden die Namen der E/A-Objekte auf der Arbeitsfläche über den Objekten ausgegeben.
- mit `Raster ein/aus` kann im Ausgabefenster ein Raster ausgegeben werden, um graphische Objekte "einzufangen".
- `Raster gross/klein` bietet eine Auswahl von zwei verschiedenen Rastern.
- `Ausgabe`
  Normalerweise wird, sobald die Arbeitsfläche von irgendeinem anderen Fenster verdeckt war, der gesamte Inhalt automatisch neu ausgegeben. Mit diesem Button kann die Restaurierung des Ausgabefensters manuell erfolgen (z.B. nach dem Zeichnen einer geschlossenen Polyline).
- `Dateiname`
  gibt den Namen, mit dem das Objekt unter VMS gespeichert wird, im Meldungsfenster aus.
- `Notepad`
  ruft den Mini-Editor zum Erstellen von Textdateien auf. Der graphische Editor pausiert, solange Notepad läuft. Nach Beenden von Notepad kann im graphischen Editor wieder weitergearbeitet werden.

10.7 Ende

Beendet wird die Sitzung mit `Datei:Ende`. Vor dem Beenden des Programms wird gefragt, ob das erstellte Objekt gesichert werden soll.

11 Anhang B: Anwendungsbeispiel

11.1 Editor

Zunächst werden mit dem Editor zwei Objekte gezeichnet, von denen Exemplare im Anwendungsbeispiel erzeugt werden sollen. Die Exemplare haben Eingangs-, Ausgangs- und Durchgangsobjekte.

Das erste Objekt (`o1.dat`, s. Abb. 11-1) besteht außer den E/A-Objekten aus einem Kreis-Segment und mehreren Text-Segmenten mit unterschiedlichen Schriftgrößen.

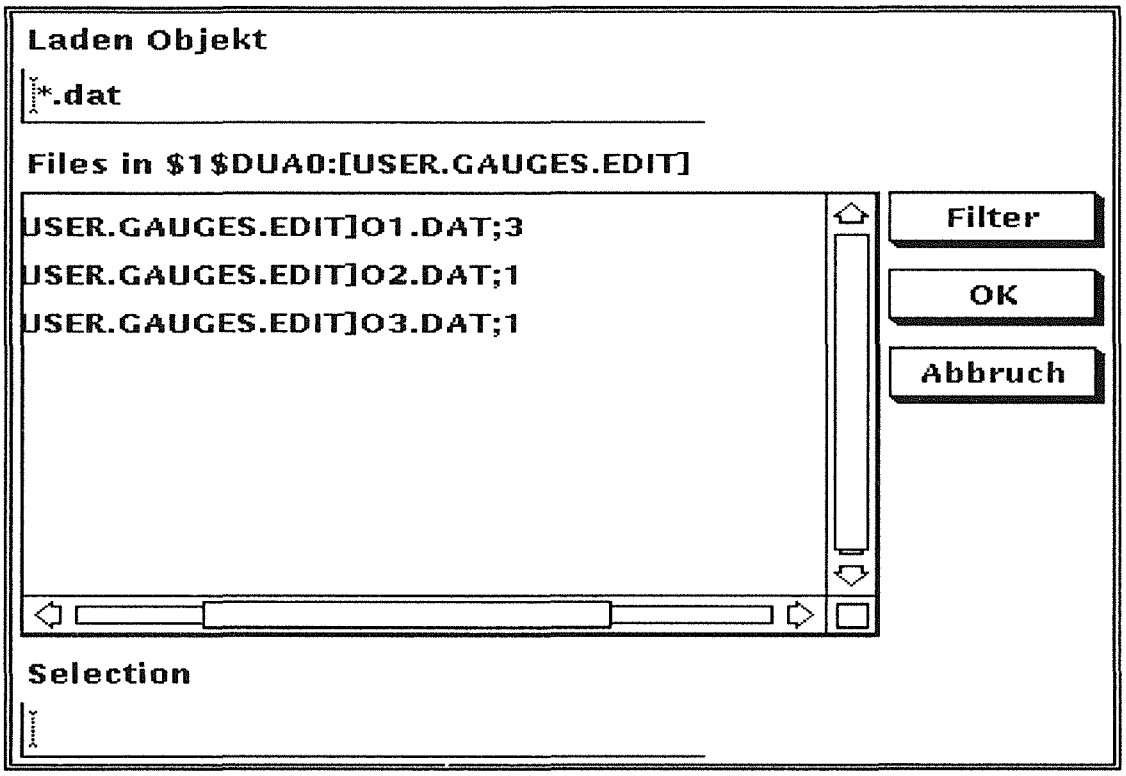


Abb. 11-1

Benutzeroberfläche Editor mit Objekt o1

Das zweite Objekt (`o3.dat`, s. Abb. 11-2) - es soll im Beispiel als Rahmen für das erste Objekt dienen - besteht außer zwei Durchgangsobjekten nur aus einem Rechteck. `o3.dat` soll `o1.dat` *enthalten*, d.h. eine Betrachtungsebene höher stehen.

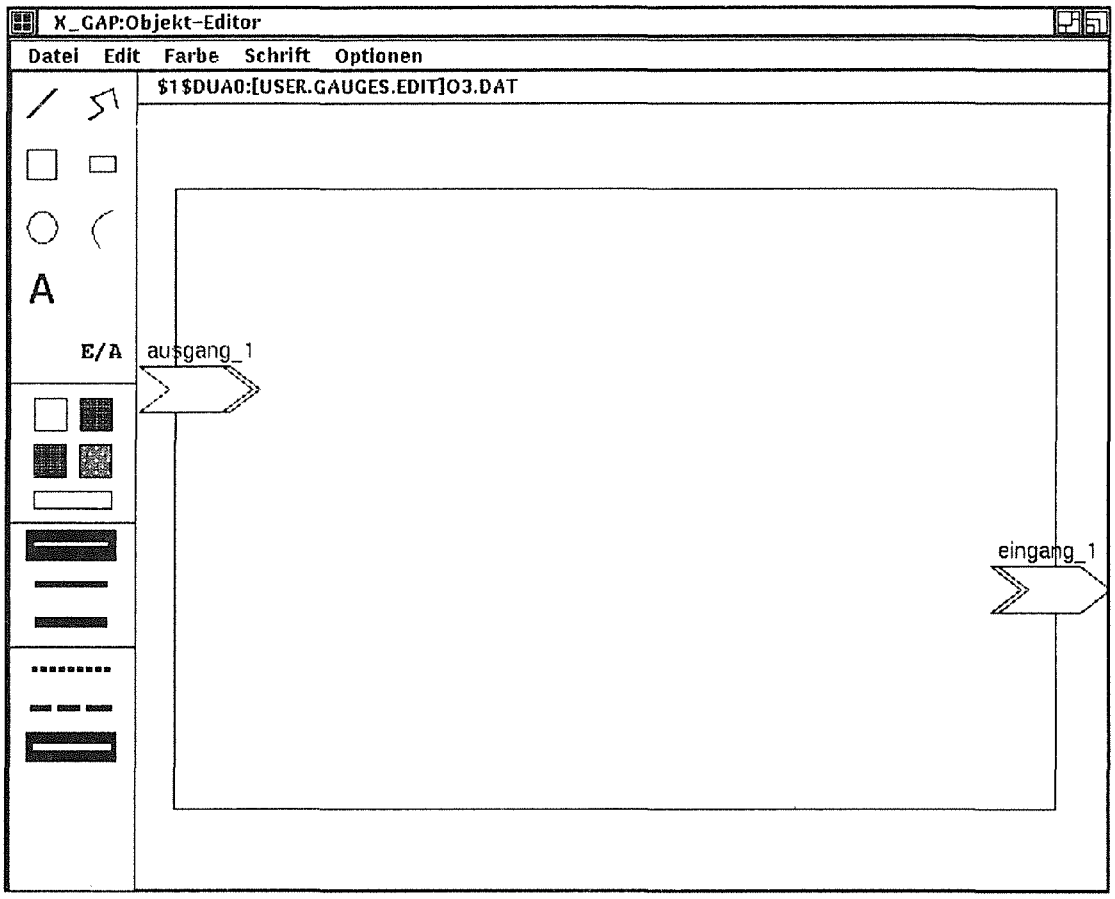


Abb. 11-2

Benutzeroberfläche Editor mit Objekt o3

11.2 Anwendung

Im folgenden Beispiel wird

- das Erzeugen von zwei Exemplaren,
- das Erzeugen von Verbindungen zwischen Exemplaren
- und die Ausgabe eines Popup-Menüs erläutert.

Die erste Anweisung in jedem XGAP-Anwendungsprogramm ist

```
init_xgap (name_erstes_fenster);
```

Damit wird XGAP initialisiert und das erste Fenster (`standard_fenster_gross`) erzeugt.
Im Beispiel wurde dann ein weiteres Standardfenster mit

```
erzeuge_fenster ("Hauptfenster", standard_fenster, 0);
```

erzeugt (Name `Hauptfenster`). Alle weiteren Funktionen beziehen sich auf dieses Fenster.

Danach werden die Buttons im Aktionsfeld belegt.
Folgende Funktionen sind dazu notwendig:

```
/* Eingabeparameter: Aktion sofort aktivieren */
setze_xgap_wert (xgap_aktion_aktiv, ja);
/* Button bleibt nach Anklicken bestehen */
setze_xgap_wert (xgap_aktion_bleibt, ja);


/* erster Button */
neuer_aktions_eintrag ("Hauptfenster",
                       aktion_1, text);

/* weitere Buttons, XGAP-Parameter bleiben bestehen, wenn sie nicht
   explizit geändert werden */
    .
    .
    .
/* letzter Button */
neuer_aktions_eintrag ("Hauptfenster",
                       aktion_8, text);
```

Wenn die Buttons belegt sind, wird darauf gewartet, daß der Benutzer einen davon anklickt:

```
auswahl_aktion ("Hauptfenster");
```

Wenn die Funktion beendet ist, wird der gedrückte Button abgefragt mit:

```
lese_xgap_wert (xgap_aktions_nr);
```

Als Ergebnis erhält man einen Integerwert (`aktion_1` bis `aktion_8`); je nach gewählter Aktion wird nun die zugehörige Funktion aufgerufen.

In Abb. 11-3 ist ein Exemplar vom Vorlageobjekt `o3.dat` ausgegeben. Es wurde mit

```
erzeuge_exemplar ("Hauptfenster",
                  "exemplar_1", "o3.dat");
```

erzeugt

und mit

```
ausgabe_exemplar ("Hauptfenster", "exemplar_1");
```

ausgegeben. Der Exemplarname (`exemplar_1`) und Dateiname des Vorlageobjekts wurde mit

```
lese_eingabe ("Hauptfenster");
```

vom Benutzer erfragt.

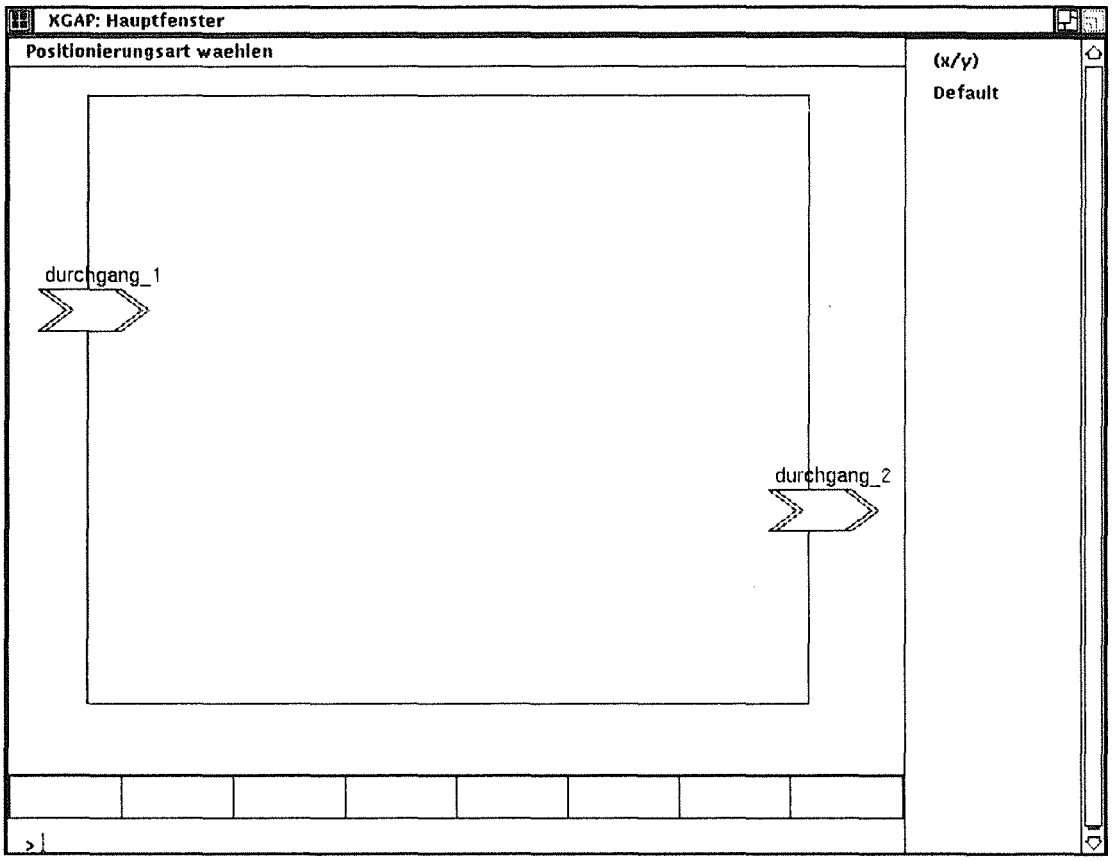


Abb. 11-3

Benutzeroberfläche XGAP mit Exemplar exemplar_1

Im Beispiel (s. Abb. 11-3) ist bereits das nächste Exemplar (`exemplar_2`, Vorlageobjekt `o1.dat`) erzeugt, es soll jetzt ausgegeben werden. Dazu hat der Benutzer den Button `Ausgabe` angeklickt. Im Menüfeld werden nun mehrere Auswahlmöglichkeiten angeboten mit

```
neuer_menu_eintrag ("x,y");
neuer_menu_eintrag ("Default");
```

Mit

```
auswahl_menu ("Hauptfenster");
```

wird darauf gewartet, daß ein Eintrag im Menüfeld angeklickt wird. Danach wird mit

```
loesche_menu_feld ("Hauptfenster");
```

die Menüfeld-Einträge gelöscht und mit

```
lese_xgap_text (xgap_menu_text);
```

abgefragt, welcher Eintrag angeklickt wurde. Im Beispiel ist dies `(x,y)`, danach wird abgefragt, an welcher Position das Exemplar ausgegeben werden soll.

Hat der Benutzer Koordinaten eingegeben, werden diese mit

```
setze_xgap_wert (xgap_positions_art, pos_koordinaten);
setze_xgap_wert (xgap_x, x-Wert);
setze_xgap_wert (xgap_y, y-Wert);
```

gesetzt, bevor die Funktion zur Ausgabe des Exemplars (`exemplar_2`) aufgerufen wird.

In Abb. 11-4 ist `exemplar_2` ausgegeben und das Durchgangsobjekt (`durchgang_1`) von `exemplar_2` mit dem Eingangsobjekt (`eingang_1`) von `exemplar_1` verbunden. Um die Verbindungen interaktiv zu erzeugen, werden nach dem Anklicken des `Verbindung`-Buttons zunächst die Funktionen

```
/* Ausgabe einer Meldung im Statusfeld */
meldung ("Hauptfenster",
         "Bitte Ausgangsobjekt anklicken");
/* Abfrage */
auswahl_objekt ("Hauptfenster");
```

aufgerufen.

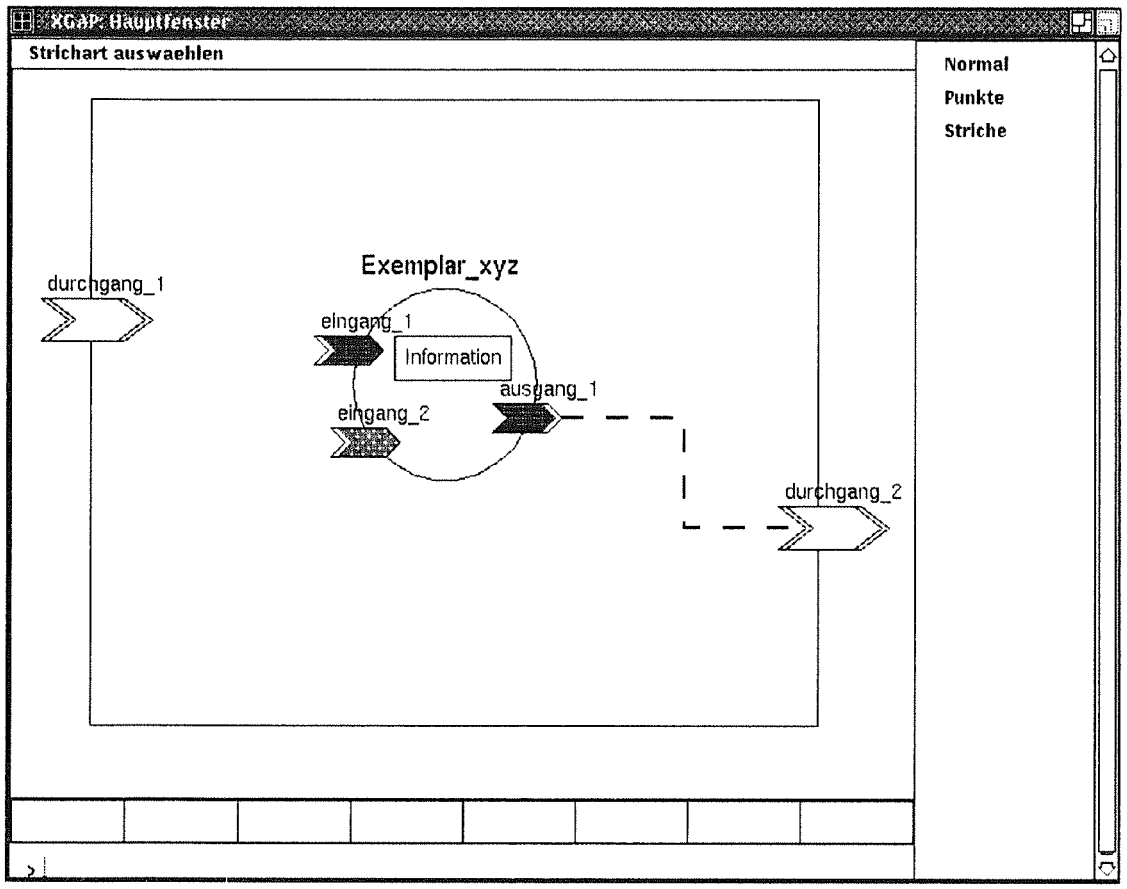


Abb. 11-4 Zwei Exemplare und eine Verbindung

Die Funktion `auswahl_objekt` wird beendet, sobald der Benutzer einen Punkt der Arbeitsfläche anklickt. Mit

```
lese_xgap_wert (xgap_typ);
```

und

```
lese_gap_wert (xgap_unter_typ);
```

kann dann überprüft werden, ob ein Exemplar (Typ `typ_exemplar`) bzw. ein Ausgangsobjekt (Untertyp `typ_ausgang`) getroffen wurde.

Eine weitere Abfrage erfolgt für das Eingangsobjekt; danach werden mit dem Menüfeld Attribute wie Farbe, Strichdicke und Strichart abgefragt. In Abb. 11-4 werden die möglichen Stricharten im Menüfeld zum Anklicken angeboten. Je nach Ergebnis wird dann mit

```
setze_xgap_wert (xgap_strichart, ...);
```

eine der drei möglichen Stricharten gesetzt.
Die Funktion

```
verbinde_exemplare ("Hauptfenster", "verbindung_2");
```

liest die mit `setze_xgap_wert` übergebenen Attribute und erzeugt `verbindung_2`.

In Abb. 11-5 ist das Popup-Menü dargestellt.

Zunächst wurden mit

```
neuer_popup_eintrag ("Hauptfenster", "Text1");
```

und

```
neuer_popup_eintrag ("Hauptfenster", "Text2");
```

die Einträge gesetzt.
Damit das Popup-Menü automatisch neben einem Exemplar positioniert wird, setzt man

```
setze_xgap_wert (xgap_positions_art, pos_automatisch);
```

und ermittelt mit

```
auswahl_objekt ("Hauptfenster");
```

ein Exemplar. Wenn der Benutzer ein Exemplar anklickt, kann der Exemplarname mit

```
lese_xgap_text (xgap_name_exemplar);
```

gelesen werden. Dieser Ausgabeparameter ist auch Eingabeparameter für die Funktion

```
ausgabe_popup_menu ("Hauptfenster");
```

mit der das Popup-Menü ausgegeben wird.

Wenn der Benutzer nun das Exemplar `exemplar_2` angeklickt hat, wird das Popup-Menü rechts neben dem Exemplar ausgegeben.

Mit

```
auswahl_popup ("Hauptfenster");
```

kann auf das Anklicken eines Eintrags vom Benutzer gewartet werden;

```
loesche_popup_feld ("Hauptfenster");
```

löscht das Popup-Menü vom Bildschirm.

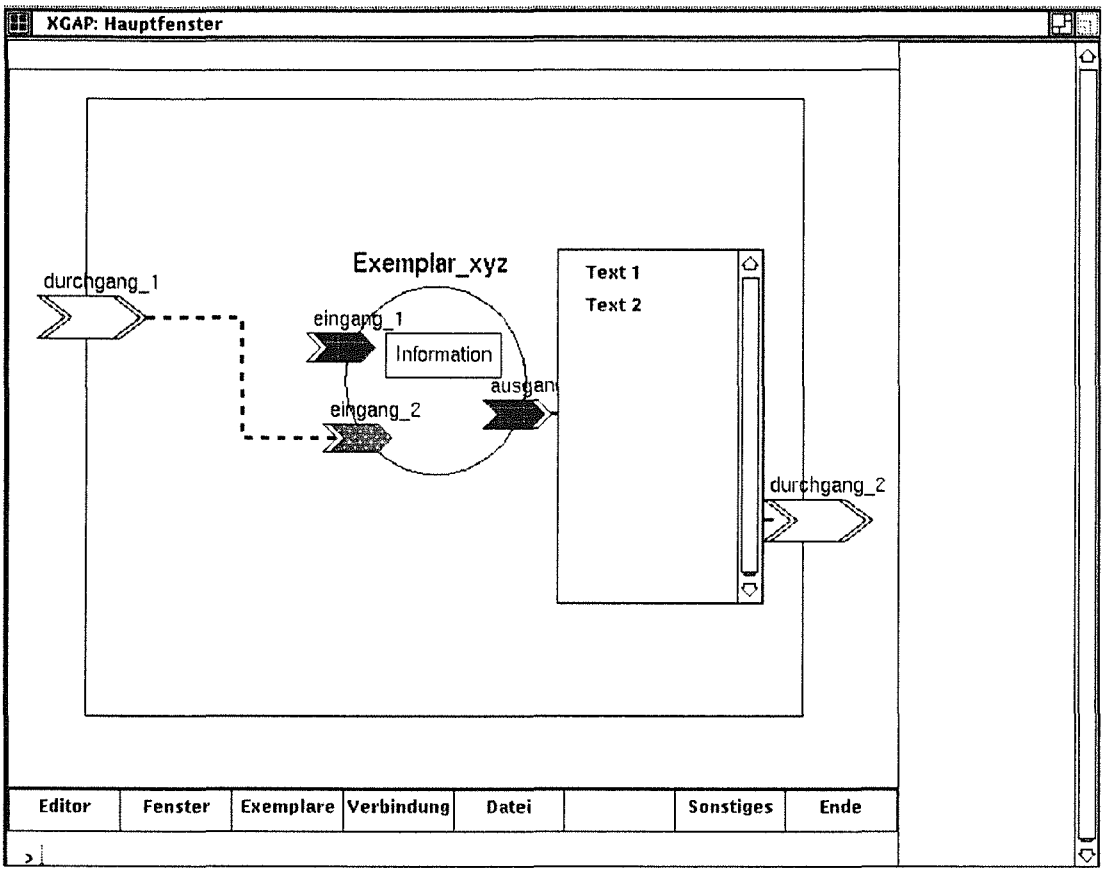


Abb. 11-5
Popup-Menü