Kapitel 2

Grafische Oberflächen und ihre Programmierung

Das erste Window-System wurde in den 70er Jahren von Xerox PARC entwickelt. Ende der 70er Jahre traten die grafischen Oberflächen mit den Apple Computern Lisa und Macintosh ihren Siegeszug an.
Grafische Benutzungsoberflächen haben sich heute zum Standard für anwenderfreundliche Applikationsprogramme entwickelt und rein textuelle Mensch-Maschine-Kommunikation weitgehend abgelöst. Was dem Benutzer im allgemeinen das Leben erleichtern soll, stellt für den Programmierer eine echte Herausforderung dar.
Auf dem zweidimensionalen Bildschirm müssen eine Vielzahl von Fenstern, Icons und Infoboxen mit unterschiedlichster Funktionalität dargestellt werden. Diese müssen durch Eingabegeräte wie Maus oder Tastatur vom Benutzer arrangiert, aktiviert oder mit Eingaben versehen werden können.
Neben der angestrebten intuitiven Bedienbarkeit von grafischen Benutzungsoberflächen erlaubt ein Window-System dem Benutzer auch die Ausgaben mehrerer Programme gleichzeitig zu beobachten und so quasi parallel zu arbeiten.

2.1 Der Window-Manager

Für die geometrische Verwaltung der einzelnen Anwendungen auf einem Bildschirm steht dem Anwender ein Window-Manager zur Verfügung. Er legt die Fenster-Layout-Politik fest.
Der Window-Manager manipuliert Umrandung, Größe und Position der Fenster sowie die Reihenfolge, in der die Anwendungen übereinanderliegen. So kann er verdeckte Fenster in den Vordergrund holen, Fenster über den Schirm bewegen und deren Größe verändern.
Für den Inhalt ihrer Fenster ist die Applikation selbst verantwortlich. Muß ein Fenster nachgezeichnet werden, weil es z.B. durch eine Verschiebe-Aktion nun nicht mehr verdeckt ist, sendet der Window-Manager lediglich eine Redraw-Nachricht an die Applikation.
Außerdem kann der Window-Manager neue Applikationen starten.

2.2 Eventhandling

Eine grafische Benutzeroberfläche muß in der Lage sein, auf zahlreiche verschiedene Ereignisse zu reagieren. Typische Events sind ein Mausklick, die Bewegung der Maus oder ein

Tastendruck. Die Ereignisse betreffen aber nicht nur die Benutzereingaben, sondern auch die Interaktion zwischen Applikationen.
Z.B. ein Expose-Event wird beim Start einer neuen Applikation ausgelöst. Wird die Maus in ein Fenster gefahren, gibt es einen Enter-Event.
Die Zahl verschiedener Events wird durch Kombinationen multipliziert. Applikationen sollen häufig auf einen Doppelklick reagieren oder erst auf das Loslassen der Maustaste; die Bewegung der Maus bei gedrückter Taste wird oft zur Auswahl in Menüs verwendet.
Die unterschiedlichsten Ereignisse können zu beliebiger Zeit in beliebiger Reihenfolge auftreten. Ein gutes Anwendungsprogramm ist so geschrieben, daß es vom Benutzer gesteuert wird und nicht umgekehrt.
Das typische Schema für die Verarbeitung von Interaktionen in einem Anwendungsprogramm ist die Event-Loop. Nach dem Schema eines endlichen Automaten, der einen zentralen Wartezustand kennt, werden die Übergänge zu anderen Zuständen durch Benutzereingaben ausgelöst.

```
Darstellung des Ausgangsbilds
do {
    Auswahl von Befehlen oder Objekten ermöglichen
    /* Das Programm wartet unbestimmte Zeit, bis Benutzer etwas eingibt */
    warte auf Benutzereingabe
    switch (Eingabe) {
        bearbeite Eingabe und führe ggf. Kommando aus
        bei Bedarf Bildschirm aktualisieren
    }
}
while(!Ende) /* Benutzer hat nicht “Ende” gewählt */
```

2.3 Eigene Applikationen

Während der Window-Manager mit einer Reihe von Events umgehen kann, die zu seiner Funktionalität gehören, muß sich der Entwickler einer grafischen Applikation mit den Ereignissen, die diese betreffen, selbst auseinandersetzen.
Soll eine Anwendung z.B. auf einen Knopfdruck reagieren, so muß der Event eines Mausklicks im Bereich der grafischen Darstellung des Buttons vom Entwickler abgehandelt werden. Dazu ist es notwendig, ein Applikationsprogramm zu schreiben, das auf dieses Ereignis hin die gewünschte Funktion ausführt, den sogenannten *Callback*.
Für (fast) jedes Betriebssystem gibt es ein zugehöriges Oberflächensystem, das mit Hilfe eines API (Application Programmer's Interface) manipuliert werden kann. Die Programmierung einer grafischen Applikation erfolgt mit Hilfe einer Graphical User Interface (GUI)-Sprache, die in einer Hochsprache alle notwendigen API-Vokabeln zur Beschreibung des Aufbaus und Ablaufs einer interaktiven grafischen Anwendung bereitstellt. Die Programmierung erfolgt zumeist in einer gängigen Programmiersprache, die GUI-Bibliotheken verwendet.
Unter UNIX ist die z.Zt. am meisten verbreitete Kombination X-Windows mit Xlib, Xt und dem Motif-Widget-Set. Die Programmierung erfolgt dabei meist mit Hilfe von C. Eine weitere Variante sind Bibliotheken, die auf diese Umgebung aufsetzen, wie z.B. Tk, das dann in Skript-Sprachen wie Tcl oder Perl5 verwendet werden kann.
Eine spezielle Umgebung kann unter Openstep verwendet werden, wo das dortige Applikation-

Kit zusammen mit der Sprache Objective-C und diversen Openstep-Tools direkt mit dem Betriebssystem zusammenarbeitet. Noch enger ist die Verbindung zwischen Betriebssystem und Grafik-API unter MS-Windows, wo kaum noch zwischen beiden unterschieden werden kann. Die Programmierung erfolgt dort meist mit Hilfe von grafischen Interface-Buildern, wo nur noch wenige Teile einer Applikation in einer klassischen Programmiersprache (C, C++, Basic) implementiert werden müssen.
Im folgenden werden anhand des X-Window-Systems typische Vokabeln der API/GUI-Programmierung dargestellt.

2.4 Das X-Window-System

2.4.1 Begriffe

Das X-Window-System wurde 1984 am MIT entworfen, seit 1987 ist es in der Version X11 verfügbar, mittlerweile als Release 6. Es werden Routinen bereitgestellt zur maschinenunabhängigen und netzwerktransparenten Formulierung von interaktiven grafischen Bedienoberflächen. Der Bildschirm wird aufgeteilt in mehrere, ggf. sich überlappende, *Fenster* (windows) mit jeweils eigenen Ein-/Ausgabefunktionalitäten. Die Fenster bilden eine baumartige Hierarchie (Kind-Fenster, Eltern-Fenster, Geschwister). Das oberste Fenster (root window) ist der Bildschirm selbst. Hierunter liegen die *Top-Level-Fenster* der einzelnen Anwendungen. Ihre Plazierung übernimmt der Window-Manager. Die Plazierung der Kinder und Enkel der Top-Level-Fenster regelt der Anwendungsprogrammierer.
X-Programme laufen auf verschiedenen Plattformen unter diversen Betriebssystemen. Das Anwendungsprogramm, genannt *X-Client*, schickt Aufträge an den *X-Server*, der die Hardware bedient.
Aus Performancegründen läuft die Client-Software häufig auf leistungsstarken Rechnern im Netz, während für den Server ein kleiner PC oder ein X-Terminal verwendet werden kann.
X-Server und X-Client können auf denselben oder auf verschiedenen Rechnern laufen. Sie kommunizieren mit Hilfe des *X-Protokolls* auf asynchrone Weise (gepuffert). Es gibt vier Arten von Nachrichten: *requests, replies, events* und *errors*.
Der X-Server hat exklusive Kontrolle über die grafischen Ausgabegeräte einschließlich Tastatur. Der Server verwaltet die *Ressourcen*, das sind komplexe Datenstrukturen für Windows, Pixmaps, Fonts, Grafik-Kontexte. Der Client referiert die Ressource über ihre ID. Dadurch wird der Kommunikationsbedarf reduziert; außerdem können mehrere Clients dieselben Ressourcen verwenden.
Der X-Client schickt Kommandos oder fragt Zustände ab. Ein Client kann Kontakt zu mehreren Servern halten. Der *Window-Manager* ist ein weiterer X-Client — und damit austauschbar. Er plaziert, verziert und ikonifiziert die Top-Level-Fenster.
Ein grafisches Ausgabegerät wird *display* genannt. Dazu gehören ein Grafik-Speicher und ggf. mehrere Bildschirme, genannt *screens*. Ein display-Name setzt sich zusammen aus

```
(<Hostname> | <Internet-Nr> ) : <Server-Nr> [.<Screen-Nr>]
```

und kann über die Environmentvariable `DISPLAY` gesetzt werden:

```
sh% export DISPLAY=magni:0
sh% export DISPLAY=131.173.12.30:0
```

Zuvor muß der angesprochene Server seine Einwilligung gegeben haben. Um z.B. dem Rechner *zeus* die Ausgabe auf dem Rechner *magni* zu erlauben, setzt man auf *magni* das Kommando ab

```
sh% xhost +zeus
```

X-Clients arbeiten ereignisorientiert, d.h., sie reagieren auf *events*, die der Server an sie schickt. Durch Event-Masken legt der Client fest, welches Fenster über welche Ereignisse informiert werden soll.

2.4.2 Architektur

Das X-Window-System ist das Fenstersystem der UNIX-Welt. X-Anwendungen verwenden Routinen aus der Xlib-Bibliothek, aus den X Toolkit Intrinsics und aus einem Widget-Set. `Xlib` enthält verschiedene Low-Level-Funktionen auf der Ebene des X-Protokolls. Z.B.

```
XDrawLine(display, window, gc, x1, y1, x2, y2);
```

zeichnet auf dem Ausgabegerät `display` im Fenster `window` unter Verwendung des Grafik-Kontextes `gc` eine Linie von Punkt `x1,y1` zu Punkt `x2,y2`.
Die X Toolkit Intrinsics, kurz *Xt Intrinsics*, sind ein Aufsatz auf die Xlib. Hiermit lassen sich die grafischen Grundobjekte, die *Widgets*, kreieren und manipulieren. *Widget* ist ein aus den Begriffen *Window* und *Gadget* zusammengesetztes Kunstwort und bedeutet Dialogobjekt. Z.B.

```
XtVaGetValues(widget, XmNwidth, &breite, XmNheight, &hoehe, NULL);
```

besorgt von einem Widget die Breite und Höhe und überträgt diese Werte in die Variablen `breite` bzw. `hoehe`.
Das X-Window-System stellt nur die Technik zur Verwaltung von grafischen Oberflächen zur Verfügung, die Toolkits hingegen bringen Verhalten ins Spiel, ein spezielles *Look and Feel* der Benutzerschnittstelle.
Ein *Widget-Set* ist eine Ansammlung von aufeinander abgestimmten Widget-Klassen, die sich durch Design und Funktionalität charakterisieren. Bekannte Beispiele für Widget-Sets sind OSF/Motif, Open Look und Athena Widgets.
In Motif gibt es z.B. einen dreidimensional aussehenden Druckknopf in der Klasse `XmPushButton` oder ein Anzeigefeld mit verschiebbarem Inhalt in der Klasse `XmScrolledWindow`. Widgets werden in Motif wie folgt klassifiziert:

Display-Widgets:	zum Anzeigen von Information z.B. `XmPushButton, XmToggleButton, XmText`
Container-Widgets:	zum Zusammenfassen der Kind-Widgets z.B. `XmForm, XmRowColumn`
Shell-Widgets:	spezielle Container-Widgets, eingesetzt als Top-Level-Fenster, stehen unter Verwaltung des Window-Managers. z.B. `XmDialogShell`

Die Widget-Klassen bilden einen Baum, bei dem die Kinder die Eigenschaften ihrer Eltern erben und erweitern. Aussehen und Verhalten der Widgets werden durch deren Ressourcen gesteuert. Z.B. besitzt ein Label-Widget die Ressourcen Widget-Größe, auszugebender Text, Font und Farbe des auszugebenden Textes. Manchen Widgets kann man die Adresse von Funktionen mitgeben. Eine solche Funktion, genannt *callback*, wird dann bei Eintreten eines bestimmten Ereignisses, z.B. Drücken eines Mausknopfes innerhalb des Widget-Fensters, aufgerufen.

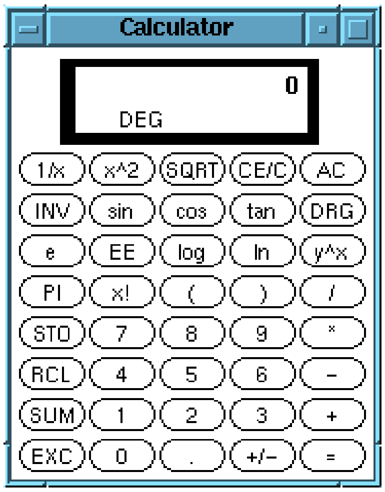
2.4.3 Ressourcen

Die wichtigsten Ressourcen:
Eine *Pixmap* ist eine rechteckige Fläche von Farbwerten im Off-Screen-Bereich, geeignet für Hintergrund und Rand eines Fensters und für gefüllte Flächen. Die Farbtiefe, *depth*, legt fest, wieviel Bits pro Farbwert zur Verfügung stehen.
Eine *Bitmap* ist eine Pixmap mit Farbtiefe 1, geeignet für Schwarzweißbilder, Cursor-Formen und Masken für Maloperationen.
Ein *Cursor* ist eine sichtbare Repräsentation der Mausposition. Seine Form wird durch eine Bitmap definiert.
Ein *Font* ist ein nicht skalierbarer Zeichensatz. Nicht jeder X-Server kennt alle Fonts. Mit dem Kommando `xfontsel` können die verfügbaren Fonts und ihr Aussehen ermittelt werden.
Ein *Grafik-Kontext* legt für ein Fenster gewisse Parameter für Zeichenoperationen fest, z.B. Vordergrundfarbe, Strichstärke, Strichart, Verknüpfungsart.
Ressourcen können gesetzt werden im Programmtext, durch Flags beim Programmstart oder durch Erwähnung in einer Ressource-Datei.

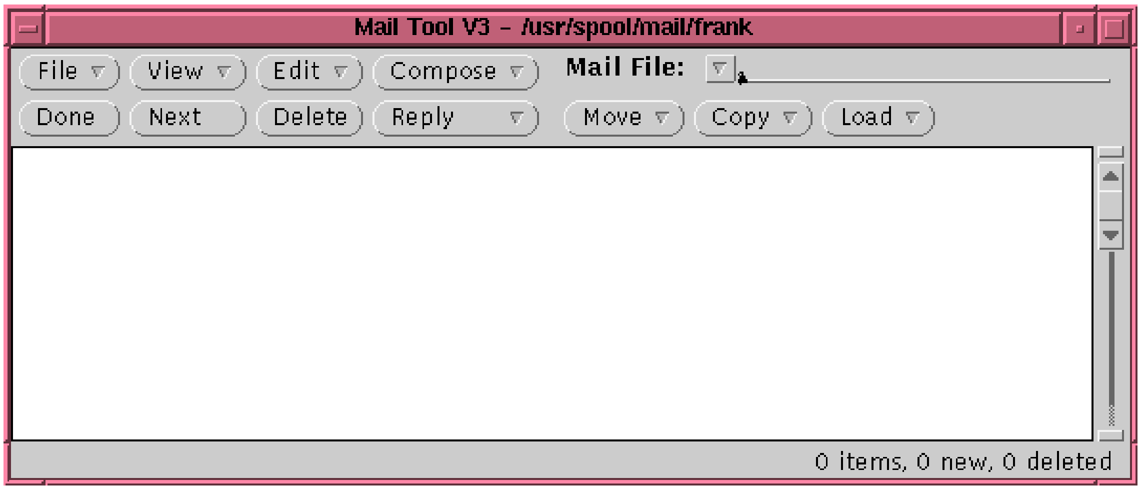
2.5 Beispiele für Widget-Sets

Im folgenden werden mit verschiedenen Widget-Sets erstellte Applikationen gezeigt.

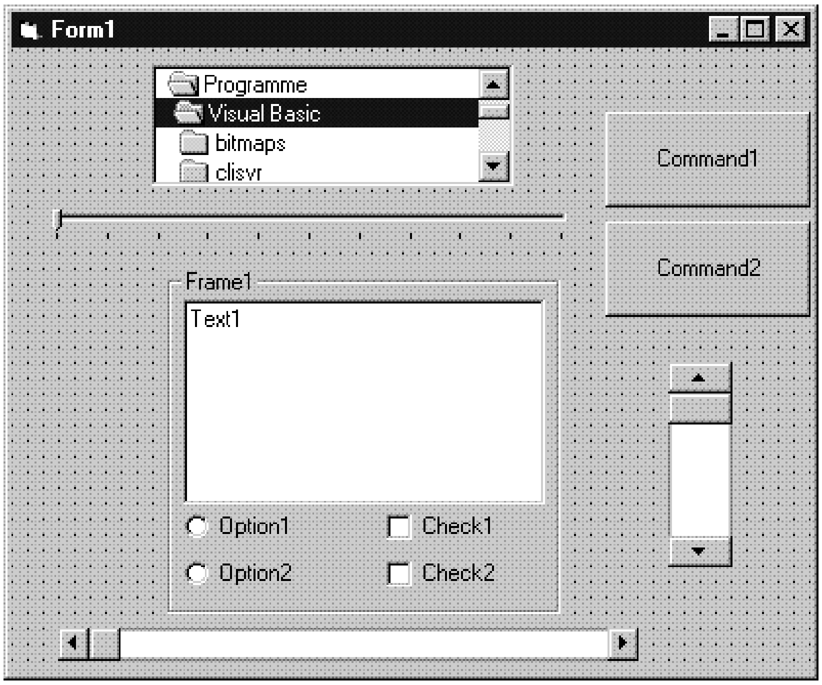
OSF/Motif: `listbox`



Athena: `xcalc`



Open Look: `mailtool`



Windows-API mit VisualBasic

2.6 AWT

Betrachtet man die verschiedenen Systeme und ihre GUIs, so stellt man fest, daß die Funktionalität bei allen relativ ähnlich ist, die Programmierumgebungen aber völlig verschieden sind. Will man für eine Applikation eine grafische Benutzerschnittstelle entwickeln, die auf allen verwendeten Systemen eine identische Funktionalität aufweist, so benötigt man eine portable Programmierumgebung. Eine solche Umgebung ist das *Abstract Window Toolkit* (AWT) der Programmiersprache Java.
Das AWT umfaßt einige der Java-Standard-Klassen, die für die portable Programmierung von GUI-Applikationen entwickelt wurden. Es heißt *abstract*, weil es kein eigenes API enthält, sondern sich auf existierende Window-Systeme abstützt. So wird auf UNIX-Systemen das Motif-Widget-Set verwendet, unter MS-Windows dagegen dessen Standard-API. Diese Unabhängigkeit vom Betriebs- bzw. Window-System ist allgemein Teil der Java-Philosophie.

2.6.1 Java

Die Programmiersprache Java wird seit 1990 von der Arbeitsgruppe Gosling bei der Firma Sun entwickelt. Als architekturunabhängige Sprache wird Java seit Aufkommen des World Wide Web als ideal geeignet für die Internetprogrammierung angesehen.
Java wird von Sun beschrieben als

- einfach,
- objektorientiert,
- verteilt,
- interpretiert,
- robust,
- sicher,
- architekturunabhängig,
- portabel,
- hochperformant,
- multithreaded,
- dynamisch.

Java hat für die Basis-Befehle die Syntax von C übernommen und diese dann um Konstrukte zur objektorientierten Programmierung erweitert. Im Unterschied zu C++, wo ANSI-C als echte Teilmenge in der Sprache enthalten ist, wurden in Java Sprachelemente entfernt, hinzugefügt und geändert.
Ein weiterer, wesentlicher Unterschied ist, daß Java eine vollständig objektorientierte Sprache ist und keine Mechanismen besitzt, um z.B. Funktionen unabhängig von Klassen und Objekten zu implementieren.
Außerdem benutzt Java keine Zeiger, also insbesondere auch keine Funktionszeiger. Da Java keine Strukturen besitzt und Felder und Zeichenketten Objekte sind, besteht kein Bedarf für

Zeiger. Java übernimmt automatisch das Referenzieren und Dereferenzieren von Objekten. Java implementiert zusätzlich eine Garbage Collection zur Speicherverwaltung.

Ein Beispiel

Das folgende Beispiel zeigt ein einfaches Java-Programm: Hello World.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Dieses Programm besteht, wie jedes Java-Programm, aus einer öffentlichen Definition. Die Klasse enthält eine Methode mit dem Namen `main()`, in der der Interpreter mit der Programmausführung beginnt.
Dieses Programm muß in einer Datei gespeichert werden, die den Namen der öffentlichen Klasse, gefolgt von der Erweiterung `.java`, besitzt. Die Übersetzung geschieht mit:

```
sh% javac HelloWorld.java
```

Dieses Kommando erzeugt die Datei `HelloWorld.class`. Um das Programm ablaufen zu lassen, wird der Java-Interpreter benutzt:

```
sh% java HelloWorld
```

Ein Programm in Java besteht im allgemeinen aus einer oder mehreren Klassendefinitionen, von denen jede in ihre eigene `.class`-Datei in *Java Virtual Machine*-Code übersetzt wird. Eine dieser Klassen muß eine `main()`-Methode definieren, in der das Programm seine Ausführung beginnt.
Der Java-Compiler erzeugt Bytecode statt direkten Maschinencode. Dieser Bytecode stellt ein architekturunabhängiges Objektdateiformat zur Verfügung, d.h., der Code ist dazu entworfen worden, Programme effizient auf verschiedene Plattformen zu transportieren. Ein Java-Programm kann auf jedem System laufen, das einen Java-Interpreter und ein Laufzeitsystem implementiert. Zusammengenommen implementieren beide die Java Virtual Machine.

2.6.2 AWT-Übersicht

In diesem Abschnitt wird eine kurze Übersicht der wesentlichen Konzepte der AWT-Klassen gegeben. Die für die Programmierung notwendigen Details sind bei Java generell der HTML-On-Line-Dokumentation zu entnehmen.
Das Basis-Paket für Grafik- und Oberflächenprogrammierung ist *java.awt*. Wie schon erwähnt, werden die dort zur Verfügung gestellten Klassen auf entsprechende Elemente eines schon vorhandenen Window-Systems abgebildet. Zu diesem Zweck gibt es das Paket *java.awt.peer*, das aber normalerweise vom Programmierer nicht direkt benutzt wird. Weitere verwandte Klassen sind z.Zt. (d.h. in der Java-Version 1.1.x) *java.awt.image* mit Klassen für die direkte Manipulation von Bildern, *java.awt.event* für das neue Eventhandling-Konzept, sowie *java.awt.datatransfer* mit Möglichkeiten zum Datenaustausch zwischen Applikationen. Erwähnt werden sollte hier auch *java.applet*, dessen einzige Klasse *Applet* verwendet wird, wenn man Java-Programme in einem HTML-Dokument benutzen will.

Die Klassen des AWT-Pakets lassen sich grob in drei Gruppen einteilen: Grafikklassen, in denen grafische Objekte wie Farben, Fonts, Bilder beschrieben werden; Komponenten, d.h. Klassen, die GUI-Komponenten wie z.B. Buttons, Menüs und Textfelder zur Verfügung stellen; sowie Layout-Manager, die die Anordnung von GUI-Komponenten in einer Applikation kontrollieren. Aus den verwandten Paketen sollen hier noch die Event-Listener besprochen werden, mit deren Hilfe die Interaktion zwischen User und Applikationen implementiert werden muß. Die folgende Übersicht erhebt keinen Anspruch auf Vollständigkeit, sondern beschreibt nur kurz die für Computergrafik-Programme wichtigen Klassen. Weitere Klassen sowie alle Details sind der entsprechenden On-Line-Dokumentation zu entnehmen.

Grafik

Die Grafik-Klassen implementieren jeweils einen bestimmten Aspekt und sind weitgehend unabhängig voneinander. Wichtige Klassen sind *Color*, *Cursor*, *Font*, *FontMetrics*, *Image*, *MediaTracker*, deren Bedeutung aus dem jeweiligen Namen ersichtlich ist. Wichtige Hilfsklassen sind *Dimension*, *Point*, *Polygon*, *Rectangle*, die jeweils entsprechende Objekte verwalten. Dabei ist zu beachten, daß alle diese Objekte keine Methoden haben, mit denen sie sich z.B. auf dem Schirm darstellen können. Diese Funktionalität wird durch die Klasse *Graphcis* bereitgestellt, in der Methoden vorhanden sind, um Bilder und andere grafische Objekte zu zeichnen, zu füllen, Farben oder Fonts zu ändern, Ausschnitte zu kopieren oder zu clippen usw. Diese abstrakte Klasse wird von diversen Komponenten bereitgestellt und kann dann verwendet werden, um das Aussehen dieser Komponente zu manipulieren.

GUI-Komponenten

Die Komponenten, mit deren Hilfe der Benutzer mit einer Oberfläche interagieren kann, sind im wesentlichen von zwei Klassen abgeleitet: *MenuComponent*, von der Klassen abgeleitet werden, mit denen Pulldown- und Popup-Menüs erzeugt werden können; und Component, von der alle anderen Komponenten abgeleitet werden. Diese kann man wiederum in zwei Gruppen einteilen. Zum einen gibt es Komponenten, die eine spezielle Funktionalität bereitstellen:

- *Button*: Es wird ein mit einem Titel versehenes Feld erzeugt, das mit der Maus angeklickt werden kann, wodurch ein entsprechender Event erzeugt wird.
- *Canvas*: Ein einfaches Feld ohne weitere Funktion, das z.B. als Zeichenfläche verwendet werden kann.
- *Checkbox*: Ein Schalter mit zwei Zuständen, der bei einem Mausklick diesen wechselt und dabei einen Event auslöst. Mehrere Checkbox-Objekte können entweder unabhängig voneinander arbeiten oder zu einer *CheckboxGroup* zusammengefaßt werden. In diesem Fall kann immer nur einer in dieser Gruppe eingeschaltet sein, so daß sich eine sogenannte *Radiobox* ergibt.
- *Choice*: Eine Gruppe von Einträgen, von der immer nur einer sichtbar ist. Das Auswählen eines Eintrags erzeugt einen entsprechenden Event.
- *FileDialog*: Diese komplexe Komponente erzeugt ein File-Listing, aus dem eine Datei ausgewählt werden muß.

- *Label*: Diese einfachste Komponente erzeugt ein Feld, in dem ein String ausgegeben wird.
- *List*: Es wird eine Liste von Strings angezeigt, aus der Einträge selektiert bzw. deselektiert werden können, was zur Erzeugung von entsprechenden Events führt.
- *Scrollbar*: Ein Rollbalken, der mit der Maus bewegt werden kann, wodurch entsprechende Events erzeugt werden.
- *TextArea*: Ein Feld, in dem ein längerer Text angezeigt und evtl. editiert werden kann.
- *TextField*: Ein Text-Feld, in dem ein einzelner String angezeigt und evtl. editiert werden kann.

Außerdem gibt es Komponenten, die keine eigene Funktionalität bereitstellen, sondern die andere Komponenten aufnehmen und anordnen können. Die wichtigsten dieser Komponenten, die von der Klasse *Container* abgeleitet werden, sind:

- *Frame*: Dieser Container erzeugt ein eigenständiges Fenster, das vom Window-Manager verwaltet wird.
- *Panel*: Ein Container, der zur Strukturierung der Komponenten innerhalb eines Fensters dient.
- *ScrollPane*: Ein Panel, das mit Scrollbars versehen ist und damit in der Lage ist, einen Ausschnitt eines größeren Feldes darzustellen.
- *Applet*: Ein Panel, das spezielle Methoden hat, die von einem Web-Browser aufgerufen werden können.

Layout-Manager

Die wesentliche Aufgabe eines Containers ist die Darstellung der darin enthaltenen Komponenten. Zur Steuerung der Anordnung dieser Komponenten werden sogenannte *Layout-Manager* verwendet. Ein solcher Layout-Manager implementiert dazu eine spezielle Layout-Politik, die mehr oder weniger konfigurierbar ist. Es gibt die folgenden vordefinierten Layout-Manager:

- *FlowLayout*: Dies ist das einfachste Layout, welches bei Panel voreingestellt ist. Jede Komponente wird in seiner bevorzugten Größe rechts von der vorigen Komponente eingefügt. Ist der rechte Rand des Containers erreicht, wird eine weitere Zeile angehängt, in der weitere Komponenten eingefügt werden können.
- *BorderLayout*: Dies ist ebenfalls ein einfaches Layout, das als Default-Layout bei *Frame* verwendet wird. Es besteht aus maximal fünf Komponenten, die an den Positionen Nord, Süd, Ost, West und Zentrum eingefügt werden können. Im Gegensatz zum *FlowLayout* werden hier die Komponenten an die Größe des Containers angepaßt: Die Nord- und Süd-Komponenten werden in X-Richtung gestreckt bzw. gestaucht; die Ost- und West-Komponenten in Y-Richtung und die Zentrums-Komponente in beiden Richtungen.

- *CardLayout*: Dies ist ein spezielles Layout, bei dem beliebig viele Komponenten eingefügt werden können, von denen aber immer nur eine zu sehen ist, da sie wie bei einem Kartenspiel übereinander angeordnet werden.

- *GridLayout*: Hier werden die Komponenten in einem zweidimensionalen Gitter angeordnet. Dabei werden alle Komponenten auf die gleiche Größe gebracht, wobei die größte Komponente die Gittergröße vorgibt.

- *GridBagLayout*: Dies ist der flexibelste Manager, mit dem beliebige Anordnungen möglich sind. Es wird wieder von einem zweidimensionalen Gitter ausgegangen, wobei hier jetzt aber eingestellt werden kann, wie viele Gitterzellen eine Komponente belegt und wie sie sich anpaßt.

Falls die Anforderungen einer speziellen Applikation mit diesen existierenden Layout-Managern nicht erfüllt werden können, besteht die Möglichkeit, weitere Manager zu implementieren. Dies ist im allgemeinen aber nicht der Fall, da mit dem *GridBagLayout* alle praktisch vorkommenden Anordnungen ermöglicht werden können.

Event-Handling

Eine offene Frage ist nun noch, wie das Aktivieren einer GUI-Komponente eine entsprechende Aktion im Benutzer-Programm auslöst. Dazu gibt es in AWT folgendes Konzept: Eine Aktion des Benutzers — wie z.B. ein Mausklick — erzeugt in der Java-Maschine einen *Event*. Je nach Art der Aktion und des Kontextes, in dem sie stattfindet, ergeben sich verschiedene Event-Typen, die wiederum verschiedene Argumente haben können. Eine Komponente kann für sie geeignete Events empfangen, indem ein sogenannter *Event-Listener* mit der Komponente verbunden wird. Für die verschiedenen Event-Typen gibt es verschiedene Listener, von denen einer oder mehrere bei einer Komponente eingetragen werden können. Ein Listener ist dabei ein Interface, in dem alle Methoden vorgegeben werden, die für eine Menge von zusammengehörigen Events notwendig sind. Die folgende Tabelle gibt eine Übersicht der verschiedenen Event-Listener:

Listener	Methoden	Komponenten
Action	actionPerformed	Button, List, MenuItem, TextField
Adjustment	adjustmentValueChanged	Scrollbar
Component	componentHidden componentMoved componentResized componentShown	Component
Container	componentAdded componentRemoved	Container
Focus	focusGained focusLost	Component
Item	itemStateChanged	Checkbox, Choice, List
Key	keyPressed keyReleased keyTyped	Component
Mouse	mouseClicked mouseEntered mouseExited mousePressed mouseReleased	Component
MouseMotion	mouseDragged mouseMoved	Component
Text	textValueChanged	TextComponent
Window	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened	Window