Multimedia mit ToolBook und Macromedia Director

Praxisorientierte Einführung in die Multimedia-Programmierung

von
Prof. Dr. Jürgen Handke MPhil,
Philipps-Universität Marburg

R. Oldenbourg Verlag München Wien 1997

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Multimedia mit ToolBook und Macromedia Director :
praxisorientierte Einführung in die Multimedia-
Programmierung / von Jürgen Handke. - München ; Wien :
Oldenbourg.
ISBN 3-486-23972-4
NE: Handke, Jürgen

Buch. - 1997

Multimedia mit ToolBook und Macromedia Director :
praxisorientierte Einführung in die Multimedia-
Programmierung / von Jürgen Handke. - München ; Wien :
Oldenbourg.
ISBN 3-486-23972-4
NE: Handke, Jürgen

CD-ROM. - 1997

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0, Internet: http://www.oldenbourg.de

Lektorat: Margarete Metzger
Herstellung: Rainer Hartl
Umschlagkonzeption: Mendell & Oberer, München
Gedruckt auf säure- und chlorfreiem Papier
Gesamtherstellung: R. Oldenbourg Graphische Betriebe GmbH, München

Inhalt

Vorwort

Die Idee zu diesem Buch entstand aus der Notwendigkeit, Lehrmaterialien für einen Kurs „Projektmanagement Multimedia" zusammenzustellen. Ziel war dabei nicht nur die Vermittlung der Grundlagen der Multimedia-Programmierung, sondern auch die Möglichkeit des Vergleichs mehrerer Multimedia-Autorensysteme. Geeignete Bücher, mit denen dieses möglich wäre, waren nicht verfügbar. Die meisten Publikationen zum Thema Multimedia-Programmierung sind bestenfalls als Einführung in die Benutzerführung der jeweiligen Programme und Erläuterung der zugrundeliegenden Programmiertechniken gedacht. Was in der Regel fehlt, sind didaktisch gut strukturierte Übungen, die in die Einzelthematiken der Multimedia-Programmierung einführen, sowie Anleitungen, wie ein komplexes Multimedia-System zunächst konzipiert und in einem zweiten Schritt in die Tat umgesetzt wird.

Das vorliegende Buch versucht, beide Aspekte zu realisieren. Es enthält umfangreiches Übungsmaterial sowie zwei komplexe Multimedia-Systeme, mit denen nicht nur die Möglichkeiten der Multimedia-Programmierung verdeutlicht, sondern auch die Vorgehensweise bei der Erstellung eines komplexen Programms illustriert wird. Das Buch kann daher als Übungsbuch sowohl kursbegleitend als auch separat zum Selbststudium verwendet werden. Der Leserkreis umfaßt auf Grund der Struktur des Buches eine große Palette, vom Novizen bis hin zu fortgeschrittenen Programmierern von Multimedia-Systemen, die hier und da einen Programmierkniff vorfinden, den sie nutzbringend einsetzen können.

Voraussetzungen für den Umgang mit dem Buch gibt es keine, lediglich der Umgang mit Windows sollte dem Leser vertraut sein.

Das Buch gliedert sich in zwei wesentliche Teile:

Teil 1: Das Autorensystem Multimedia ToolBook (Kapitel 2 und 3)

Teil 2: Das Autorensystem Macromedia Director (Kapitel 4 und 5)

Beide Teile sind so gegliedert, daß in einem ersten Kapitel zunächst der Umgang mit dem jeweiligen Autorensystem anhand zahlreicher Einzelübungen vorgestellt wird. Danach folgt ein Kapitel, in dem die Erstellung eines marktreifen Multimedia-Systems ausgehend von den Vorarbeiten bis hin zur Auslieferung auf CD-ROM erarbeitet wird.

Beiden Teilen geht im ersten Buchkapitel eine kurze Einführung in die Grundlagen der Multimedia-Programmierung sowie eine Übersicht über Programmiersprachen mit besonderer Betonung der objektorientierten Programmierung voran. Am Ende des Buches folgt ein Vergleich der beiden im Buch vorgestellten Autorensysteme sowie ein umfangreicher Anhang mit Hinweisen zu den im Buch aufgebauten Programmierprojekten.

Warum Multimedia ToolBook, warum Macromedia Director, warum nicht andere Autorensysteme?

Die Antwort ist recht einfach. Macromedia Director ist weltweit der Marktführer im Bereich Multimedia-Autorensysteme, und Multimedia ToolBook, oft als Kraftpaket unter den Multimedia-Autorensystemen bezeichnet, steht Macromedia Director in nichts nach. Mit diesen beiden Systemen lassen sich alle Multimedia-Bereiche lückenlos abdecken. Beherrscht man beide Programme, ist man darüber hinaus in der glücklichen Lage, anhand des gewünschten Einsatzgebietes eines geplanten Multimedia-Systems und auf Grund der unterschiedlichen Konzeption beider Autorensysteme zu entscheiden, welches der beiden Programme der geplanten Anwendung besser gerecht wird. Denn trotz der Mächtigkeit beider Programme gibt es beträchtliche Unterschiede in ihrer Funktionalität, die zu verschiedenen Einsatzgebieten beider Systeme führen.

Einer Reihe von Personen, ohne die dieses Buch nicht möglich geworden wäre, bin ich zu großem Dank verpflichtet. So hat mir die IAD Marburg (Gesellschaft für Informationsverarbeitung und angewandte Datentechnik mbH), insbesondere Alfons Greif und Eva Götte, nicht nur die Möglichkeit gegeben, als Kursleiter im Kurs „Multimedia-Projektmanagement" die im Buch vorgestellten Inhalte im Unterricht zu erproben, sondern ich bin in allen die erforderlichen Hard- und Software betreffenden Fragen tatkräftig unterstützt worden. Allen Teilnehmern des Kurses „Multimedia-Projektmanager" habe ich zudem für ihre Geduld, ihre Mühe, aber auch für ihre Anregungen zur Vorgehensweise beim Lehren multimedialer Programmierung zu danken. Die zahlreichen Hinweise der Kursteilnehmer sind Grundlage für die didaktische Konzeption dieses Buches.

Bei der Anfertigung verschiedener Grafiken und Videoclips, die auf der beiliegenden CD-ROM ausgeliefert werden, bin ich von meiner wissenschaftlichen Mitarbeiterin, Frauke Intemann, in herausragender Weise unterstützt worden. Ihr gebührt daher ein besonderer Dank. Darüber hinaus bin ich nicht nur Frauke Intemann, sondern auch den übrigen Mitgliedern unseres Projektteams „Syntax Interactive - ein linguistisches Lernsystem auf CD-ROM", Jacqueline Bauer, Christian Eckhardt und Claudia Handwerker für Anregungen zum Text zu Dank verpflichtet.

Ein besonderes Kapitel dieses Buches ist das Kapitel 5. Hier wird ein komplexes Multimedia-System in didaktisch aufeinander abgestimmten

Einzelschritten entwickelt: das Projekt „Jethro Tull - from Roots to Branches". Dieses System bedient sich zahlreicher Grafiken, Sounddateien und Videoclips, die normalerweise unter strenge Lizenzbedingungen fallen. Die Verwendung und Zurverfügungstellung dieser Daten auf der dem Buch beiliegenden CD-ROM ist nur dadurch möglich geworden, daß sich Ian Anderson, der Chef von Jethro Tull und einer der profiliertesten Rockmusiker unserer Zeit, persönlich dafür eingesetzt hat. Meine eigenen musikalischen Erfahrungen und meine persönlichen Kontakte zu Ian Anderson haben dazu geführt, daß die Schallplatten- und Musikverlage EMI/Köln, Chrysalis/London und Salamander/London durch den persönlichen Einsatz von Ian Anderson ihre Zustimmung zur Verwendung der Daten gegeben haben. Ich bin dafür außerordentlich dankbar.

Der größte Dank gebührt jedoch meiner Familie, insbesondere meiner Frau Heike und unserem während der Anfertigung des Buches geborenen Sohn Florian. Beide mußten während dieser Zeit so manches Mal auf mich verzichten.

Marburg, Oktober 1996

Jürgen Handke

Voraussetzungen, Konventionen und Hinweise

Alle im Buch verwendeten Beispiele und Übungen wurden auf einem Pentium 120 PC mit 16 MB RAM, einer 24-Bit Grafikkarte und einer Soundblaster-kompatiblen Soundkarte unter Windows 95 entwickelt. Um effizient mit den Übungen im Buch arbeiten zu können, sollte ein ähnlicher PC zur Verfügung stehen, der folgende Leistungsdaten nicht unterschreitet:

- mindestens 8 MB RAM,
- mindestens 200 MB freie Festplattenkapazität,
- mindestens einen 80486-Prozessor.

Zusätzlich sollte der Leser im Besitz von Multimedia ToolBook, Version 4 und Macromedia Director, Version 5 sowie den dazugehörigen Handbüchern sein.

Dem Buch liegt eine CD-ROM bei. Darauf befinden sich alle Programmierübungen, die dazugehörigen Grafiken, Texte, Sounds und Videoclips. Eine Weitergabe dieser Daten ist nicht gestattet. Nähere Angaben zur Struktur der CD-ROM befinden sich im Anhang C.

Zur besseren Lesbarkeit wurden eine Reihe von typografischen Konventionen getroffen. Diese beziehen sich insbesondere auf Programmieranweisungen und Variablen innerhalb von Texten:

- Variablen sind im Text unterstrichen, z.B. lvText.
- Prozedur- bzw. Routinennamen erscheinen kursiv, z.B. *myGetVersion*.
- Eckige Klammern [] umschließen, wenn nicht näher spezifiziert, optionale Zusätze in Programmieranweisungen.
- Spitze Klammern <> umschließen Schlüsselwörter in Programmieranweisungen.

1 Grundlagen der Multimedia-Technologie

Der Begriff „Multimedia" bezieht sich auf Computerprogramme, die „multiple mediums" kombinieren, d.h. Graphik, Text, Video, Animation und Sound. Dabei ist es unerheblich, ob ein Programm alle diese Medien verwendet oder nur einige. Als „interaktives" Multimedia-System bezeichnet man ein Programm, das die genannten Elemente enthält und zusätzlich eine Interaktion mit dem Benutzer eines solchen Systems verlangt. Dabei wird der Terminus „interaktiv" häufig übertrieben. Viele Programme werben mit diesem Begriff, obwohl sich die Interaktivität lediglich auf einen Mausklick zum Umblättern von Seiten oder zum Vorrücken im Programm beschränkt. In der Praxis bezeichnen sich viele Programme als Multimedia-Programme, wenn sie auf CD-ROM ausgeliefert werden und einige Sound- und Videosequenzen enthalten.

Echte Interaktivität und damit Multimedia-Systeme im reinen Sinn verlangen nicht nur die Reaktion des Anwenders auf Programmvorgaben, sondern auch die Kontrolle des Anwenders über den Programmablauf und die Programmsteuerung.

Folgende Hauptanwendungsgebiete sind heute für Multimedia-Systeme relevant:

- Enzyklopädien,
- Atlanten, Reiseführer,
- interaktive Informationssysteme,
- Produktkataloge,
- Lernprogramme (CBT),[1]
- Simulationen,
- Geschäftssysteme mit Datenbankunterstützung,
- Unterhaltung.

[1] CBT = computer-based training (computergestütztes Lernen)

Bevor man mit der Entwicklung eines Multimedia-Projekts beginnt, hat man eine Reihe von grundlegenden Entscheidungen zu treffen (hier eine Auswahl):

- Entwicklung eines generellen Konzepts
  - Festlegung der Zielgruppe
  - Festlegung der inhaltliche Konzeption
- Festlegung allgemeiner Anforderungen an das System
  - zeitlicher Rahmen
  - Datensammlung
- Gestaltung des Systems
- Programmierung des Systems
- Testen des Systems
- Auslieferung des Systems
- Wartung des Systems

Dieses Buch widmet sich ausschließlich dem Punkt „Programmierung". Es soll gezeigt werden, wie mit den heute gängigen Multimedia-Entwicklungswerkzeugen ein Multimedia-System erstellt wird und welche Möglichkeiten die ausgewählten Programmierwerkzeuge bieten. Die in diesem Zusammenhang wichtigste Entscheidung betrifft die Auswahl der Programmierumgebung zur Entwicklung des Multimedia-Systems. Dabei gilt es, u.a. folgende Kriterien zu beachten:

- die Entwicklungskosten für das Multimedia-System,
- den Vertriebspreis für das Multimedia-System,
- Basiskosten für die Programmierumgebung,
- eventuelle Gebühren für Runtime-Lizenzen,
- die Leistungsfähigkeit der Programmierumgebung.

Gegenwärtig bestimmen drei Programmierumgebungen den Markt für Multimedia-Anwendungen:

- die direkte Programmierung mit Assembler, C++ oder Pascal,
- das Autorensystem Multimedia ToolBook 4.0 von Asymetrix Corp.,
- das Autorensystem Macromedia Director 5.0 von Macromedia Inc.

Zwar gibt es noch weitere Produkte, z.B. CBT-Spezialist Authorware, Visual Basic oder Borland Delphi, doch kann man Multimedia ToolBook und Macromedia Director auf Grund ihrer großen Verbreitung und seit kurzem

wegen ihrer problemlosen Konvertierbarkeit in den HTML-Code des Internets als Marktführer betrachten.[1]

Wo liegen die Vor- und Nachteile der führenden Programme?

Die effizienteste Programmierung eines Multimedia-Systems in Bezug auf sein Laufzeitverhalten und die optimale Ausnutzung der Hardware erzielt man durch die direkte Programmierung, z.B. mit C++, unter Ausnutzung aller Hard- und Software-Ressourcen. Daher sollten Multimedia-Systeme, bei denen es auf Geschwindigkeit und optimale Ausnutzung des Arbeitsspeichers ankommt, direkt programmiert werden. Das allerdings verlangt lange Entwicklungszeiten und eine große Programmiererfahrung.

Gerade wegen der benötigten Erfahrung im Umgang mit Programmiersprachen wie C++ oder Pascal werden heute viele Multimedia-Anwendungen unter Zuhilfenahme sogenannter Autorensysteme erstellt. In diesem Buch sind dies die Autorensysteme Multimedia ToolBook und Macromedia Director.

Eine eindeutige Entscheidung für oder gegen eines dieser beiden Programme läßt sich nicht treffen. Macromedia Director hat sicherlich den Vorteil, daß es systemübergreifend als Apple-Autorensystem und als Windows-Autorensystem zur Verfügung steht, während Multimedia ToolBook ein reines Windows-Programm ist. Die „cross-platform"-Fähigkeit bezahlt Macromedia Director aber mit einer gegenüber Multimedia-ToolBook geringeren Funktionalität, insbesondere bei der Ausnutzung von windows-spezifischen Ressourcen und bei der Verwendung der mitgelieferten Programmierumgebung. Auch das zugrundeliegende Konzept ist unterschiedlich. Während Multimedia ToolBook wie ein Buch aufgebaut ist, verwendet Macromedia Director ein Filmkonzept mit synchronisierten Szenen. Gerade wegen dieser unterschiedlichen Funktionalität ist es nicht verwunderlich, daß die primären Einsatzgebiete beider Systeme unterschiedlich sind. Während Macromedia Director hauptsächlich zu Präsentationszwecken und Produktvorstellungen mit einem hohen Anteil von Animationssequenzen und damit einem hohen Erlebniswert dient, wird Multimedia ToolBook insbesondere für Aufgaben wie interaktive Kataloge, Schulungssoftware und Programme, in die Datenbanken und aufwendige Datenspeicherungskonzepte integriert werden, verwendet.

Beide Autorensysteme stellen umfangreiche Entwicklungswerkzeuge zur Verfügung, so daß sich der reine Programmieraufwand im Gegensatz zur direkten Programmierung in Grenzen hält. Dennoch läßt sich auch ein mit einem derartigen Autorensystem angefertigtes Multimedia-Programm nicht allein durch Bedienung eines Benutzermenüs oder durch das Ziehen von Objekten per Mausklick erstellen. Gerade wenn es um die wirkliche Ausnutzung der Möglichkeiten eines Autorensystems geht, ist eine breite Erfahrung im Umgang mit der zum Autorensystem gehörenden Programmiersprache notwendig. In

[1] HTML = Hypertext Markup Language

Multimedia ToolBook ist dies die Programmiersprache *OpenScript*, in Macromedia Director die Programmiersprache *Lingo*. Zwar sind diese Programmiersprachen in ihrer Handhabung sehr komfortabel und daher auch für Programmierneulinge leicht zu erschließen, dennoch - und das zeigt die umfangreiche Lehrerfahrung des Autoren - birgt die Programmierung in beiden Sprachen zahlreiche Probleme.

Um diese Programmierprobleme auf ein Minimum zu beschränken, geht das vorliegende Buch einen didaktisch orientierten Weg. In den beiden Kernabschnitten (Kapitel 2 und 3: Multimedia ToolBook, Kapitel 4 und 5: Macromedia Director) werden zunächst die Grundlagen der Bedienung beider Autorensysteme erläutert (Kapitel 2 für Multimedia ToolBook, Kapitel 4 für Macromedia Director). Das geschieht anhand zahlreicher kleiner Einzelübungen, die zunächst noch einen geringen Programmieraufwand erfordern. In den Folgekapiteln 3 und 5 wird dann für das jeweilige Autorensystem ein komplexes Multimedia-Programm erstellt, das die Funktionalität des Autorensystems und der integrierten Programmiersprache vorführen soll (Kapitel 3: Multimedia ToolBook, das Projekt „Der interaktive Gemüsegarten", Kapitel 5: das Projekt „Jethro Tull - from Roots to Branches"). Bei beiden Projekten wird nicht notwendigerweise Wert auf Aspekte der Bedienungsergonomie oder Prinzipien des Screendesigns gelegt. Hauptziel beider Projekte ist es, möglichst viele Funktionen in die Programme zu integrieren, um die Möglichkeiten aber auch die Unterschiede beider Systeme aufzuzeigen.

Ist eine Multimedia-Anwendung fertiggestellt und getestet, stellt sich die Frage nach der Distribution, insbesondere nach dem Datenträger, auf dem die Anwendung zur Verfügung gestellt wird. Hier gilt die Faustregel: Wenn die Datenmenge für das gesamte Programm in komprimierter Form die 20 MB-Grenze überschreitet, ist von einer Distribution auf den heute handelsüblichen Disketten abzusehen und die Herstellung einer CD-ROM vorzuziehen. Da beide in diesem Buch vorgestellten Projekte große Datenmengen mit sich bringen, wird in den Kapiteln 3 und 5 auch der Aspekt der Auslieferung des Endprodukts auf den entsprechenden Datenträgern berücksichtigt.

1.1 Grundlagen der Programmierung

Auch wenn es in einem Multimedia-Autorensystem eine Reihe von Möglichkeiten gibt, über die man komplexe Probleme lösen kann, bedarf ein professionelles Multimedia-System umfangreicher Programmierroutinen in den dazugehörigen Programmiersprachen. Die Hoffnung vieler Anwender und Entwickler, Multimedia-Systeme seien ohne großen Programmieraufwand zu realisieren, kann noch nicht erfüllt werden. Daher ist eine Auseinandersetzung mit den Programmiersprachen OpenScript (Multimedia ToolBook) und Lingo (Macromedia Director) und ein Verständnis der Verfahren bei der Computerprogrammierung auch bei einem Multimedia-Autorensystem unerläßlich.

1.1.1 Programmiersprachen

Programmiersprachen haben zum Zweck, die Kommunikation zwischen Mensch und Maschine auf eine für den Menschen mehr oder wenig bequeme und für die Maschine eindeutig und verständliche Weise zu steuern. Dabei gibt es ein Verständigungsproblem. Während für den Menschen das ideale Kommunikationsmittel die eigene Muttersprache ist, können Maschinen nur einen Code aus Nullen und Einsen, den sogenannte Dualcode verstehen. Da beide Extreme für den jeweiligen Empfänger gewaltige Probleme mit sich bringen (die menschliche Sprache ist viel zu komplex, zu vage und enthält zu viele Mehrdeutigkeiten, der Dualcode ist für Menschen kaum lesbar), hat man Wege entwickelt, die für eine Maschine lesbar und für den Menschen verständlich sind. Dabei können verschiedene Ebenen der Kommunikation und damit der Programmierung unterschieden werden:

Ebene 1: Maschinensprachen

In der Maschinensprache werden Programmbefehle als Folge von Nullen und Einsen, also im Dualcode dargestellt. Programme, die in einem solchen Code geschrieben werden, sind extrem unübersichtlich und daher auch fehleranfällig. Hier ist ein Beispiel für die Addition 3 + 4:

```
00011010          0011 0100
```
(Befehl: „Addiere“, Operanden „3“ und „4“)

Ebene 2: Assemblersprachen

Mit einem Assembler (dt. Monteur) lassen sich Maschinenbefehle übersetzen und somit einprägsamer und verständlicher darstellen. Der gleiche Additionsbefehl könnte in Assembler so lauten:

```
ADD 3,4
```

Ebene 3: Höhere Programmiersprachen

Da auch Assembler für die meisten Menschen ein nahezu unverständlicher Programmiercode ist, hat man seit den 50er Jahren zahlreiche Programmiersprachen entwickelt, die noch einen Schritt weitergehen. Sie sind nicht nur in ihrem Befehlsvorrat erweitert und damit verständlicher geworden, sondern sie sind auch für ganz bestimmte Einsatzgebiete entwickelt worden. So ist z.B. die Sprache FORTRAN (**for**mula **tran**slator) eine speziell arithmetisch ausgerichtete Sprache oder LISP (**L**ist **P**rocessor) eine Programmiersprache für Probleme in der Künstlichen Intelligenz. Es gibt verschiedene Wege die mittlerweile zahlreichen höheren Programmiersprachen zu klassifizieren (nach dem zugrundeliegenden Programmierstil: anweisungs- bzw. funktionsorientiert oder nach der Organisation der Datentypen: deklarativ oder prozedural). In

Bild 1.1 werden die wichtigsten Programmiersprachen nach ihrem Einsatzgebiet eingeteilt.

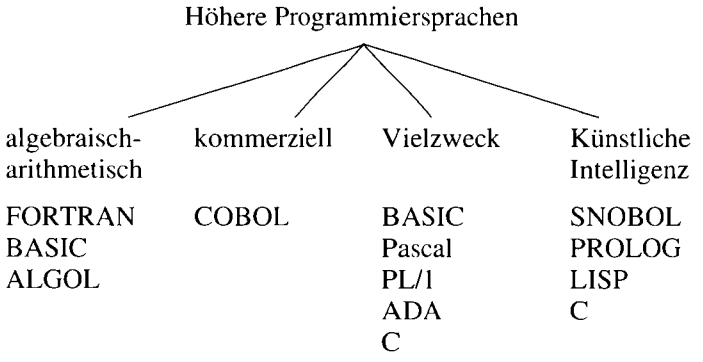

Bild 1.1: Höhere Programmiersprachen, eine Auswahl (eingeteilt nach Einsatzgebiet)

1.1.2 Objektorientierte Programmierung

Bereits seit den 80er Jahren arbeitet man an einem weitergehenden Programmierkonzept, dem der objektorientierten Programmierung. Als Basis diente dazu zunächst die Programmiersprache LISP. Heute werden viele objektorientierte Programmiersprachen, so auch OpenScript und Lingo, in C entwikkelt. Mit anderen Worten: eine objektorientierte Programmiersprache ist ein Computerprogramm, dessen Quellcode in einer anderen höheren Programmiersprache geschrieben wurde.

Grundprinzip einer objektorientierten Programmiersprache ist der Versuch, die in traditionellen Programmierumgebungen bestehende Trennung von Daten und darauf anzuwendenden Prozeduren, aufzuheben (Hansen, 1987: 334 ff). Es werden nicht mehr Verarbeitungsanweisungen (Algorithmen) für globale und lokale Datenbereiche definiert, sondern einzig Objekte. Diesen Objekten kann man über ihren Namen Nachrichten (engl. messages) zuleiten, auf die dann eine entsprechende Reaktion erfolgt (Klöppel et al., 1996: 4ff). Zusätzlich wird die objektorientierte Programmierung noch durch bestimmte Klassenhierarchien von Objekten und Vererbungsmechanismen gekennzeichnet (Welsch, 1996: 301).

Mit der LISP-basierten objektorientierten Programmiersprache SMALLTALK wurde in den 80er Jahren ein Anfang gemacht. Allerdings war damals der im Vergleich zu konventionellen höheren Programmiersprachen benötigte Speicherbedarf einer objektorientierten Programmiersprache enorm. Heute sind sowohl verfeinerte Programmierverfahren als auch größere Speichermedien verfügbar, so daß dieser Unterschied nicht mehr ins Gewicht fällt.

Da dieses Buch nicht als Grundlagenbuch zur objektorientierten Programmierung gedacht ist, sollen in der Folge lediglich die speziellen Auswirkungen

dieser noch recht jungen Programmiermethodik auf die Programmentwicklung erläutert werden.

Die Vorteile objektorientierter Programmierung zeigen sich zunächst einmal in der Einfachheit der Entwicklung eines Programmes. Folgendes Programmierproblem soll diesen Aspekt verdeutlichen:

> Immer wenn der Benutzer mit der Maus über einen bestimmten Bereich, z.B. über ein Rechteck, auf dem Bildschirm fährt, soll ein Tonsignal ausgegeben werden.

In einer herkömmlichen höheren Programmiersprache müßten zunächst folgende Probleme programmiertechnisch umgesetzt werden:

- Definition der genauen Position des Rechtecks,
- Auslesen der Mausdaten bei der Bewegung der Maus,
- Vergleich der aktuellen Mausdaten mit den Zieldaten,
- Ausgabe des gewünschten Signals.

Die enorm komplexen Lösungsschritte für dieses Problem sind in Deiß et al. (1990: 356 ff) mit LISP illustriert worden. In einer objektorientierten Umgebung stellt sich das ganze Problem wesentlich einfacher dar. Alles, was benötigt wird, ist ein Name für den Bereich, z.B. „Rechteck17", der die gewünschte Aktion veranlassen soll. Die Positionsdaten des Bereichs werden zusammen mit dem Namen gespeichert. Somit muß nur noch eine entsprechende Nachricht an das Objekt „Rechteck17" versandt werden, die die gewünschte Reaktion hervorruft, z.B.

```
mouseEnter
  beep
end
```

Weiterer Programmieraufwand ist nicht notwendig. Allein die Botschaft „mouseEnter" reicht aus, um das gewünschte Signal „beep" bei Berührung des Objekts „Rechteck17" zu veranlassen. Voraussetzung dafür ist, daß das Objekt „Rechteck17" in irgendeiner Form mit der aufgeführten Programmierroutine verbunden ist.

Eine weitere Eigenheit der objektorientierten Programmierung zeigt sich in der Art der Erstellung eines Programms. Während man von konventionellen Programmiersprachen komplette Programmlistings, d.h. mehrere Seiten aufeinanderfolgenden Programmiercodes, gewöhnt ist, stellen sich die Programme in einer objektorientierten Programmiersprache als einzelne in sich geschlossene Programmblöcke dar. Diese Module nennt man auch *Skripte*. Daher werden objektorientierte Programmiersprachen wie OpenScript oder Lingo auch häufig Skriptsprachen genannt.

Ziel dieses Buches ist nicht die Einführung in die Prinzipien der objektorientierten Programmierung, sondern das Erlernen des Umgangs mit einer solchen

Programmierumgebung. Daher wird primär Wert auf die praktische Anwendung und nicht auf die theoretische Untermauerung dieses noch recht jungen Programmierkonzepts gelegt.

1.2 Einige Hinweise zur Benutzung des Buches

Wie bereits erwähnt gliedert sich das Buch in zwei Kernteile:

- Kapitel 2 und 3: Multimedia ToolBook
- Kapitel 4 und 5: Macromedia Director

Man mag darüber argumentieren, ob dies die richtige Reihenfolge für die Einführung in die beiden Systeme ist. Auf der Basis der vorhandenen Unterrichtserfahrung spricht vieles dafür. So hat sich gezeigt, daß der Umgang mit Macromedia Director auf Grund seiner zahlreichen Fenster und des zugrundeliegenden Filmkonzepts für viele Anwender zunächst sehr gewöhnungsbedürftig ist, während Multimedia ToolBook sich in seiner Oberfläche in vielen Bereichen auch ohne große Vorkenntnisse als recht vertraut erweist.

Da beide Teile aber in sich geschlossen sind, kann der interessierte Leser alternativ auch zuerst in die Kapitel 4 und 5 einsteigen und danach den Multimedia ToolBook Teil bearbeiten. Wichtig ist in diesem Zusammenhang aber die aufeinander aufbauende Struktur der einzelnen Kapitel. Kapitel 3 baut auf Kapitel 2 auf und Kapitel 5 auf Kapitel 4. In den „Projektkapiteln" 3 („Der interaktive Gemüsegarten") und 5 („Jethro Tull - from Roots to Branches") sind zahlreiche Querverweise zu Grundlagen zu finden, die in den jeweils vorangehenden Kapiteln gelegt wurden. Daher sollten Neulinge im Umgang mit Multimedia-Autorensystemen unbedingt das Kapitel 2 vor Kapitel 3, sowie das Kapitel 4 vor Kapitel 5 lesen.

Die Kapitel 3 und 5 basieren auf einer Strategie der sukzessiven Programmerweiterung. Nehmen wir als Beispiel das Projekt „Jethro Tull - from Roots to Branches" im Kapitel 5. Dieses besteht aus insgesamt 13 Programmversionen, die aufeinander aufbauen. Während sich die erste Version mit dem Import und der Plazierung von z.B. Grafiken auf der Bühne (dem Hauptfenster in Macromedia Director) befaßt, hat z.B. die Version 10 die Integration von Fenstertechniken zum Thema. Zu allen Programmversionen liegen die entsprechenden Dateien auf der dem Buch beiliegenden CD-ROM. Für erfahrene Leser bietet sich daher ein direktes Vorgehen zu dem entsprechenden Thema und dem Dazuladen der entsprechenden Programmversion an.

Da das reine Ansehen bereits geschriebener Programme allerdings nur wenig Lerneffekt mit sich bringt, sei an dieser Stelle dringend empfohlen, alle Übungen und komplexen Systeme auf Grund der Vorgaben im Buch nachzuprogrammieren. Denn nur das eigene Programmieren - mit allen Triumphgefühlen und Fehlern - führt letztendlich zum gewünschten Lerneffekt.

2 Das Autorensystem Multimedia ToolBook

Multimedia ToolBook ist ein Autorensystem der Firma Asymetrix für die Entwicklung von Multimedia-Software, die ohne Einschränkungen vertrieben werden darf. Es ist das Multimedia-Vollprodukt zu ToolBook und umfaßt sämtliche Funktionsmerkmale von ToolBook und zusätzlich eine Reihe von benutzerfreundlichen Multimedia-Werkzeugen. Seit Anfang 1996 liegt Multimedia Toolbook in der Version 4.0 vor, Zusätzlich gibt es eine spezielle CBT-Edition (Computer-Based Training), eine erweiterte Version für die einfache Erstellung von computergestützten Schulungsanwendungen.[1]

Obwohl Multimedia ToolBook ein reines Windows-Programm ist und damit nur auf Window PCs zum Einsatz gelangt, ist es gerade im geschäftlich orientierten Multimedia-Bereich ein ungeheuer populäres Entwicklungswerkzeug für multimediale Anwendungen. Die gängigsten mit Multimedia ToolBook erstellten Anwendungen sind:

- Hypermedia-Anwendungen, wie z.B. on-line Lexika,
- interaktive Schulungsanwendungen (Computer-Based Training),
- Datenbankanwendungen, z.B. Front-Ends für externe Datenbanken,
- Spiele mit grafischen Elementen.

Die Popularität von Multimedia ToolBook begründet sich nicht nur im Programm selbst, sondern auch in den zahlreichen zusätzlichen Werkzeugen, die das Erstellen von Multimedia-Programmen extrem benutzerfreundlich machen. So gehören zum Lieferumfang:

- ein Bitmap-Editor,
- ein Icon-Editor,

[1] Basis für die Ausführungen in Kapitel 2 und 3 ist die Multimedia-Version von Asymetrix ToolBook in der Version 4.0, in der Folge Multimedia ToolBook genannt.

- ein Wave-Editor,
- ein vektor-basiertes Grafikprogramm,
- ein Menüleisteneditor,
- ein Installationsmanager,
- zahlreiche Importfilter,
- zahlreiche Hilfssysteme.

Insbesondere die fast unbegrenzte Importfähigkeit von externen Daten (Text, Sound, Animation, Video und Grafik) und Datenbanken (u.a. dBase) machen Multimedia ToolBook in seiner Funktionalität zu einem der flexibelsten verfügbaren Autorensysteme auf dem Markt. Diese Flexibilität muß Multimedia ToolBook allerdings bei manchen Anwendungen mit eingeschränktem Laufzeitverhalten bezahlen (siehe Kapitel 6).

Über die diversen Werkzeuge hinaus wurde von Asymetrix mit OpenScript eine eigene objektorientierte Programmiersprache für Multimedia ToolBook entwickelt. Erst durch den Einsatz dieser Programmierumgebung können die Möglichkeiten von Multimedia ToolBook voll ausgereizt werden.

Zur Installation von Multimedia ToolBook sind die geltenden MPC-Anforderungen völlig ausreichend (vgl. Gertler, 1995: 29; Klimsa, 1995: 228). Um einen effektiven Einsatz zu gewährleisten, sollte der Arbeitsspeicher mindestens 8 MB umfassen.

2.1 Multimedia ToolBook - Grundkonzepte

Multimedia ToolBook bietet sowohl zum Erstellen als auch zum Ausführen von Multimedia-Anwendungen eine interaktive Umgebung. Mit dieser kann man über die Zuhilfenahme von diversen Hilfsmitteln Texte bearbeiten, Grafiken erstellen und manipulieren oder Schaltflächen definieren. Mit der eingebauten Programmiersprache OpenScript läßt sich dann das Verhalten der einzelnen Elemente in einer Anwendung steuern. Bild 2.1 stellt die Entwicklungsumgebung anhand einer ausgewählten Bildschirmdarstellung dar.

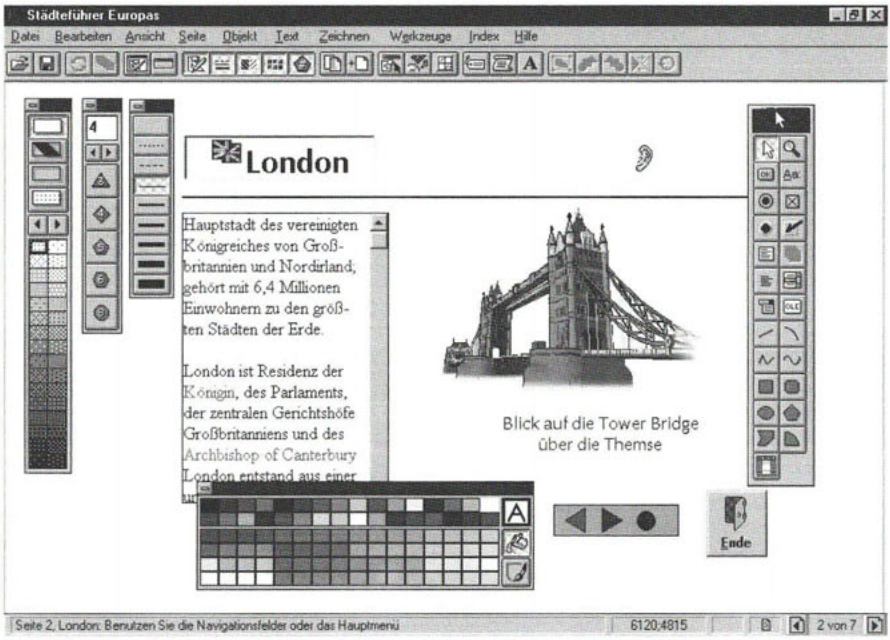

Bild 2.1: Die ToolBook Entwicklungsumgebung mit einer Anwendung

Auch dem Benutzer einer Multimedia ToolBook-Anwendung steht - wenn auch eingeschränkt - die interaktive Umgebung zur Verfügung. Diese übernimmt nicht nur die Kommunikation zwischen Benutzer und System, sondern sie steuert auch die internen Abläufe im System, so z.B. die Nutzung der Windows-Ressourcen.

Multimedia ToolBook ist auf einer Reihe von Konzepten aufgebaut, die das Verhalten des gesamten Systems leiten:

- Bücher und Seiten,
- Vorder- und Hintergrund,
- Objekt und Objekteigenschaften,
- Ereignisse und Botschaften,
- Autor und Leser,
- Skripte,
- ereignis-orientierte Programmierung.

Diese Grundkonzepte von Multimedia ToolBook werden in den folgenden Abschnitten erläutert und anhand einer Reihe von Übungen exemplarisch vorgeführt.

2.1.1 Das Buchprinzip[1]

Jede ToolBook-Anwendung kann als ein *Buch* (engl. book) angesehen werden. Ein Buch besteht aus *Seiten* (engl. pages). Die Seiten eines Buches werden in Fenstern angezeigt, die in ToolBook *Ansichtsobjekt* (engl. viewer) genannt werden.

Die Begriffe *Buch* und *Seite* sollten metaphorisch interpretiert werden, implizieren sie doch eine relativ lineare Organisationsstruktur einer Multimedia-Anwendung, die mit Multimedia ToolBook geschrieben wurde. Viele Autoren durchbrechen das Linearitätsprinzip insbesondere mit *Hyperlinks*, d.h. mit Verknüpfungen, die in bestimmten Fenstern angezeigt werden oder mit Querverweisen, die über Seiten hinweggehen.

Im Unterschied zu den Seiten in einem Buch aus Papier können die Seiten in einem ToolBook Buch verschiedene Eigenschaften (Größe, Farbe etc.) haben, oder es können mehrere Seiten zugleich angezeigt werden.

Wie viele andere Anwendungen im Grafikbereich (z.B. CorelDraw, PowerPoint etc.) unterscheidet auch Multimedia ToolBook zwischen *Vordergrund* (engl. foreground) und *Hintergrund* (engl. background). Zusammen bilden sie eine ToolBook Seite. Dabei wirkt der Vordergrund wie eine Transparenzfolie, auf der man Objekte im Vordergrund einer Seite plaziert, während der Hintergrund die gemeinsame Basis für verschiedene Vordergründe bilden kann. Der Hintergrund fungiert daher wie z.B. eine Folienvorlage im Präsentationsprogramm PowerPoint.

2.1.2 Objekte und deren Eigenschaften

Eine Seite in Multimedia ToolBook kann *Felder* (engl. fields), *Grafiken* (engl. graphic items) oder *Schaltflächen* (engl. buttons) enthalten. Jedes dieser Elemente wird als *Objekt* (engl. object) bezeichnet. Mit anderen Worten: Sämtliche visuellen Elemente in einer ToolBook-Anwendung (sogar die Seiten und Hintergründe, sowie das Buch selbst) sind Objekte. Diese werden durch zahlreiche Eigenschaften definiert, von denen die wichtigsten in der Folge näher beschrieben sind:

---

[1] Alle wichtigen Multimedia ToolBook-Begriffe werden zusätzlich - auch wenn es kleinlich erscheinen mag - in ihrer englischen Übersetzung (amerikanische Varietät) gegeben, da die englischen Begriffe Grundlage der Programmierumgebung OpenScript sind. Gleiches gilt auch für das Autorensystem Macromedia Director und die darin verwendete Programmiersprache Lingo (siehe Kapitel 4 und 5).

Identitätsnummer/Name

Alle Objekte haben *Identitätsnummern* (engl. identity number = ID), die von ToolBook automatisch zugewiesen werden. Diese IDs kann man nicht ändern. Aber anstatt sich diese IDs merken zu müssen, um Objekte ansprechen zu können, kann man ihnen auch *Namen* (engl. name) geben. Das geschieht menügesteuert oder mit einer OpenScript-Anweisung, z.B. mit

```
name of button id 0 = "Taste1"
```

Diese OpenScript-Anweisung weist der Schaltfläche (engl. button) mit der ID Nummer 0 den Namen "Taste1" zu. Damit kann die Schaltfläche nicht nur über ihre ID Nummer, sondern auch über einen Namen angesprochen werden.

Allgemeine Eigenschaften

Jedes Objekt hat einen Satz von Eigenschaften, die das Erscheinungsbild und das Verhalten des Objekts bestimmen. So können die *Abmessungen* (engl. bounds) und die *Position* (engl. position) von Objekten aber auch andere Eigenschaften wie *Farbe* (engl. color) oder *Typ* (engl. type) bestimmt werden, oder die Prinzipien, wie ein Text in einem Feld angezeigt wird (zu den Möglichkeiten der exakten Zuweisung von Eigenschaften an Objekte siehe Abschnitt 2.3.6.6).

Schichten

Ähnlich wie in Grafikprogrammen können Objekte in mehreren Schichten (engl. layer) übereinandergelegt werden, können aber - unabhängig von ihrer Ebenenposition - jederzeit erreicht werden.

Skripte

Wie bereits gezeigt können alle ToolBook-Objekte mit Programmieranweisungen der Sprache OpenScript angesprochen werden. Eine Sammlung solcher Programmieranweisungen nennt man *Skript* (engl. script = Drehbuch). Ein Skript kann aus einer oder mehreren *Behandlungsroutinen* (engl. handles) bestehen, die wiederum einzelne OpenScript-Anweisungen (engl. statements) enthalten. Das folgende Beispielskript bezieht sich auf ein fiktives Objekt namens „RoterKreis“ und besteht aus zwei Behandlungsroutinen, der Routine *mouseEnter* und der Routine *mouseLeave*. Beide Routinen bestehen aus je 3 OpenScript-Anweisungen, eine Anweisung je Zeile.[1]

---

[1]Die Prinzipien der Programmierung mit OpenScript und die relevante Terminologie werden ab Abschnitt 2.2 im Detail vorgestellt.

```
to handle mouseEnter
      sysCursor = 44
end mouseEnter

to handle mouseLeave
      sysCursor = default
end mouseLeave
```

Die Reihenfolge, mit der die Objekte durch Skripte erreicht werden, nennt man *Objekthierarchie*. Gibt es z.B. ein Skript für eine Schaltfläche, versucht Multimedia ToolBook zunächst die Schaltfläche selbst, dann die Gruppe, zu der die Schaltfläche gehört, anzusprechen, dann die Seite usw. Mit anderen Worten: Es gibt eine Hierarchie, nach der ToolBook-Elemente benachrichtigt werden:

- das ToolBook-System,
- DLLs (dynamic link libraries),
- System Bücher (Skriptsammlungen),
- das Buch (Anwendung),
- der Hintergrund einer oder mehrerer Seiten,
- die Seite,
- die Gruppe,
- gemeinsame Skripte (seit Multimedia ToolBook Version 4.0),
- das Objekt.

Dabei gilt, daß zunächst das Element angesprochen wird, für das die Routine selbst geschrieben wurde. Gibt es also eine Behandlungsroutine im Objektskript für eine Schaltfläche „Farbe", welche dieses Objekt rot färbt und eine weitere Behandlungsroutine im Seitenskript für dasselbe Objekt, die es weiß färbt, so hat dasjenige Skript Priorität, das näher am Objekt steht, also in diesem Fall das Objektskript (siehe hierzu auch die modifizierte Version von Übung 1 in Abschnitt 2.3.1).

Die Methode, Objekte über eine Hierarchie zu adressieren, erlaubt dem Programmierer eine höchst logische Anordnung von Programmierroutinen, über die Objekte erreicht werden können. Will man z.B. mit einer Routine mehrere Objekte benachrichtigen, dann sollte man dies zweckmäßigerweise von der Seite aus tun, von der aus die Objekte zu erreichen sind. Gibt es Routinen, die von überall aus erreicht werden sollen, empfiehlt sich eine Plazierung im Buch.

2.1.3 Botschaften

Bei Programmen, die mit herkömmlicher Programmiertechnik geschrieben werden, wird der Benutzer vom Programm gesteuert, d.h. Daten werden angezeigt und der Benutzer zu Eingaben aufgefordert. Multimedia ToolBook verhält sich genau entgegengesetzt und ist - wie Windows - *ereignisgesteuert* (engl. event-driven). Das bedeutet, daß eine ToolBook-Anwendung abwartet, bis der Benutzer irgendwelche *Ereignisse* (engl. events) veranlaßt.

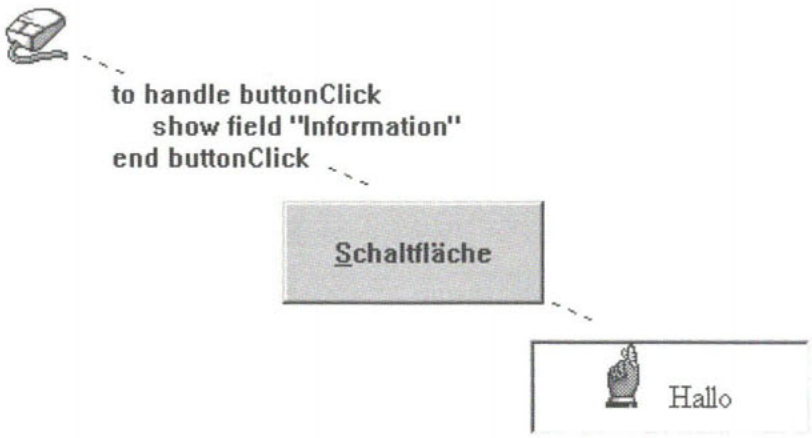

Bild 2.2: Ereignisse, Botschaften und Reaktionen

Ein Ereignis kann ein Mausklick oder Tastendruck, das Bewegen eines Objekts oder die Eingabe eines Textes sein. ToolBook übersetzt diese Ereignisse dann intern in *Botschaften* (engl. messages), die an ein Objekt versendet werden, um das Objekt über das aufgetretene Ereignis zu benachrichtigen. Wenn z.B. der Benutzer mit der Maus auf eine Schaltfläche geklickt hat, sendet ToolBook eine Botschaft an die Schaltfläche, um die Schaltfläche darüber zu informieren, daß soeben geklickt wurde und die Schaltfläche eine Reaktion hervorrufen soll, z.B. ihre Farbe verändern, ein bestimmtes Feld anzeigen soll, etc. Bild 2.2 illustriert diese Art der Informationsverarbeitung in Multimedia ToolBook.

2.1.4 Autor und Leser

Eine ToolBook-Anwendung wird auf der Autorenebene entwickelt, d.h. der Entwickler eines ToolBook-Programms ist dessen *Autor* (engl. author).

In diesem Arbeitsmodus stehen alle Zeichenhilfsmittel und Programmierwerkzeuge zur Verfügung, mit denen man neue Bücher erstellen, Objekte erzeugen und manipulieren, bzw. OpenScript-Anweisungen programmieren kann.

Der Benutzer einer ToolBook-Anwendung bedient das System von der Leserebene aus; er ist der *Leser* (engl. reader). In diesem Arbeitsmodus stehen alle zur Ausführung der Anwendung erforderlichen Werkzeuge zur Verfügung. Der Leser kann - je nach Programmierung - beliebig durch das Buch blättern, bestimmte Seiten anschauen, Text eingeben, Informationen abrufen etc. Auf die Programmierwerkzeuge kann im Normalfall jedoch nicht zugegriffen werden.

Eine marktfertige ToolBook-Anwendung ist in der Regel nur auf den Lesermodus begrenzt. Zu einer solchen Applikation gehört eine "Runtime-Version" von ToolBook, d.h. ein bis auf die Grundfunktionen abgespecktes ToolBook-Programm, mit dem die Anwendung direkt gestartet werden kann.

Während der Entwicklung eines ToolBook-Programms ist ein schneller Wechsel von der Autoren- zur Leserebene möglich.

2.2 OpenScript - Grundlagen

Zur Erstellung einer ToolBook-Anwendung verwendet man zwar zahlreiche interaktive Werkzeuge, z.B. zum Zeichnen von Objekten oder zum Erstellen von Hyperlinks. Will man jedoch alle Vorteile von ToolBook ausnutzen, muß man sich der integrierten Programmierumgebung *OpenScript* bedienen. Erst damit kann man das Verhalten einer Anwendung genauer oder direkter steuern.[1]

OpenScript ist eine professionelle Programmiersprache mit Befehlen zur Durchführung verschiedener Anweisungen, vom Erstellen neuer Objekte bis hin zur Verknüpfung von integrierten Windows-Funktionen. Trotz seiner enormen Leistungsfähigkeit ist OpenScript wegen seiner benutzerfreundlichen englischen Syntax und seiner breiten Befehlspalette einfach in der Handhabung. Mit ToolBook und OpenScript kann man ausgereifte Anwendungen in wesentlich kürzerer Zeit entwickeln, als z.B. für Entwicklungen mit C++ notwendig wäre.

---

[1] Verwendet man sowohl die interaktive Ebene zur Eingabe von Befehlen, z.B. zur Färbung eines Objekts, als auch die Möglichkeit, entsprechende Skripte dazu zu schreiben, so haben die Skripte Priorität, d.h. sie bestimmen letztlich die Eigenschaft eines Objekts.

OpenScript ist vollständig in ToolBook integriert. Man schreibt OpenScript-Programme mit einem eigenen Editor, der von ToolBook aus geladen werden kann, oder man kann Befehle direkt in ein spezielles Befehlsfenster eingeben. OpenScript-Programme werden direkt unter Multimedia ToolBook mit dem integrierten ToolBook-Compiler compiliert, und ToolBook führt die fertigen Skripte dann auf Leserebene aus. Das erleichtert das Testen und die Programmpflege erheblich.

OpenScript basiert auf dem Konzept der *strukturierten Programmierung*. Typische Eigenschaften strukturierter Programmierung sind zum einen die geringe Verwendung oder gänzliche Abwesenheit von programminternen *go to*- oder *jump*-Anweisungen. Diese werden durch sog. *Kontrollstrukturen* (z.B. durch bedingte Schleifen, siehe Abschnitt 2.3.6.1) ersetzt. Zum anderen sind die meisten Datenstrukturen, die den Ablauf eines Programms steuern, auf sich selbst beschränkt (engl. self-contained). Diese eingekapselten (engl. encapsulated) Strukturen oder Programmblöcke werden wie bereits erwähnt in OpenScript als *Behandlungsroutinen* (engl. handles) bezeichnet. Behandlungsroutinen sind damit die Basis aller OpenScript-Programme. Ausgehend vom Buch läßt sich ein OpenScript-Programm vereinfacht wie in Bild 2.3 darstellen. Mit anderen Worten: Ein OpenScript-Programm besteht aus mehreren Einzelskripten, die den verschiedenen ToolBook-Objekten zugeordnet werden. Jedes Einzelskript besteht aus einer Reihe von Behandlungsroutinen (in einer umfangreichen Multimedia-Anwendung können dies einige Hundert sein), und jede Behandlungsroutine besteht aus einer Menge von OpenScript-Anweisungen.

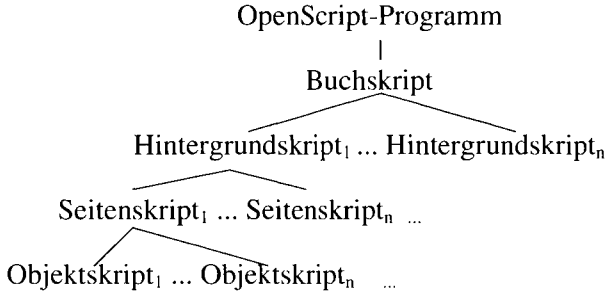

Bild 2.3: Schema eines OpenScript-Programms[1]

Eine Behandlungsroutine ist nach folgendem Schema aufgebaut:

```
to handle <botschaft> [p1,p2,..., pn]
     ...
end [<botschaft>]
```

OpenScript unterscheidet folgende Varianten von Behandlungsroutinen:

- *to handle* - eine Reaktion auf eine Botschaft,
- *to set* - eine Botschaft zur Definition von Objekteigenschaften,
- *to get* - eine Botschaft zum Abruf von Objekteigenschaften,
- *notify* - eine unerwartete Botschaft.

Das bedeutet, daß am Anfang einer Behandlungsroutine außer der Sequenz *to handle* alternativ - je nach Anforderung - auch *to get, to set* oder *notify* stehen können. Danach folgt mindestens eine Leerstelle und ein Name für die Botschaft (engl. message), die versendet werden soll. Zur Kenntlichmachung von Namen werden per Konvention die Klammern <..> verwendet. Sie gehören selbst nicht mit zum Programmiercode und sollen lediglich andeuten, daß zwischen ihnen ein bestimmtes Schlüsselwort stehen muß. Der Name der Botschaft besteht aus mehreren Zeichen und stammt entweder aus einer Menge von mehreren hundert vordefinierten und damit reservierten OpenScript-Schlüsselwörtern, z.B. *buttonClick*, oder er wird vom Benutzer selbst vergeben, z.B. *myTestButton*. Nach dem Namen kön-

[1]Zusätzlich gibt es ab Multimedia ToolBook Version 4.0 gemeinsame Skripte als Ressourcen. Außerdem können sich in dieser Programmhierarchie noch Gruppenskripte zwischen den Seiten- und Objektskripten einfügen.

nen - wiederum getrennt durch eine Leerstelle - sogenannte Parameter stehen, die ihrerseits durch Kommata getrennt sein müssen. Die Parameter werden innerhalb einer Behandlungsroutine als lokale Variablen interpretiert (siehe Abschnitt 2.3.6.3).

Nach dieser ersten, einleitenden Zeile einer Behandlungsroutine folgen beliebig viele, mehr oder wenig komplexe OpenScript-Anweisungen, eine je Zeile. Die Anweisungen werden sequentiell von oben nach unten abgearbeitet. Aus Gründen der besseren Lesbarkeit werden die einzelnen Anweisungen eingerückt. Die Behandlungsroutine wird mit dem OpenScript-Schlüsselwort *end* abgeschlossen und optional vom Namen der Nachricht gefolgt. Die eckigen Klammern [...] werden üblicherweise verwendet, um zu zeigen, daß die zwischen den Klammern stehende Sequenz auf Wunsch weggelassen werden kann.

Beim Schreiben von OpenScript-Anweisungen spielen Groß- bzw. Kleinschreibung keine Rolle, es wird aber empfohlen, zur besseren Kenntlichmachung im Skript bestimmte Konventionen einzuhalten, z.B. die interne Großschreibung von Botschaften, also *mouseEnter* statt *mouseenter*.

Jede Programmiersprache gestattet dem Programmierer die Integration von Programmkommentaren, die zum besseren Verständnis eines Programms dienen, bei der Compilierung allerdings ignoriert werden und damit speichertechnisch keine Rolle spielen. Daher wird empfohlen, möglichst viele Programmkommentare in ein Skript einzubinden. Programmkommentare werden mit einem Doppelbindestrich eingeleitet und können entweder in separaten Zeilen oder auch rechts neben OpenScript-Anweisungen geschrieben werden.

```
-- Dies ist ein Programmkommentar
-- Dies ist ein zweiter Programmkommentar

to handle enterPage  -- Dies ist ein dritter Kommentar
      ...
end enterPage
```

Sollte eine OpenScript-Anweisung zu lang für eine Zeile sein, wird der Backslash \ als Umbruchzeichen verwendet, um anzuzeigen, daß eine OpenScript-Anweisung in der nächsten Zeile fortgesetzt wird:

```
to handle buttonUp
 request "Spezielle Käseart aus Italien," && \
         "der ideal zum Reiben und Würzen ist." with \
         "Zurück"
end buttonUp
```

2.3 Der Umgang mit Multimedia ToolBook und OpenScript

Die folgenden Abschnitte dienen dazu, anhand vieler kleiner Übungen den Umgang mit Multimedia ToolBook zu erlernen. Dabei wird besonderer Wert auf die Verknüpfung von menüorientierten Techniken mit den Prinzipien der Programmierung in OpenScript gelegt. So sollen nicht nur die zahlreichen Werkzeuge, die zum Lieferumfang von Multimedia ToolBook gehören, in ihrer Funktionalität vorgestellt werden, sondern es sollen gleichzeitig die Möglichkeiten erläutert werden, die ein ToolBook Programm durch die Integration effizient gestalteter OpenScript-Routinen erhält.

Alle Übungen stehen auf der beiliegenden CD-ROM im Ordner TOOLBOOK\EXERCISE zur Verfügung. Sie haben jeweils den Dateinamen EXER-**.TBK, wobei ** für die Übungsnummer, z.B. 07, und TBK für die von Multimedia ToolBook automatisch vergebene Dateierweiterung stehen. Zum Üben wird empfohlen, auf der Festplatte des eigenen Computers einen Ordner \TOOLBOOK\EXERCISE anzulegen und dort die nachprogrammierten Übungen unter den jeweils angegebenen Dateinamen abzulegen. Diese Namen sind identisch mit den Vorlagen auf der CD-ROM.

2.3.1 Einfache Mausklicks

Beginnen wir mit einfachen Übungen, bei denen durch Mausklickaktionen Objekte ihre Eigenschaften ändern. Diese Übungen bestehen jeweils aus einer einzigen Buchseite.

In Übung 1 soll erreicht werden, daß sich ein standardmäßig weiß gefülltes Textfeld mit dem darin enthaltenen Text „Übung 1" bei einem Mausklick auf ein anderes Objekt wahlweise, rot, grün bzw. wieder weiß einfärbt. Die rote Färbung soll zustande kommen, wenn auf einen roten Kreis und die grüne Färbung, wenn auf einen grünen Kreis geklickt wird. Das Textfeld soll wieder weiß werden, wenn der Benutzer an eine andere Stelle auf der Seite klickt. Bild 2.4 zeigt einen Ausschnitt des Bildschirmaufbaus dieser Übung.

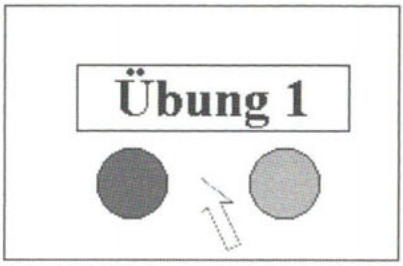

Bild 2.4: Der Bildschirmaufbau von Übung 1 (Ausschnitt)

Zur Lösung dieser ersten Übung sind eine Reihe von Vorarbeiten notwendig. Zunächst einmal muß Multimedia ToolBook vom Desktop, über das Windows Startmenü oder aber über den direkten Aufruf der Datei MTB40.EXE aus dem Ordner MTB40 geladen werden. Danach präsentiert sich das Programm auf der *Autorenebene*, d.h. auf der Ebene zur Erzeugung eines Multimedia-Programms. Demgegenüber steht die *Leserebene*, d.h. die Ebene, von der aus der Benutzer das fertige Multimedia-System bedient. Zwischen beiden Ebenen kann man über das Menü „Bearbeiten" hin- und herschalten. Da aber bei der Entwicklung eines Multimedia-Programms ein ständiger Wechsel zwischen Autoren- und Leserebene erforderlich ist, sei hier die Taste F3 empfohlen, mit der der Wechsel schneller vorgenommen werden kann.

Starten wir also Multimedia ToolBook und begeben uns auf die Autorenebene. Daß man sich auf der Autorenebene befindet, merkt man am Vorhandensein einer wesentlich komplexeren Menüleiste als auf der Leserebene, sowie daran, daß alle Autorenhilfsmittel, wie z.B. die Hilfsmittelpalette und die Schalterleiste eingeblendet werden können. Bild 2.5 zeigt die wichtigsten Autorenwerkzeuge, wobei die in der Regel horizontal unter der Menüleiste befindliche Schalterleiste hier vertikal dargestellt ist. Mit Hilfe der in Bild 2.5 abgebildeten Werkzeuge können nun die in Übung 1 gestellten Aufgaben gelöst werden.

Zunächst sollen zwei Kreise an beliebigen Positionen auf dem Bildschirm plaziert werden. Dazu wählt man aus der Hilfsmittelpalette die Option „Ellipse" aus. Zur Auswahl eines Hilfsmittels klickt man mit der Maus auf die jeweils gewünschte Option. Zusätzlich wird in der Statuszeile (das ist die Zeile ganz unten am Bildschirmrand) angezeigt, welches Hilfsmittel man ausgewählt hat. Im Fall des von uns gewünschten Ellipsenwerkzeuges erscheint die Bemerkung „Zeichnet eine Ellipse. (sysTool: ellipse)".

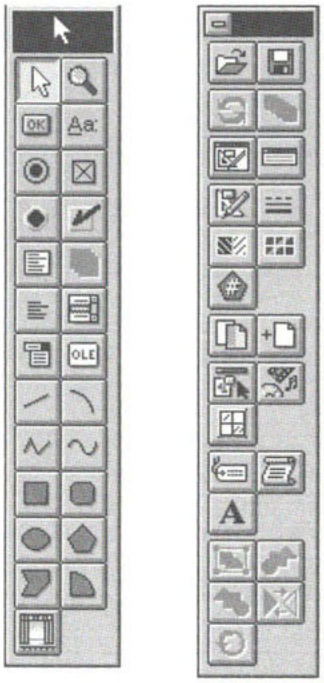

Bild 2.5: Die Autorenwerkzeuge Hilfsmittelpalette (links) und Schalterleiste (rechts)

Das tatsächliche Zeichnen der Ellipse geschieht durch Mausbewegung bei gleichzeitigem Herunterdrücken der linken Maustaste. Soll - wie in Übung 1 - anstelle der Ellipse ein Kreis erscheinen, ist gleichzeitig die STRG-Taste gedrückt zu halten. Sind die Kreise positioniert, soll der eine Kreis eine rote, der andere eine grüne Füllung erhalten.

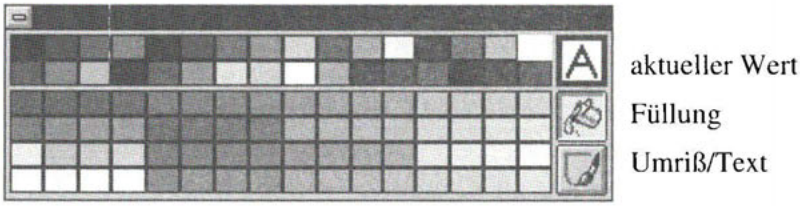

Bild 2.6: Das Werkzeug Farbpalette aus der Schalterleiste

Die menügesteuerte Farbgebung von Objekten (Füllung, Umriß) erfolgt über die Farbpalette (Bild 2.6) aus der Schalterleiste. Die Farbpalette besteht aus 32 Farbfeldern, die für Windows reserviert sind (der obere Teil)

und 64 Farbfeldern die zunächst voreingestellt sind, aber durch den Benutzer verändert und durch eigene Farben ersetzt werden können.

Zur Füllung eines Objekts markiert man dieses mit dem Mauszeiger (Statuszeile: sysTool:select), klickt anschließend auf den Farbeimer und die gewünschte Farbe. Auf diese Weise können unsere Kreise mit den gewünschten Farben gefüllt werden.

Zur besseren Bezugnahme und aus Übungszwecken sollen beide Kreise anstelle ihrer standardmäßig zugewiesenen ID-Nummer Namen erhalten. Dazu markiert man das Objekt, z.B. den roten Kreis, anschließend wählt man die Menüoption „Objekt/Grafikeigenschaften". Dieser erste Menüpunkt des Menüs „Objekt" reagiert flexibel auf die Art eines ausgewählten Objekts. Die Option „Grafikeigenschaften" bezieht sich auf Objekte, wie z.B. einen Kreis oder ein Rechteck. Bei anderen Objekten heißt dieser Menüpunkt „Bildeigenschaften", „Feldeigenschaften", „Schaltflächeneigenschaften" etc. In dem sich dann öffnenden Dialogfeld „Vordergrund-Grafikeigenschaften" kann man im Dialogfeld „Name" den Namen eingeben und dies mit „ok" bestätigen.

Nach diesem Prinzip sollen die beiden Kreise die Namen „RoterKreis" bzw. „GrünerKreis" erhalten, wobei wie bei der Eingabe von OpenScript-Routinen die Unterschiede zwischen Groß- und Kleinschreibung ignoriert werden können. Alternativ kann die Namengebung auch über den Schalter „Objekteigenschaften" in der Schalterleiste erfolgen.

Zusätzlich soll die aktuelle Seite aus Gründen der besseren Bezugnahme den Namen „Seite1" erhalten. Auch hier kann die Namengebung über die Menüoption „Objekt/Seiteneigenschaften" oder den Schalter „Objekteigenschaften" vorgenommen werden. Wichtig bei der Anwendung des Schalters „Objekteigenschaften" auf eine Seite ist, daß kein Objekt der Seite markiert sein darf.

Prinzipiell ist bei der Vergabe von Namen bis zu einer Maximallänge von 32 Zeichen jeder Name möglich. Die Namen dürfen alle Buchstaben und Zahlen, sowie einige Sonderzeichen, z.B. _ oder @ enthalten. Eine Unterscheidung in Groß- und Kleinschreibung wird dabei nicht getroffen. Auch Leerstellen sind innerhalb eines Namens erlaubt. So könnten alternative Namen für den roten Kreis „Roter Kreis", „Der rote Kreis" etc. sein. Will man allerdings Objekte über ihren Namen mit OpenScript-Behandlungsroutinen ansprechen, muß man die Namen auch in den Skripten entsprechend ausbuchstabieren. Dabei kann es sehr leicht vorkommen, daß man Leerstellen übersieht. Um diese erfahrungsgemäß häufig auftretenden Fehler auszuschließen, wird an dieser Stelle dringend empfohlen, grundsätzlich auf Leerstellen in Objektnamen zu verzichten, also Namen wie „RoterKreis" statt „Roter Kreis" oder „Der rote Kreis" zu verwenden.

Mit der Namengebung ist die Arbeit an den beiden Kreisen und an der Seite von Übung 1 fertig. Nun soll noch ein Textfeld auf die Seite gebracht werden. Dazu wählt man das Hilfsmittel „Textfeld" aus der Hilfsmittelpalette und zieht zunächst, wie bei der Erzeugung einer Ellipse, das Feld an einer beliebigen Stelle der Seite auf. Für die Eingabe einfacher Textfelder gibt es die Varianten mit oder ohne Umrandung (sysTool: Field, sysTool: borderlessField). Zu Übungszwecken soll hier zunächst ein normales Textfeld mit Umrandung gezeichnet werden. Ist dies geschehen, erscheint das Feld zunächst mit schwarzer Umrandung auf dem Bildschirm.

Die Texteingabe erfolgt nach folgenden Schritten: Zunächst wird das Feld markiert. Per Doppelklick auf das Feld kann anschließend der gewünschte Text, in unserem Fall der Text „Übung 1", eingegeben werden. Sollte dabei der Text nach links oder nach oben verschwinden, ist das Feld zunächst zu klein, kann aber nachträglich über die Haltepunkte des Feldes per Mausbewegung bei gleichzeitigem Drücken der linken Maustaste vergrößert werden.

Der Text soll anschließend nach vorheriger Markierung des Textfeldes über die Menüoption „Text/Zeichen" (alternativ durch Betätigen der Funktionstaste F6) wie folgt formatiert werden:

- Schriftart: Times New Roman
- Schriftstil: Fett
- Schriftgröße: 20

Um eine Zeichenformatierung vorzunehmen, kann entweder ein Textfeld vor der Texteingabe markiert werden oder ein bereits in einem Textfeld befindlicher Text durch Überstreichen bei gedrückter linker Maustaste markiert werden und danach die Zeichenformatierung vorgenommen werden.

Generell ist bei der Auswahl von Schriftarten Vorsicht geboten. Wählt man eine „exotische" Schriftart aus, kann es passieren, daß diese Schriftart auf einem Fremdcomputer nicht zur Verfügung steht. Zwar kann man auch Schriftarten als Ressourcen in ein Buch einbinden (siehe Abschnitt 2.3.5.5) und diese dem Anwender dann mit ausliefern, doch sollte man dies nur unter Beachtung der entsprechenden Lizenzbedingungen tun. Um in jedem Fall abgesichert zu sein, sollte man sich bei der Erstellung eines Buches auf die Standardschriftarten von Windows, z.B. auf die Schriftart „Times New Roman", beschränken.

Auf ähnliche Weise wie die Formatierung von Zeichen soll der Text „Übung 1" nun noch zentriert werden. Über die Taste F7 oder die Menüoption „Text/Absatz" kann man im Optionsfeld „Ausrichtung" die entsprechende Absatzformatierung vornehmen.

Nach Abschluß der Formatierungsarbeiten soll das Textfeld nun noch eine Reihe von Eigenschaften zugewiesen bekommen. Über das Menü „Objekt/Feldeigenschaften" sind im Dialogfeld „Vordergrund-Feldeigenschaften" folgende Eigenschaften festzulegen:

- Name: Farbänderung
- Rahmenstil: Rechteck
- Feldtyp: einzeilig
- aktiviert (Tastatur gesperrt), d.h. es sind keine Eingaben in dieses Feld möglich
- aktiviert[1]

Damit sind alle Vorarbeiten abgeschlossen, und es können nun Skripte für die einzelnen Objekte, d.h. für das Objekt „RoterKreis", für das Objekt „GrünerKreis" und für die Seite „Seite1" eingegeben werden.

Schreiben wir zunächst die beiden Objektskripte. Dazu wird zunächst das gewünschte Objekt markiert und anschließend der Skript-Editor aufgerufen. Letzteres geschieht entweder über das Menü „Objekt/Feld- bzw. Objekteigenschaften" und die Schaltfläche „Skript ..." oder direkt über das Symbol des Skript-Editors in der Schalterleiste (Meldung in der Statuszeile unten links „Skript-Editor"). Im Skript-Editor werden anschließend die gewünschten Behandlungsroutinen geschrieben. Ist das Skript erstellt, kann durch Betätigung des Skriptsymbols (Meldung in der Statuszeile unten links „Skript aktualisieren ...") die Skripterstellung abgeschlossen werden. Sollte eine Behandlungsroutine im Skript einen Fehler enthalten, reagiert der eingebaute „Debugger" (engl. debug = entlausen, technisch: Fehler finden) mit einer entsprechenden Fehlermeldung. Auf diese Weise können nun nacheinander die gewünschten Objektskripte geschrieben werden.

Basis für die Ausführung der Skripte soll ein Mausklick auf das jeweilige Objekt sein. OpenScript stellt hierzu unter anderem die Anweisung

```
buttonClick
```

zur Verfügung, die ausgeführt wird, wenn der Cursor über dem gewünschten Objekt ist und die linke Maustaste gedrückt und wieder freigegeben wird. Ähnliche OpenScript-Anweisungen sind *buttonDown* (Herunterdrücken der linken Maustaste) und *buttonUp* (Herunterdrücken und Loslassen der linken Maustaste, siehe Abschnitt 2.3.7.3). Objekte erhalten Mausbotschaften in der folgenden Reihenfolge: *buttonDown*, *buttonUp*

[1] Bis Multimedia ToolBook Version 3 hieß diese Option „verfügbar", seit Version 4.0 mißverständlicherweise ebenfalls „aktiviert".

und *buttonClick*. Wir können daher einfache Skripte schreiben, die aus Behandlungsroutinen mit der folgenden Struktur bestehen:

```
to handle buttonClick
   -- hier die entsprechende Anweisung 1
   -- hier die entsprechende Anweisung 2
   -- ...
   -- hier die entsprechende Anweisung n
end buttonClick
```

Für die Farbfüllung kann man folgende OpenScript-Anweisung einsetzen:

```
fillColor of <objekt> = <wert>
```

Anstelle der *fillColor*-Anweisung ist auch die Anweisung *RGBFill* möglich, mit der die Füllfarbe von Objekten auf der Basis der Farbanteile Rot, Grün und Blau definiert wird. Mit der *fillColor*-Anweisung (engl. fill = füllen, color = Farbe) läßt sich die Füllfarbe eines Objekts mit der Bezeichnung <objekt> in Konstanten (red, green, yellow, blue, white etc.) oder in FHS-Werten (F = Farbe, H = Helligkeit, S = Sättigung) definieren. Für unsere Zwecke reicht zunächst eine Definition der Farbwerte über die Konstanten *white*, *red* und *green* aus (weitere Konstanten befinden sich in der OpenScript-Referenz im Anhang A-26). Die gültigen Objektreferenzen <objekt> haben wir ja zum Teil schon durch die Auswahl der entsprechenden Hilfsmittel und deren Anzeige in der Statuszeile auf Autorenebene kennengelernt (z.B. sysTool: Field, sysTool: Ellipse). Weitere Objektbezeichnungen kann man aus der OpenScript-Referenz unter dem Begriff *object* entnehmen.[1] Hier sind einige Beispiele für die *fillColor*-Anweisung:

```
(1) fillColor of field "Farbänderung" = red
(2) fillColor of ellipse id 2 = white
(3) fillColor of self = yellow
```

In Beispiel (1) wird dem Objekt Textfeld (field) mit dem Namen „Farbänderung" die Farbkonstante rot (red) zugewiesen. Wichtig ist in diesem Zusammenhang, daß benutzerdefinierte Objektnamen in OpenScript-Anweisungen stets in Anführungszeichen stehen müssen. Die Anweisung in Beispiel (2) füllt das Objekt Ellipse, das ja eine Ellipse oder ein Kreis sein kann, mit der Farbe weiß (white). Die Ellipse hat keinen expliziten benutzerdefinierten Namen und wird daher über ihre ID-Nummer ID 2 angesprochen. Mit der Variante (3) wird dem Objekt, dessen Skript gerade ausgeführt wird, die Füllfarbe gelb (yellow) zugewiesen (engl. self = sich selbst).

---

[1] Im Rahmen der Übungen in Kapitel 2 und des Projekts in Kapitel 3 werden die diversen Objekttypen begleitend erläutert.

Die *fillColor*-Anweisung läßt sich nun so in die *buttonClick*-Routine der Objekte einbauen, daß das Textfeld „Farbänderung" bei einem Mausklick eine andere Füllfarbe erhält:

Das Objektskript für die Grafik „RoterKreis":

```
to handle buttonClick
      fillColor of field "Farbänderung" = red
end buttonClick
```

Das Objektskript für die Grafik „GrünerKreis":

```
to handle buttonClick
      fillColor of field "Farbänderung" = green
end buttonClick
```

Beide Skripte haben eine nahezu identische Struktur und sind selbsterklärend. Einziger Unterschied ist der Farbwert, den sie dem Feld „Farbänderung" bei einem Mausklick zuweisen. Klickt man auf den roten Kreis, wird das Feld „Farbänderung" mit roter Farbfüllung versehen, klickt man auf den grünen Kreis, wird das Feld „Farbänderung" grün.

Um bei einem Mausklick an eine andere Stelle der Seite zu veranlassen, daß das Feld „Farbänderung" wieder weiß wird, benötigen wir eine zusätzliche Routine im Seitenskript unserer bisher einzigen Buchseite:

Das Seitenskript für die Seite „Seite1":

```
to handle buttonClick
      fillColor of field "Farbänderung" = white
end buttonClick
```

Ein Seitenskript schreibt man über die Menüoption „Objekt/Seiteneigenschaften" und die dortige Schaltfläche „Skript". Alternativ kann man den Skript-Editor für das Seitenskript auch direkt aus der Schalterleiste auswählen. Man muß dabei allerdings sicherstellen, daß kein Objekt auf der jeweiligen Seite markiert ist. Um sich zu vergewissern, was für eine Art Skript gerade erstellt wird, sollte man stets die Titelzeile des Skript-Editors im Auge behalten. Dort steht jeweils „Skript für <objekt> <name>", wobei <objekt> ein Feld, eine Seite etc. sein kann und <name> der Name des Objekts oder seine ID-Nummer.

Mit der *buttonClick*-Routine im Seitenskript „Seite1" wird nun sichergestellt, daß sich das Feld „Farbänderung" wieder weiß färbt, sobald man an eine beliebige Stelle der Seite klickt. Man könnte vermuten, daß diese *buttonClick*-Routine auch für die Objekte „RoterKreis" und „GrünerKreis" gilt. Das ist aber nicht der Fall, da ein Seitenskript in der Objekthierarchie höher angesiedelt ist als ein Objektskript. Somit wird die *buttonClick*-Routine im Seitenskript nie erreicht, wenn auf einen der Kreise geklickt

wird. Lediglich durch einen Mausklick an irgendeine andere Stelle, z.B. auch auf das Feld „Farbänderung" kommt die *buttonClick*-Routine im Seitenskript zum Tragen.

Zum Ausprobieren der fertigen Version der ersten Übung sollte man nun mit der Taste F3 auf die Leserebene umschalten. Per Mausklick auf die gewünschten Stellen nimmt nun das Textfeld „Farbänderungen" die jeweils gewünschte Füllfarbe an.

Die fertige Version dieser ersten Übung erhält den Dateinamen EXER-01.TBK.

Zur Verdeutlichung der Auswirkungen der Objekthierarchie soll nun die Behandlungsroutine *buttonClick* im Objektskript „RoterKreis" vorübergehend durch zwei OpenScript-Anweisungen erweitert werden.

```
to handle buttonClick
      fillColor of field "Farbänderung" = red   -- a
      pause 1 seconds                           -- b
      forward                                   -- c
end buttonClick
```

Nach wie vor enthält die Routine in Zeile (a) die Anweisung zur Farbfüllung. In Zeile (b) wird eine Pause von 1 Sekunde veranlaßt. Die OpenScript-Anweisung

```
pause <n> seconds
```

löst eine Pause von <n> Sekunden aus, bevor der verbleibende Teil der Routine ausgeführt wird. In diesem verbleibenden Teil (c) steht die Anweisung *forward* (engl. forward = weiterleiten). Sie veranlaßt, daß die *buttonClick*-Botschaft an das nächsthöhere Objekt in der Multimedia ToolBook Objekthierarchie, also in unserem Fall an das Seitenskript, gesendet wird.

Bild 2.7: Das Weiterleiten einer Botschaft (Basis: Übung 1)

Die Veränderung des Objektskripts bewirkt, daß bei einem Mausklick auf den roten Kreis zunächst die *buttonClick*-Routine für das Objekt „RoterKreis" ausgeführt wird. Diese färbt das Textfeld „Farbänderung" rot ein und veranlaßt dann eine Pause von einer Sekunde. Durch die Weiterleitung an das Seitenskript wird danach das Textfeld „Farbänderung" durch die Ausführung der *buttonClick*-Routine im Seitenskript weiß eingefärbt. Bild 2.7 verdeutlicht diese Art der Versendung der Mausklick-Botschaft in Übung 1.

2.3.2 Das Verbergen und Anzeigen von Objekten

Ziel des folgenden Abschnitts ist die Einführung in den Umgang mit OpenScript-Anweisungen, die das Verbergen und Anzeigen von Objekten auslösen. Dazu soll - ausgehend von Übung 1 - folgende Änderung in das Programm integriert werden: Immer wenn ein Mausklick auf einen der beiden Kreise erfolgt, soll im Anschluß daran der Kreis, auf den geklickt wurde, verschwinden. Klickt man dagegen an eine beliebige andere Stelle, sollen beide Kreise sichtbar sein. Die Farbfüllung des Textfeldes „Farbänderung" bleibt davon unberührt.

Um diese zweite Übung mit möglichst wenig Aufwand zu bearbeiten, ist zunächst Übung 1 (die Datei EXER-01.TBK) über die Menüoption

„Datei/Öffnen" zu öffnen. Alternativ läßt sich eine Datei auch über die Dateiliste im Menü „Datei" öffnen. Voraussetzung dafür ist, daß sie sich unter den vier zuletzt bearbeiteten Dateien befindet. Anschließend ist die Datei über die Menüoption „Datei/Speichern unter ..." unter dem neuen Namen EXER-02.TBK abzuspeichern. An dieser neuen Datei können nun die gewünschten Modifizierungen vorgenommen werden. Diese beziehen sich ausschließlich auf die Skripte. Die bisher definierten Objekteigenschaften bedürfen keiner Änderung.

Basis für das Anzeigen und Verbergen von Objekten sind die folgenden OpenScript-Anweisungen:

```
show <objekt> <name>
hide <objekt> <name>
```

Schon mit Grundkenntnissen der englischen Sprache sind diese Anweisungen fast vollständig zu interpretieren. Die *show*-Anweisung zeigt ein Objekt des Typs <objekt> mit dem Namen <name> auf dem Bildschirm an (engl. show = zeigen). Die *hide*-Anweisung verbirgt ein Objekt des Typs <objekt> mit dem Namen <name> (engl. hide = verbergen). Hier sind einige Beispiele für die *hide*- bzw. *show*-Anweisung:

```
(1) hide field "Farbänderung"
(2) show ellipse "RoterKreis"
(3) hide self
```

Während in Beispiel (1) das Textfeld „Farbänderung" verborgen wird, führt die Anweisung in Beispiel (2) zu einer Anzeige der Ellipse bzw. des Kreises „RoterKreis" (Kreise sind ja auch Objekte des Typs Ellipse). Die Variante (3) verbirgt das Objekt, dessen Skript gerade ausgeführt wird.

Mit diesen Anweisungen können nun die Objektskripte „RoterKreis" und „GrünerKreis", sowie das Seitenskript „Seite1" ergänzt werden. Die Ergänzung bestehender Skripte erfolgt nach den gleichen Schritten wie die Neueingabe. Man markiert das Objekt, lädt den Skript-Editor entweder über die Menüoption „Objekt/**eigenschaften" oder über den Schalter „Skript-Editor" und nimmt die gewünschten Änderungen (fettgedruckt) vor.

Das Objektskript für die Grafik „RoterKreis":

```
to handle buttonClick
  fillColor of field "Farbänderung" = red-- a0
  hide self                                    -- a1
  show ellipse "GrünerKreis"                   -- a2
end buttonClick
```

Diese veränderte *buttonClick*-Routine weist dem Textfeld „Farbänderung" nach wie vor eine rote Füllung zu (Zeile a0). Zusätzlich verbirgt die

buttonClick-Routine das Objekt „RoterKreis", also „sich selbst" (a1) und zeigt das Objekt „GrünerKreis" an (a2).

Das Objektskript für die Grafik „GrünerKreis":

```
to handle buttonClick
  fillColor of field "Farbänderung" = green     -- b0
  hide self                                     -- b1
  show ellipse "RoterKreis"                     -- b2
end buttonClick
```

Analog ist die *buttonClick*-Routine für das Objekt „GrünerKreis" aufgebaut. Der einzige Unterschied besteht in der Farbfüllung für das Textfeld „Farbänderung" (b0) und in der *show*-Anweisung in Zeile (b2).

Das Seitenskript für die Seite „Seite1":

```
to handle buttonClick
  fillColor of field "Farbänderung" = white     -- c0
  show ellipse "RoterKreis"                     -- c1
  show ellipse "GrünerKreis"                    -- c2
end buttonClick
```

Die *buttonClick*-Routine im Seitenskript sollte nun ebenfalls selbsterklärend sein. Wiederum wird in Zeile (c0) dem Textfeld „Farbänderung" die gewünschte Farbe zugewiesen, anschließend werden in den Zeilen (c1) und (c2) die beiden Objekte „Roter-„ bzw. „GrünerKreis" angezeigt.

Mit diesen Erweiterungen der *buttonClick*-Routinen sind die Anforderungen an Übung 2 erfüllt. Der Vollständigkeit halber sollte man noch den Text im Feld „Farbänderung" dahingehend ändern, daß dort nun „Übung 2" erscheint.

Sind alle Änderungen durchgeführt, kann man sich mit F3 auf die Leserebene begeben und vom korrekten Ablauf des Programms überzeugen.

2.3.3 Die Ausgabe von Texten

Mit einer erneuten Erweiterung der bestehenden Skripte soll nun in Übung 3 erreicht werden, daß im Textfeld „Farbänderung" zusätzlich die Bezeichnung der Farbe als Text erscheint, also „ROT" bei einem Mausklick auf den roten und „GRÜN" bei einem Mausklick auf den grünen Kreis. Klickt man an eine andere Stelle soll der Text „Übung 3" im Textfeld erscheinen.

Basis für die Ausgabe eines Textes in einem Textfeld ist die OpenScript-Anweisung

```
text of field <name> = text
```

mit der einem Feld mit dem Namen <name> ein bestimmter Text zugewiesen wird. Dabei müssen sowohl Feldname als auch der auszugebende Text in Anführungszeichen stehen. Ausnahmen bilden hierbei Feldnamen, die als ID-Nummer behandelt werden und Texte, die als Variablen verwendet werden (siehe Abschnitt 2.3.6.2).

Mit einigen einfachen (fettgedruckten) Änderungen in den bereits in Übung 2 geschriebenen Skripten läßt sich die Textausgabe wie gewünscht erzielen.

Das Objektskript für die Grafik „RoterKreis“:

```
to handle buttonClick
      fillColor of field "Farbänderung" = red
      text of field "Farbänderung" = "ROT"
      hide self
      show ellipse "GrünerKreis"
end buttonClick
```

Das Objektskript für die Grafik „GrünerKreis“:

```
to handle buttonClick
      fillColor of field "Farbänderung" = green
      text of field "Farbänderung" = "GRÜN"
      hide self
      show ellipse "RoterKreis"
end buttonClick
```

Seitenskript für die Seite „Seite1“:

```
to handle buttonClick
      fillColor of field "Farbänderung" = white
      text of field "Farbänderung" = "Übung 3"
      show ellipse "RoterKreis"
      show ellipse "GrünerKreis"
end buttonClick
```

Die Änderungen bedürfen eigentlich keiner weiteren Erklärung. In allen drei Skripten ist eine *text of field*-Anweisung eingefügt worden, die jeweils vor den *hide*- bzw. *show*-Anweisungen die gewünschte Textausgabe vornimmt.

Das modifizierte Programm erhält den Dateinamen EXER-03.TBK.

2.3.4 Sensitive Flächen

Als sensitive Flächen kann man alle diejenigen Objekte bzw. Bereiche von Objekten verstehen, bei denen durch bloße Berührung mit dem Mauscursor eine bestimmte Aktion ausgelöst wird.

Ziel der nun folgenden Varianten der Übung 4 ist es, durch Berührung eines von zwei Textfeldern mit dem Mauscursor nach und nach folgende Aktionen auszulösen:

- Füllung eines Vielecks,
- Umwandlung der Cursorform,
- Änderung der Umrißform und -farbe des Vielecks.

Das Vieleck soll darüber hinaus nicht sichtbar sein, wenn keines der beiden Textfelder berührt wird. Bild 2.8 illustriert den Seitenaufbau einer Variante dieser Übung bei der Berührung eines der beiden Textfelder.

Die dazu notwendigen OpenScript-Techniken sind grundsätzlich bekannt. Allerdings muß zusätzlich geklärt werden, wie Cursorformen eingestellt werden, wie Vielecke erzeugt und angesprochen werden und wie die Umrißfarbe und die Umrißbreite für ein Objekt festgelegt wird.

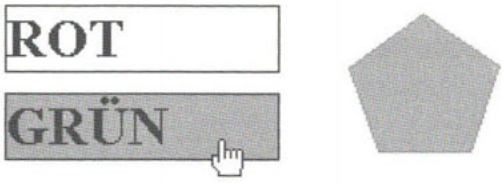

Bild 2.8: Übung 4: Mausberührung, Cursorform, Farbfüllung und Umrißvariation

2.3.4.1 Einfache Mausberührungen

Beginnen wir mit der Erzeugung eines Vielecks, in unserem Fall eines Vielecks mit fünf Ecken. Dazu bedient man sich der Option Vieleck (sysTool: polygon) aus der Hilfsmittelpalette. Die Anzahl der Ecken wird durch die Vieleckpalette aus der Schalterleiste festgelegt (Bild 2.9).

Nun können zunächst die Objekte, anschließend die Skripte für diese Übung aufgebaut werden.

Basis für die Mausberührung sind zwei gleich große Textfelder. Das eine führt den Namen „FeldRot" und bekommt die Beschriftung „ROT", das andere heißt „FeldGrün" und erhält die Beschriftung „GRÜN". Abgesehen von diesen Unterschieden sollen beide Textfelder folgende Eigenschaften haben:

- Feldtyp: einzeilig
- Rahmenstil: Rechteck
- aktiviert (Tastatur gesperrt), aktiviert

Bild 2.9: Die Vieleckpalette

Das Vieleck (ToolBook-Kennung: Polygon) hat außer dem Namen „Farbe" keine weiteren festzulegenden Eigenschaften.

Nach diesen Vorarbeiten können nun die diversen Skripte geschrieben werden. Zur Abfrage der Mausberührung bzw. des Verlassens eines Objekts mit dem Mauscursor dienen die Behandlungsroutinen:

```
mouseEnter
mouseLeave
```

Mit *mouseEnter* (engl. enter = einreten, betreten) wird das Eintreten, mit *mouseLeave* (engl. leave = verlassen) das Verlassen eines Objekts festgestellt. Damit können wir nun wie bei den bisherigen Übungen zwei Objektskripte und ein Seitenskript schreiben. Im Objektskript für die Textfelder „FeldRot" und „FeldGrün" stehen dann je eine *mouseEnter*-Routine der folgenden Art, hier als Beispiel die Routine für das Textfeld „FeldRot":

```
to handle mouseEnter
      fillColor of field "FeldRot" = red        -- a0
      show polygon "Farbe"                      -- a1
      fillColor of polygon "Farbe" = red        -- a2
end mouseEnter
```

Mit dieser Routine wird zunächst das Textfeld selbst rot eingefärbt (Zeile a0). Alternativ könnte in diesem Objektskript in Zeile (a0) auch stehen:

```
fillColor of self = red
```

Aus Gründen der Übertragbarkeit auf eine erweiterte Version belassen wir es bei der komplexeren Anweisung. In Zeile (a1) wird anschließend das Vieleck „Farbe" angezeigt und dieses ebenfalls rot eingefärbt (a2). Um das Vieleck wieder zu verbergen, nachdem der Mauscursor das Textfeld „FeldRot" verlassen hat, muß zusätzlich eine *mouseLeave*-Routine der folgenden Art im Seitenskript geschrieben werden:

```
to handle mouseLeave
      fillColor of field "FeldRot" = white
      fillColor of field "FeldGrün" = white
      hide polygon "Farbe"
end mouseLeave
```

Alternativ könnte man die *mouseLeave*-Routine auch in den Objektskripten oder auch als gemeinsames Skript für die beiden Textfelder schreiben (siehe Abschnitt 2.3.5.5).

Diese erste Version der Übung 4 erhält den Dateinamen EXER-04A.TBK.

2.3.4.2 Das Umwandeln der Cursorform

Wenden wir uns nun der zusätzlichen Umwandlung der Cursorform zu. Das Umschalten auf eine bestimmte Cursorform erfolgt mit:

```
sysCursor = <n>
```

Dabei ist <n> eine Zahl aus einer Liste systeminterner Cursorformen von Multimedia ToolBook (siehe Seite 2-379 OpenScript Referenz). Interessante Cursorformen sind z.B. die Hand (Cursorform 44), die Sanduhr (Cursorform 4) oder ein Fragezeichen (Cursorform 38). Sollen eigene Cursorformen, die über die in der Liste vordefinierten Cursor hinausgehen, in ein Buch eingebunden werden, ist eine leicht modifizierte Version der *sysCursor*-Anweisung zu wählen (siehe Abschnitt 3.14).

Mit einem einfachen Zusatz in den *mouseEnter*-Routinen in den Objektskripten (fettgedruckt) kann nun veranlaßt werden, daß sich die Standardcursorform, der Pfeil, bei Berührung des Objekts in eine Hand verwandelt (hier wieder exemplarisch die *mouseEnter*-Routine für das Objektskript des Textfeldes „FeldRot"):

```
to handle mouseEnter
      sysCursor = 44
      fillColor of field "FeldRot" = red
      show polygon "Farbe"
      fillColor of polygon "Farbe" = red
end mouseEnter
```