Technische Universität Dresden

Fakultät Wirtschaftswissenschaften

Lehrstuhl Wirtschaftsinformatik,

insbesondere Systementwicklung

User Interface für ein generisches Modellierungswerkzeug

Diplomarbeit

zur Erlangung des akademischen Grades

„Diplom-Wirtschaftsinformatiker“

Bearbeiter:	Yves Köth
	Matrikelnummer: 2417018
Betreuer:	Prof. Dr. Werner Esswein
	Steffen Greifenberg
Bearbeitungszeit:	01.10.2000 – 30.04.2001

Abstract

Bei sehr vielen Softwareprojekten ist die Entwicklung einer anspruchsvollen Benutzungsschnittstelle unverzichtbar (vgl. [VoNe98] S. 1). Dies liegt vor allem an den mit den technischen Möglichkeiten gestiegenen Erwartungen von Benutzern. Ein Softwareprodukt wird nicht nur an seiner Funktionalität sondern auch an seiner Benutzungsschnittstelle (engl. User Interface) bewertet. Umfangreiche Studien haben zu Tage gebracht, dass sich *gutes* User Interface Design kostensenkend bei der Nutzung des Produkts auswirkt. Auch in der Entwicklungsphase kann sich ein durchdachtes Design positiv auswirken, da Anpassungen bzw. Änderungen in späten Projektphasen zeitaufwendig und kostensteigernd sind.

Die Beurteilung von Oberflächen wird durch den Benutzer selbst durchgeführt. Damit übernimmt er eine zentrale Rolle beim Entwurf von Benutzungsschnittstellen. Aus diesem Grund beschäftigt sich der erste Teil der Diplomarbeit mit den psychologischen Aspekten der menschlichen Informationsverarbeitung. Darauf aufbauend befasst sich der Hauptabschnitt mit dem Entwurf von Oberflächen. Dabei wird der von der ISO geforderte benutzerorientierte Gestaltungsprozess eingehend erläutert. Es wird herausgearbeitet, welche Kriterien die Benutzbarkeit von Software sowohl positiv als auch negativ beeinflussen. Daraus werden Forderungen für die Gestaltung von Oberflächen abgeleitet. Anschließend werden Entwurfstechniken diskutiert, die den Entwickler bei der Umsetzung der Anforderungen unterstützen können. Diese sehr allgemein formulierten Empfehlungen werden in einem Style Guide konkretisiert.

Der letzte Teil der Diplomarbeit beschäftigt sich mit der Evaluierung von Benutzungsschnittstellen. In diesem Rahmen wird auch der Prototyp für das generische Modellierungswerkzeug GME 2001 beschrieben. Für seine Entwicklung werden Use case Diagramme und Klassendiagramme ausgearbeitet. Ihre Umsetzung erfolgt mit Hilfe der Entwicklungsumgebung JBuilder 4.0 Foundation. Um die Auswahl eines geeignetes Tools zu ermöglichen, wird im Vorfeld eine Analyse von vier verbreiteten Entwicklungsumgebungen durchgeführt.

In der abschließenden Zusammenfassung wird die Arbeit kritisch gewürdigt, sowie ein Ausblick auf zukünftige Trends und Entwicklungen im Bereich der Benutzungsschnittstellen gegeben.

Inhaltsverzeichnis

0 Einleitung 1

1 Ziele der Softwareergonomie 3

2 Psychologische Aspekte der MCI 6

2.1 Informationsaufnahme des Menschen 6

2.1.1 Informationsverarbeitung 6

2.1.2 Die psychologischen Gestaltgesetze 7

2.2 Mentale Modelle 9

2.3 Metaphern 10

3 Design ergonomischer Oberflächen 13

3.1 Usability Design 13

3.2 Aufgaben und Benutzeranalyse 15

3.2.1 Aufgabenanalyse 15

3.2.2 Benutzeranalyse 17

3.3 Gestaltungsgrundsätze für Dialoge 18

3.3.1 Aufgabenangemessenheit 19

3.3.2 Selbstbeschreibungsfähigkeit 20

3.3.3 Steuerbarkeit 21

3.3.4 Erwartungskonformität 22

3.3.5 Fehlertoleranz 23

3.3.6 Individualisierbarkeit 24

3.3.7 Lernförderlichkeit 25

3.4 Entwurfstechniken 25

3.4.1 Fenstersysteme 26

3.4.2 Interaktionsformen 28

3.4.2.1 Deskriptive Interaktion 28

3.4.2.2 Menüs 30

3.4.2.3 Direkte Manipulation 32

3.4.3 Antwortverhalten 34

3.4.4 Icons 36

3.4.5 WYSIWYG-Prinzip 38

3.4.6 Hilfesystem 39

3.4.7 Fehlerbehandlung 42
3.4.8 Individualisierung 44

4 Style Guide 47

4.1 Entwicklung des Style Guide 49
4.2 Beschreibung der visuellen Klassen 50
4.2.1 GMEJFrame 50
4.2.2 JPanel 50
4.2.3 JScrollPane 51
4.2.4 JTabbedPane 52
4.2.5 JSplitPane 52
4.2.6 JToolBar 53
4.2.7 Border 53
4.2.8 GMEBorder 53
4.2.9 Nutzung von Layoutmanagern 54
4.2.10 JMenu, JPopupMenu 54
4.2.11 JLabel 55
4.2.12 JButton 55
4.2.13 JTextField 56
4.2.14 JDateField 57
4.2.15 JTextArea, JEditorPane 57
4.2.16 JTextPane 57
4.2.17 JCombobox 57
4.2.18 JList 58
4.2.19 JTree 58
4.2.20 JTable 59
4.2.21 JCheckBox 59
4.2.22 JRadioButton 59
4.2.23 Tooltips 59
4.3 Icons 60

5 Evaluation ergonomischer Oberflächen 61

5.1 Prototyping 63
5.1.1 Horizontaler Prototyp 64
5.1.2 Vertikaler Prototyp 64
5.1.3 Szenario 65

5.1.4 Entwicklung des Prototypen 65
5.2 Werkzeuge 66
5.2.1 VisualAge for Java, Entry Edition 3.0 69
5.2.2 Forte for Java, Community Edition 1.0 70
5.2.3 VisualCafé 4.0 Standard Edition 72
5.2.4 JBuilder 4.0 Foundation 73
5.3 Entwicklung des Prototypen GME 2001 74
5.3.1 Beschreibung des E^3-Ansatz 75
5.3.2 Ziele des Prototypen 75
5.3.3 Spezifikation des Prototypen 76
5.3.3.1 Workspace 76
5.3.3.2 Typeneditor 78

6 Zusammenfassung und Ausblick 81

Abbildungsverzeichnis I

Abkürzungsverzeichnis II

Literaturverzeichnis III

Anhang VII

A Ergänzende Abbildungen VII
A.1 Vergleich der Entwicklungsumgebungen anhand einer Checkliste VII
A.2 Das E^3-Modell VIII
A.3 Mehrstufige Sichtenbildung im E^3-Modell VIII
A.4 Abbildung des Workspace IX
A.5 Wizards im Prototyp X
A.5.1 Unterstützung beim Anlegen eines neuen Schemas X
A.5.2 Unterstützung beim Anlegen eines neuen Projektes X
A.6 Fehlersystem im Prototyp XI
A.7 Hilfesystem im Prototyp XI
A.8 Design von Benutzungsschnittstellen XIII
A.9 Gesamtprozeß der Piktogrammentwicklung XIV
A.10 Evaluierungsprozeß von Software XV
A.11 Rating Level XV
A.12 Qualitätsmodell für Benutzungsschnittstellen (grob) XVI

B Checklisten XVII
B.1 Evaluierung von Fenstern XVII
B.1.1 Überprüfung von Struktur und Layout des GMEJFrame XVII
B.1.2 Überprüfung von Panelen (inkl. JScrollPane und JTabbedPane) ... XVII
B.1.3 Überprüfung sonstiger Bildschirmelemente XVIII
B.1.4 Verhalten von Bildschirmelementen XVIII

C Spezifikation des Prototyp XIX
C.1 Statische Sicht XIX
C.1.1 Use Case Diagramme des Prototyp GME 2001 XIX
C.1.2 Klassendiagramme des GME Prototyp XX
C.1.3 Klassendiagramm einzelner Interaktionselemente XXII
C.2 Dynamische Sicht XXIV
C.3 Detaillierte Beschreibung ausgewählter Klassen XXV
C.3.1 Spezifikation der Klasse **`Project`** XXV
C.3.2 Spezifikation der Klasse **`Task`** XXVI
C.3.3 Spezifikation der Klasse **`Model`** XXVI
C.3.4 Spezifikation der Klasse **`ChangeDescription`** XXVII
C.3.5 Alle bisher definierten Randtypen XXVIII

Ehrenwörtliche Erklärung XXIX

0 Einleitung

Das Thema Softwareergonomie hat in den letzten Jahren deutlich an Bedeutung gewonnen. Seit nunmehr fast 20 Jahren wird in Deutschland auf diesem Gebiet der Wissenschaft intensiv geforscht. Im Jahr 1993 wurde eine EG-Richtlinie veröffentlicht, die bei der Einführung von Arbeitsplatzsystemen fordert, dass das Wissen über Softwareergonomie bei der Entwicklung von Anwendungssystemen zu nutzen ist. Seither hat sich diese vergleichsweise junge Wissenschaft rasant weiterentwickelt.

Nach Anfangs wagen Definitionen von Softwareergonomie bildeten sich, abhängig von dem Interesse und der Sichtweise des Verfassers, verschiedene Ansätze heraus. Auch heute ist eine eindeutige Definition nicht möglich. EBERLE definiert **Softwareergonomie** als disziplinübergreifende Wissenschaft, die sich „speziell mit der *benutzergerechten Gestaltung der Mensch-Computer-Interaktion*“ beschäftigt ([Eber94] S. 1) . Wie in der Definition mit dem Begriff „disziplinübergreifend“ angedeutet, werden in der Softwareergonomie Teile der Disziplinen Psychologie, Organisation, Design, Statistik und Informatik genutzt. TREU definiert erweiternd **Interaktion** als die gegenseitige Beeinflussung von mindestens zwei Partnern (vgl. [Treu94] S. 23). Der Begriff umfasst damit nicht nur die eigentliche Kommunikation, sondern auch alle anderen Einflüsse. Dazu gehört beispielsweise die Erzeugung von Stress oder Zufriedenheit. Unter **Mensch-Computer-Interaktion (MCI)** wird darauf aufbauend die zweckorientierte Kombination von physischen, logischen, begrifflichen und sprach-basierten Aktionen zwischen Benutzer und Computer verstanden. Wie EBERLE bereits beschrieben hat, sollte die MCI benutzergerecht gestaltet sein. Damit wird der Fokus auf den Benutzer gesetzt. Ein **Benutzer** ist ein Individuum, welches eine Anwendungssystem nutzt. Im folgenden wird er auch als Anwender bezeichnet. Unter benutzergerechter Gestaltung wird der Entwurf von Softwaresystemen mit dem Wissen über kognitive Fähigkeiten und Beschränkungen des Benutzers beschrieben. Das Ziel besteht darin, die Produktivität zu erhöhen und eine sichere, komfortable und befriedigende Benutzung zu ermöglichen.

Um eine Interaktion zwischen Mensch und Computer überhaupt zu ermöglichen wird eine Schnittstelle benötigt. Als **Benutzungsschnittstelle** wird die gesamte Kommunikationsschnittstelle zwischen Mensch und Rechner bezeichnet (vgl. [VoNe98] S. 2). Zu ihr gehören Input-Output Geräte wie Bildschirm, Tastatur und Maus, genau so wie das Systemverhalten der Software oder die Dokumentation. Mit der **Benutzungsoberfläche**

(kurz: Oberfläche) wird in der folgenden Arbeit nur die Darstellung und das Verhalten der Software gemeint. Sie definiert den nach außen sichtbaren Anteil der Benutzungsschnittstelle. Auf die physikalischen Aspekte der Benutzungsschnittstelle wird hier nicht eingegangen. Die Darstellung der Benutzungsoberfläche gibt dem Benutzer die Möglichkeit, einen Ausschnitt des Systems zu sehen. Das Verhalten der Oberfläche ermöglicht eine Veränderung sowohl des Systems selbst, als auch der Sicht auf das System. Dieser Aspekt wird durch einen Benutzungsschnittstellenprogrammierer umgesetzt. Im Gegensatz dazu wird das Aussehen der Benutzungsoberfläche durch einen Benutzungsoberflächendesigner gestaltet. Dieser entwickelt unter Mithilfe der zukünftigen Benutzer Layout und Dialogverhalten des Anwendungssystems nach softwareergonomischen Kriterien. Die folgende Arbeit richtet sich vor allem an den zuletzt genannten Personenkreis.
In den kommenden Ausführungen werden die folgenden allgemeinen Formatkonventionen benutzt. Auf davon vorkommende Abweichungen wird explizit im Text hingewiesen.

Definition	kennzeichnet den im Satz definierten Begriff
Hervorhebung	stellt eine besondere Bedeutung des Begriffes dar
`Klasse`	bezeichnet eine Klasse
`Attribut`	bezeichnet ein Attribut einer Klasse
`Operation`	bezeichnet eine Operation einer Klasse

1 Ziele der Softwareergonomie

Das Ziel eines Softwaresystems ist die optimale Unterstützung des Benutzers bei der Aufgabenerfüllung. Die Benutzungsschnittstelle, als Bestandteil der Software muss eine effiziente und effektive Kommunikation zwischen Anwender und dem funktionalem Kern des Anwendungssystem gewährleisten. Die **Effizienz** gibt den im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzten Aufwand an, mit dem Benutzer ein bestimmtes Ziel erreichen (vgl. [ISO9241-11] S. 4). Die **Effektivität** gibt die Genauigkeit und Vollständigkeit an, mit der Benutzer ein bestimmtes Ziel erreichen.

SHNEIDERMAN nennt fünf zentrale Kriterien für den Entwurf von Benutzungsschnittstellen: Erlernbarkeit, Geschwindigkeit der Aufgabenerledigung, Fehlerrate, Erinnerungsrate und subjektive Zufriedenheit des Benutzers (vgl. [Shne98] S. 15). NIELSEN tauscht in seiner Aufzählung lediglich die Geschwindigkeit der Aufgabenerledigung mit dem Begriff Effizienz, meint aber prinzipiell das Gleiche (vgl. [Niel93] S. 26ff). Die einzelnen Kriterien beeinflussen sich gegenseitig, so dass für ihre Ausprägung immer ein Kompromiss gefunden werden muss. Die genannten Aspekte werden unter dem Begriff **Usability** (deutsch: Benutzbarkeit) zusammengefasst und können zur Evaluation von Software verwendet werden. Die Nutzung der Schnittstelle darf nicht selbst zur Aufgabe bzw. zu einem Problem werden, sondern sollte so transparent wie möglich ablaufen. Damit kann ergonomische Software in einem Unternehmen zu erheblichen Kostensenkungen führen. In diesem Bereich wurden umfangreiche Studien durchgeführt. Das Hauptproblem bei diesen ist die Messung und Zuordnung der genauen Kostensenkungen oder -erhöhungen. NIELSEN beschreibt in seinen Buch einige Beispiele (vgl. [Niel93] S. 2ff). So sparte ein australisches Versicherungsunternehmen ca. A$500.000 durch das Neudesign der Bildschirmformulare in einer Anwendung. Die Kosten des Projekts betrugen A$100.000. Durch das neue Design wurde die Fehlerrate deutlich gesenkt und damit Kosten eingespart. In vielen dieser Beispiele, die NIELSEN in seinem Buch beschreibt, werden Kosten durch Zeiteinsparung gesenkt. Ein Benutzer kann an vielen verschiedenen Stellen im Anwendungssystem Zeit sparen. So können bereits mit sinnvollen Vorgaben in Eingabefeldern erhebliche Steigerungen der Arbeitsgeschwindigkeit erzielt werden. Da viel Zeit bei der Korrektur von Fehlern verloren geht, sollten Fehlerquellen durch entsprechendes Design vermieden werden. So wurde in einem anderen Versicherungsunternehmen das eingesetzte System durch die Anwender bewertet und 130 Benutzungsprobleme festgestellt. Es ist offensichtlich, dass der

Entwicklungsprozess nicht unter Mithilfe der Benutzer vonstatten ging bzw. die Evaluation schlecht durchgeführt wurde. Viele der genannten Probleme hätten sich einfach vermeiden lassen, wären sie bekannt gewesen.

Diese Studie ist kein Einzelfall. Nach einer Einschätzung des TÜV Rheinland sind 60 Prozent der Anwender mit ihrem Computerprogramm unzufrieden. Durch Handhabungsprobleme gehen 20 Prozent der Terminal-Arbeitszeit verloren (vgl. Burm+00] S. 54), was einen enormen wirtschaftlichen Verlust darstellt. Diese Benutzungsprobleme führen nicht nur zu einem schleichenden Imageverlust und hohen Kosten der Hotline des Herstellers, es treten auch andere Schwierigkeiten auf. So ist vor allem bei der Einführung eines neuen Systems mit erheblichen Widerstand der betroffenen Personen zu rechnen. Wird dieser durch das Auftreten ergonomischer Schwächen verstärkt, kann das ein Scheitern des gesamten Projektes zur Folge haben. Das Ergebnis ist ein Anwendungssystem, dass niemand nutzt. Die Akzeptanz von Systemen steigt automatisch mit dem Grad Ihrer Ergonomie.

Während man in der Vergangenheit ca. 20 Prozent der Entwicklungszeit eines Softwaresystem für die Benutzerschnittstelle zuordnete, muss man heute mit einer inversen Verteilung rechnen (vgl. [ReMo95] S. xi). D.h., dass bis zu 80 Prozent der Entwicklungszeit für die Gestaltung der grafischen Benutzungsschnittstelle (BS) benötigt werden. Es gibt eine Vielzahl von Gründen für diese Entwicklung (vgl. [Prei99] S. 1f). Zum einen ist aufgrund der technischen Entwicklung eine grafische Oberfläche für fast jedes System realisierbar. Zum anderen haben sich die Aufgabenbereiche und damit die Benutzer geändert. Wurden früher Computersysteme ausschließlich von Experten genutzt, stehen sie heute einem breiten Benutzerkreis zur Verfügung. Die neuen Aufgabenbereiche erfordern größtenteils die im Gegensatz zur asynchronen (Batch-)Arbeitsweise stehende interaktive Kommunikation mit dem Anwendungssystem. Um am Markt bestehen zu können, muss das Softwaresystem mit einer angemessenen grafischen BS ausgestattet sein. Im Ergebnis daraus werden diese immer umfangreicher und komfortabler. Das bedeutet gleichzeitig, dass ein großer Anteil der Projektkosten in diesem Bereich der Softwareentwicklung eingesetzt werden muss. Beschleunigt man die Entwicklung der grafischen BS so sinken die Gesamtkosten des Projektes. Eine Beschleunigung kann durch verschiedenste Maßnahmen erreicht werden. Einer der wichtigsten Punkte besteht darin, nachträgliche Korrekturen zu minimieren. Wird von Anfang an auf ergonomische Aspekte geachtet, so müssen diese nicht nachträglich eingebracht werden.

Akzeptanz beim Kunden und Entwicklungskosten der Software haben entscheidenden Einfluss auf den Markterfolg eines Anwendungssystems. Es ist der Trend zu beobachten, dass Kunden vermehrt auf eine ergonomische Benutzungsschnittstelle achten. Dies liegt auch an der seit 1.1.2000 gültigen Bildschirmarbeitsverordnung, die in ihrem Anhang (Nr. 20-22) Grundsätze der Softwareergonomie darlegt (vgl. [GEAE97]). Mit Einführung europäischer Normen wie beispielsweise der ISO 9241 wurden dem Kunden auch entsprechende Werkzeuge für die Bewertung von Software zur Verfügung gestellt. In Zeiten des Outsourcing und der Profitcenter, wo kaum noch Softwareprojekte unabhängig vom Wettbewerb entwickelt werden können, gewinnen diese Aspekte immer mehr an Bedeutung.

2 Psychologische Aspekte der MCI

2.1 Informationsaufnahme des Menschen

Eine notwendige Grundlage für Oberflächenentwickler ist das Verständnis für die kognitiven Fähigkeiten des Benutzers (vgl. [Shne98] S. 20). Diese sind die Voraussetzung des Anwenders, um die vom Anwendungssystem angebotenen Informationen aufzunehmen und zu verarbeiten.

2.1.1 Informationsverarbeitung

Seit den sechziger Jahren wird in der Psychologie versucht, Verhalten und Erleben des Menschen als Informationsverarbeitung zu betrachten (vgl. [Glas94] S. 7ff). Dies wird durch die Nutzung der Computermetapher ermöglicht. Die Hardware findet ihre Analogie im physischen Organismus, dem Untersuchungsgebiet der Biologie und Physiologie, die Software entspricht dem geistigen Geschehen im Menschen, dem Gebiet der Psychologie. Die psychologische Grundlagenforschung geschieht heute überwiegend im Rahmen dieses Informationsverarbeitungsansatzes.

Auf dieser Basis wurde das Modell der Architektur der menschlichen Kognition entwickelt (vgl. Abbildung 1). In diesem steht das Individuum der Umwelt gegenüber. Die informationellen Einwirkungen der Umwelt auf den Menschen werden unter dem Begriff Wahrnehmung subsumiert. Sie stellen den Input für den Benutzer, d.h. gleichzeitig den Output eines Softwaresystems dar. Dieser Bereich ist vor allem für die Gestaltung der Benutzungsschnittstelle von Bedeutung (vgl. Abschnitt 2.1.2). Die Verarbeitung der Informationen im Exekutivsystem wird mit Hilfe des Gedächtnissystems realisiert. Dieses arbeitet unter Nutzung von mentalen Modellen. Sie sind für die Struktur einer Benutzungsschnittstelle von entscheidender Bedeutung (vgl. Abschnitt 2.2). Das Gedächtnissystem greift auf das Kurz- und Langzeitgedächtnis zurück. In verschiedenen Arbeiten wurde versucht, dass Gedächtnissystem zu quantifizieren, um direkte Aussagen über die Fähigkeiten von Benutzern treffen zu können. Damit lässt sich für verschiedene Aufgaben berechnen, wie lange der Benutzer für Ihre Erfüllung braucht und in welchem Maße das Gedächtnissystem belastet wird. Auf eine intensivere Behandlung dieses Themas wird an dieser Stelle verzichtet. Zusammenfassend lässt sich feststellen, dass das Kurzzeitgedächtnis äußerst begrenzt ist. Müssen Informationen länger gemerkt werden, muss das Gedächtnissystem dies aktiv unterstützen. Diese aktive Unterstützung, die mittels des

Exekutivsystems realisiert wird, ist jedoch so aufwendig, dass anderweitige Aufgaben nur sehr begrenzt erfüllt werden können.

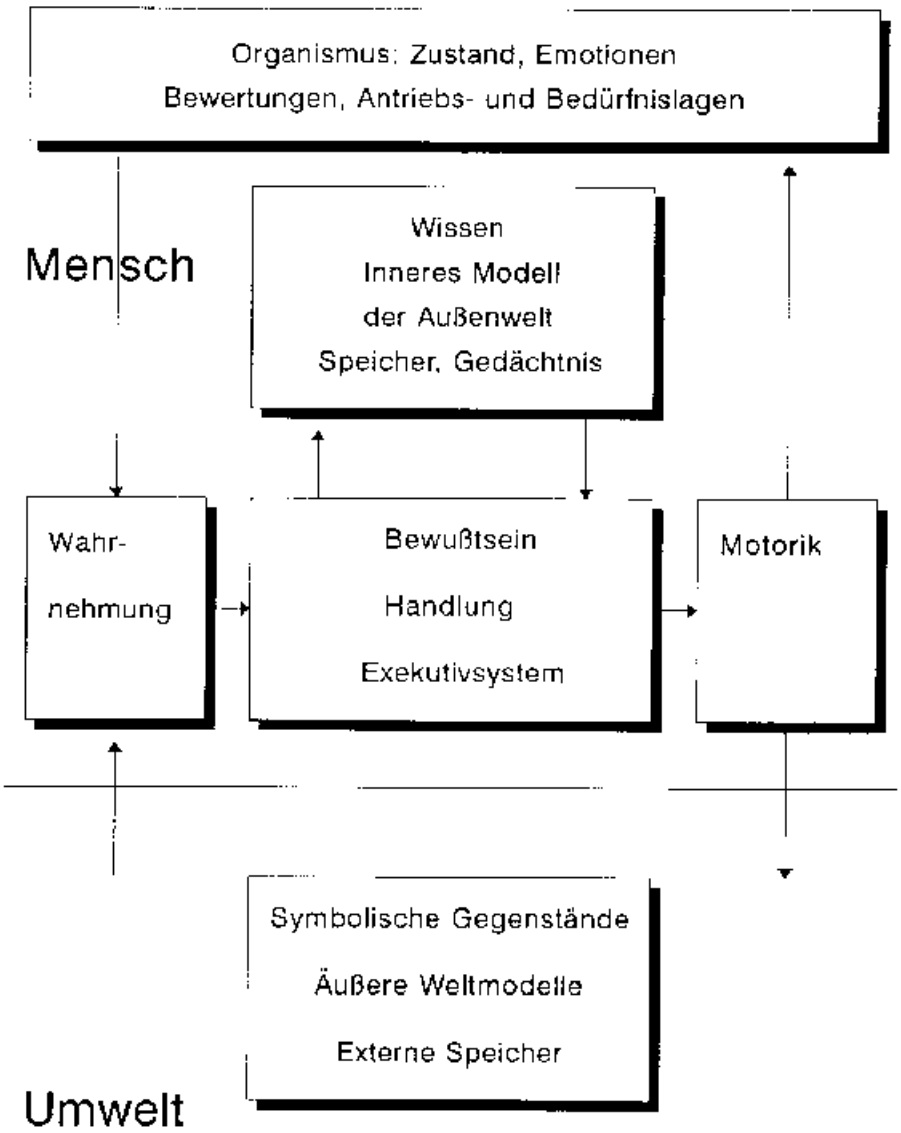


Abbildung 1 Architektur der menschlichen Kognition (vgl. [Glas94] S. 10)

Für die Softwareergonomie bedeutet das, möglichst immer alle relevanten Informationen bereit zustellen, so dass der Benutzer sein Kurzzeitgedächtnis wenig belasten muss.

2.1.2 Die psychologischen Gestaltgesetze

Der Bildschirm ist zur Zeit die wichtigste Informationsquelle bei der MCI. Das bedeutet, dass die meisten Informationen für den Benutzer visuell wahrgenommen werden. Aus diesem Grund wird im folgenden Abschnitt vor allem auf die Gestaltgesetze eingegangen, obwohl sie nur einen kleinen Teil der Wahrnehmungstheorie darstellen.

Die Gestaltgesetze sind Phänomene der visuellen Wahrnehmung, die in umfangreichen Untersuchungen herausgearbeitet wurden. Sie basieren auf der Tatsache, dass der Mensch bei der Aufnahme von visuellen Informationen das Wahrnehmungsfeld in Figur und Grund zerlegt. Alles was als Figur identifiziert wird, entspricht einem wahrgenommenen Objekt, der Grund bleibt unbeachtet. Die Frage, was als eine Figur erkannt wird sollen die Gestaltgesetze beantworten (vgl. [Glas94] S. 25ff).

Das Gesetz der Nähe als eines der bekanntesten Gestaltgesetze besagt, dass Elemente mit enger räumlicher Nähe als zusammengehörend wahrgenommen und damit als ein Objekt bzw. eine Figur erkannt werden. Ordnet man eine zweidimensionale Matrix von Punkten so an, dass der Abstand in der horizontalen Richtung größer ist als in der vertikalen, so entsteht ein Bild von mehreren Spalten. Das Gesetz der Gleichheit bzw. Ähnlichkeit besagt, dass ähnliche oder gleich aussehende Elemente schneller zum Zusammenschluss anregen als unterschiedliche Elemente. Wählt man im vorigen Beispiel gleiche Abstände zwischen den Elementen und ändert die Farbe in vertikaler Richtung, so entsteht bei der Wahrnehmung der Eindruck von Zeilen. Das Gesetz der guten Fortsetzung besagt, dass sich schneidende Konturen so interpretiert werden, dass beteiligte Linien wenn möglich nicht als geknickt erscheinen. Bei dem Gesetz der Schließung bzw. Geschlossenheit wird ausgesagt, dass nahezu geschlossene Konturen als eine Figur wahrgenommen werden, wobei das Innere die Figur und das Äußere der Grund wird. Dabei werden nicht existierende Teile einer Figur hinzugefügt. Dies gilt für alle Objektgruppen, die räumlich getrennt sind und bei denen der Betrachter eine kohärente Figur zu erkennen versucht (vgl. [ISO9241-12] S. 10). Die folgende Abbildung illustriert die beschriebenen Gesetze.

Abbildung 2 Die wichtigsten Gestaltgesetze

Die Gestaltgesetze werden oft in Kombination angewandt. Dabei folgt der Mensch dem Minimalprinzip, indem für die analytisch-geometrische Beschreibung des Wahrnehmungsfeldes so wenig wie möglich Informationen benötigt werden.

Zusammenfassend lässt sich sagen, dass der Mensch außergewöhnliche Fähigkeiten im Bereich der Wahrnehmung besitzt. Er kann beispielsweise Oberflächen bereits mit kleinsten Hinweisen wahrnehmen. Im Beispiel in Abbildung 3 glaubt man, ein Dreieck über drei Kreisen zu sehen. Sogar die Seiten des Dreiecks über dem weißen Papier scheinen sichtbar.

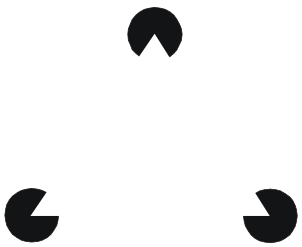


Abbildung 3 Beispiel für subjektive Konturen

Diese Fähigkeit hilft auch bei der Wahrnehmung dreidimensionaler Elemente. Auf den Gestaltgesetzen lässt sich keine geschlossene Theorie aufbauen, die den Aufbau des Bildschirminhalts beschreibt. Einige Hinweise lassen sich dennoch ableiten. So sollten Objekte auch als solche auf dem Bildschirm dargestellt werden. Dazu werden umschließende Linien und geschlossene Farbflächen genutzt. Gliedernde Elemente wie Trennlinien sollten selbst keine Figuren, sondern Grund sein. Die Bildschirmelemente können durch Abstände in logisch getrennte Bereiche gegliedert werden. Da Oberflächen und ganze Objekte auch aus Andeutungen erkannt werden, kann eine dreidimensionale Darstellung auf dem Bildschirm genutzt werden. Wichtige Objekte im Vordergrund werden vollständig dargestellt, während im Moment nicht benötigte überdeckt im Hintergrund angezeigt werden.
Der Ansatz der Gestaltgesetze gibt Aufschluss darüber, wie Objekte dargestellt werden sollen. Er gibt keine Auskunft darüber, welche Objekte wann und wo angezeigt werden sollen. Der nächste Abschnitt beschäftigt sich deshalb mit einem anderen Bereich der Psychologie, der bei diesen Fragen helfen kann.

2.2 Mentale Modelle

„Die allgemeinste begriffliche Kennzeichnung der Inhalte des Langzeitgedächtnisses besagt, daß diese **mentale Modelle** der Außenwelt in allgemeiner, semantischer (‚Häuser haben Dächer‘) oder spezieller, episodischer Form (‚Mein Haus hat zwölf Fenster‘) darstellen.“ ([Glas94] S. 43) Mit ihrer Hilfe versucht der Mensch reale Vorgänge zu erklären und steuernd darauf einzuwirken. Die kognitive Psychologie definiert mentale Modelle als Modellierung von Prozessen (vgl. [Alle97] S. 49). Dabei fasst der Mensch mehrere Schritte eines Prozesses

zusammen und sieht diese als Einheit mit bestimmten In- und Output. Der Output wird durch die mentalen Modelle „berechnet“ und entspricht bei einem realen Prozess dem vorhergesagten Ergebnis. Die einzelnen Schritte sind nicht Inhalt des mentalen Modells, sondern vielmehr unbekannt. Mentale Modelle können nur indirekt beeinflusst werden.

Ein Benutzer besitzt für die von ihm zu erledigenden Aufgaben mentale Modelle. Diese sind je nach Vertrautheit mit dem Aufgabenbereich unterschiedlich stark ausgeprägt. Um ein Anwendungssystem nutzen zu können, muss der Benutzer sich von diesem ein mentales Modell bilden. Dieses Modell ist anfangs nur sehr vage und unvollständig. Der Benutzer versucht nun, dieses unvollständige Modell ständig zu erweitern und damit das Anwendungssystem zu erfassen und zu verstehen. Für ein schnelles Einarbeiten und effizientes Arbeiten ist es günstig, wenn mentales Modell und Anwendungssystem in möglichst vielen Punkten übereinstimmen. Das heißt, dass der vorhergesagte Output des mentalen Modells des Benutzers dem tatsächlichen Output des Softwaresystems entspricht. Ist der Benutzer mit dem Aufgabenbereich nicht vertraut, kann er sich bei Bildung seines mentalen Modells am Anwendungssystem orientieren. Soll das Computersystem jedoch bereits existierende Geschäftsprozesse unterstützen, so besitzt der Benutzer für diese bereits mentale Modelle. Wird ein Anwendungssystem entwickelt, das diesen entgegensteht, kann es sehr lange dauern, bis sich der Benutzer angepasst hat. In diesem Fall ist es günstiger, das Modell des Anwendungssystem dem mentalen Modell des Benutzers anzupassen. Allerdings ist zu beachten, dass der Einsatz von Software oftmals auch als Ansatz genutzt wird, bisherige Prozesse zu analysieren und optimieren.

Grundsätzlich kann davon ausgegangen werden, dass mentales Modell und Anwendungssystem nie ganz deckungsgleich sind. Aus diesem Grund müssen Mechanismen bereitgestellt werden, die den Anwender bestmöglich beim Erlernen des Modells unterstützen. Hilfreich sind dabei Hilfesysteme und Nutzung von Gleichnissen, auch Metaphern genannt.

2.3 Metaphern

Metaphern stellen einen wichtigen Aspekt im Softwaredesign dar. **Metaphern** sind die Überführung von bekanntem Wissen eines Gebietes auf ein anderes artfremdes Gebiet (vgl. [NeCa97] S. 441). Das Ziel beim Einsatz einer Metapher besteht darin, das Anwendungssystem an ein bekanntes Referenzsystem anzulehnen. Benutzer können damit bereits bekanntes Wissen nutzen und in einem neuen Kontext anwenden. Da ein Benutzer lediglich Analogien zwischen Quell- und Zielsystem ziehen muss, wird der Lernprozess

erheblich beschleunigt. Die Quelle von Metaphern stellen bekannte Prozesse, Aufgaben und Situationen dar. Eine der bekanntesten Metaphern ist die Schreibtischmetapher. Sie beschreibt den Bildschirmaufbau des Betriebssystems als Schreibtisch. Dabei werden Objekte wie Dokumente oder Ordner so dargestellt, dass der Anwender sie sofort als solche erkennt. Auch die Funktionalität kann mittels einer Metapher ausgedrückt werden. So kann ein Benutzer ein Dokument mit der Maus direkt über den Papierkorb „ziehen“ und „loslassen“ (engl.: Drag and Drop) und damit löschen. Er hat jedoch auch die Möglichkeit, das Dokument wieder aus dem Papierkorb zurückzuholen und auf dem Schreibtisch wieder herzustellen. Metaphern werden jedoch nicht nur eingesetzt, um die Lernprozess zu beschleunigen. Durch die Nutzung von Gleichnissen ist auch eine einfachere, weil verständlichere Bedienung erreichbar.

Die Entwicklung zahlreicher Metaphern führte dazu, Klassifizierungen einzuführen. HUTCHINS unterscheidet drei Klassen (vgl. [NeCa97] S. 444f). Metaphern die an den Zielen des Benutzers ausgerichtet sind, zählt er zu den aktivitätsorientierten Metaphern. Die zweite Klasse beschreibt die grundlegende Interaktion zwischen Benutzer und Computer. Die aufgabenorientierten Metaphern beschreiben dagegen, wie eine Aufgabe strukturiert ist. MARCUS unterscheidet lediglich zwischen den organisierenden und vorgangsbeschreibenden Metaphern. Unter Organisation versteht er Strukturen, Klassen, Objekte und Attribute. Zu beschreibenden Vorgänge sind Prozesse, Aktionen und Algorithmen. Es wurden weitere Klassifizierungen entwickelt, die den Oberflächenentwickler bei der Auswahl der passenden Metapher unterstützen sollen. Dies ist vor allem vor dem Hintergrund hilfreich, dass Metaphern selten einzeln, sondern meist in Kombination genutzt werden (vgl. [Prei99] S. 177).

Der Einsatz von Metaphern birgt jedoch auch Gefahren. Aus diesem Grund muss bei ihrem Einsatz auf Konsistenz geachtet werden. Unterscheiden sich die Annahmen und Erwartungen des Benutzers von der in der Software bereitgestellten Funktionalität entstehen Probleme. Die Metapher ist in diesem Fall irreführend und beeinträchtigt die Effizienz der Aufgabenerledigung. Es lassen sich drei verschiedene Konfliktbereiche identifizieren (vgl. [Prei99] S. 167). Das erste Problem kann darin liegen, dass das Anwendungssystem Funktionalität bereitstellt, die in der Analogie nicht möglich ist und somit nicht erfasst wird. Wählt man für ein Textverarbeitungsprogramm die Metapher der Schreibmaschine, so ist dem Benutzer die Funktion des automatischen Inhaltsverzeichnisses unbekannt. Die Metapher unterstützt den Anwender auch nicht, diese Funktionalität zu erforschen. Solche Aspekte müssen damit durch zusätzliche Mechanismen, wie beispielsweise Hilfesysteme dem

Benutzer zugänglich gemacht werden. Der zweite Problembereich kann umgekehrt darin liegen, dass der Anwendung im Vergleich zur Analogie Funktionalität fehlt. Das hat zur Folge, dass der Benutzer vergeblich versuchen wird, die gewünschte Funktion auszuführen. Die dritte Klasse von Problemen tritt auf, wenn eine bestimmte Funktion zwar im Anwendungssystem und in der Analogie existiert, jedoch unterschiedlich reagiert. Das Ergebnis ist damit für den Benutzer nicht voraussagbar. Die dadurch bei der Anwendung auftretenden Fehler müssen anschließend durch den Benutzer im Rahmen des Fehlermanagement beseitigt werden. Weiterhin ist bei einem internationalem Einsatz der Software zu beachten, dass die verwendete Metapher in verschiedenen Kulturbereichen nicht immer die gleiche Aussage besitzen muss.

Auf Grund der zuvor aufgezeigten Probleme müssen die im Softwaresystem eingesetzten Metaphern bereits frühzeitig evaluiert werden. Damit kann vermieden werden, dass einerseits Funktionalität unerkannt und ungenutzt bleibt und das andererseits die Anwendung nicht überschätzt wird.

3 Design ergonomischer Oberflächen

Unter dem **Design** ergonomischer Oberflächen wird sowohl die Analyse als auch der Entwurf einer Benutzungsschnittstelle verstanden. Design und Evaluation einer Oberfläche stellen mit ca. 50 Prozent einen gewichtigen Anteil am gesamten Softwareentwicklungsprozeß dar.
Der Prozess der Oberflächenentwicklung gestaltet sich grundsätzlich in iterativer Art und Weise (vgl. [Prei99] S. 5f). Dies unterscheidet ihn von dem typischerweise phasenorientierten Software Engineering. Es ist nahezu unmöglich, beim ersten Versuch ein annehmbares Ergebnis zu erzielen. Das größte Problem besteht darin, dass sich die Anforderungen während der Entwicklung verändern. So können Benutzbarkeitsprobleme oft erst erkannt werden, wenn für die zu lösende Aufgabe auch ein Prototyp zur Verfügung steht. Eine weitere Ursache liegt in der Vielfalt der Aspekte, die eine Benutzungsschnittstelle beeinflussen. Diese besteht nicht nur aus visuellen Elementen, die auf dem Bildschirm angezeigt werden. Leistungsfähigkeit, Sicherheit, Unterstützung durch Hilfesysteme, Individualisierung, Organisation der Informationen oder Mehrsprachigkeit sind weitere Gesichtspunkte, die beachtet werden müssen. Die Besonderheiten des Entwicklungsprozess für Benutzungsschnittstellen haben zu dem von NIELSEN geprägten Begriff **Usability Engineering** geführt. Ein so ausgerichteter Softwareentwicklungsprozeß stellt sich das Ziel, durch ergonomisches Design Effektivität bei höchstmöglicher Effizienz und Benutzerzufriedenheit zu gewährleisten. Usability Engineering ist durch einen engen Kontakt von Benutzern und Entwicklern charakterisiert. Typisch ist auch die Verwendung von Werkzeugen, um das Prototyping zu beschleunigen.

3.1 Usability Design

Die wörtliche Übersetzung von Usability ist Benutzbarkeit. Die DIN neigt jedoch dazu, dieses Wort mit Gebrauchstauglichkeit zu übersetzen (vgl. [Dzid94] S. 387). Im folgenden Beitrag werden beide Varianten synonym gebraucht. Die ISO definiert **Gebrauchstauglichkeit** als „das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“ ([ISO9241-11] S. 4) Der **Nutzungskontext** definiert die Benutzer, Arbeitsaufgaben, Arbeitsmittel (Hardware, Software und Materialien) sowie die physische und soziale Umgebung, in der das Produkt genutzt wird. NIELSEN nennt fünf Eigenschaften von Softwaresystemen, die entscheidend die Benutzbarkeit beeinflussen. Dazu

gehören Erlernbarkeit, Effizienz, Wiedererkennbarkeit, Fehlerbehandlung und Benutzerzufriedenheit (vgl. [Niel93] S. 26).

Beurteilt wird das Anwendungssystem von dem Benutzer mit seinen individuellen Bedürfnissen. Er spielt demzufolge die zentrale Rolle bei der Betrachtung der Gebrauchstauglichkeit. Aus diesem Grund schlägt die ISO 13407 einen benutzerorientierten Gestaltungsprozess vor. Dieser ist eine Art der Entwicklung interaktiver Systeme, die sich darauf konzentriert, Systeme gebrauchstauglich zu machen (vgl. [ISO13407] S. 2). Obwohl in der Norm explizit darauf hingewiesen wird, kann der Begriff „benutzerorientiert“ fälschlicherweise so verstanden werden, lediglich die Belange des Benutzers zu berücksichtigen (vgl. Burm+00] S. 54). Aus diesem Grund wird synonym auch von der benutzerzentrierten Softwaregestaltung gesprochen, die eine aktive Einbeziehung des Benutzer klarer ausdrückt.

Der Gestaltungsprozess beginnt mit der Feststellung für die Notwendigkeit des benutzerorientierten Ansatzes. Steht Ausmaß und Umfang fest, kann ein interdisziplinäres Gestaltungsteam zusammengestellt werden, welches aus DV-Verantwortlichen, Entwicklern und Endbenutzern besteht. Dieses Team bleibt für die gesamte Projektdauer zusammen. Wurde ein der Größe des Projektes angepasstes Team gebildet, können die vier benutzerorientierten Gestaltungsaktivitäten durchlaufen werden (vgl. Anhang A.8). Im ersten Schritt wird der Nutzungskontext identifiziert (vgl. [ISO13407] S. 6ff). Dabei werden die Benutzermerkmale, Arbeitsaufgaben und die organisatorische und physische Umgebung analysiert. Abschnitt 3.2 beschäftigt sich mit diesen Aktivitäten eingehend. Im anschließenden zweiten Schritt werden daraus die Benutzeranforderungen und die organisatorischen Anforderungen abgeleitet. Das IEEE fordert, dass die Anforderungsspezifikation abstrakt, eindeutig, nachvollziehbar und überprüfbar ist. (vgl. Burm+00] S. 55). Das Entwerfen der Gestaltungslösungen ist Inhalt des dritten Schrittes. Er stellt die komplexeste Phase dar. Dabei können die Anforderungen mit Hilfe von Simulationen, Modellen oder Prototypen konkretisiert werden. Die Abschnitte 3.3 und 3.4 unterstützen bei der Grobgestaltung, zu der Navigationsstruktur und Informationspräsentation gehören. Für die Entwicklung von geeigneten Metaphern kann zusätzlich der Abschnitt 2.3 zu Rate gezogen werden. In der Feingestaltung werden grafische Gestaltung und Interaktionsverhalten bis ins Detail ausgestaltet. Dabei werden die in Kapitel 4 genauer betrachteten Style Guides verwendet. Der vierte Schritt des Gestaltungsprozesses besteht in der Beurteilung der Gestaltungslösungen gegenüber den Anforderungen. Die Beurteilung bzw. Evaluierung wird ausführlich in

Kapitel 5 beschrieben. Sind die Anforderungen nicht erfüllt, wird erneut der erste Schritt des benutzerorientierten Ansatzes ausgeführt. Praktische Erfahrungen haben gezeigt, dass vor allem bei den ersten Iterationen erhebliche Verbesserungen der Gebrauchstauglichkeit zu erwarten sind. Der Gestaltungsprozess sollte mindestens eine Iteration aufweisen, wobei eine Zweite wünschenswert ist. Erfüllt die Gestaltungslösung die Anforderungen, so ist die Entwicklung abgeschlossen. Die ISO 13407 bietet im Anhang Formulare an, die bei der praktischen Umsetzung der vier Aktivitäten genutzt werden können. Die Norm berücksichtigt auch, dass bestimmte Benutzbarkeitsprobleme erst während der Anwendung über einen längeren Zeitraum erkannt werden können. Sie fordert daher Pläne und Verfahren für die Langzeitbeobachtung (vgl. [ISO13407] S. 9). So können durch eine Hotline oder eine Seite im Internet dem Benutzer Möglichkeiten zur Rückmeldung gegeben werden.
Wie bereits beschrieben, besteht der erste Schritt im benutzerorientierten Gestaltungsprozess aus der Identifikation von Aufgaben, Benutzern und Umgebung.

3.2 Aufgaben und Benutzeranalyse

Die Analyse von Benutzern und Aufgaben, ist eine zentrale Voraussetzung um eine ergonomische Oberfläche zu entwickeln. Ebenso entscheidend sind organisatorische und ökonomische Rahmenbedingungen. Dieser zweite Bereich wird an dieser Stelle nicht genauer betrachtet.
Eine 1997 durchgeführte Studie ergab, dass etwa 60 Prozent aller Benutzbarkeitsprobleme darauf zurückgehen, dass der Dialogablauf nicht dem Arbeitsablauf entspricht. Das heißt, dass die Benutzungsschnittstelle nicht optimal an die zu lösende Aufgabe angepasst ist (vgl. [GeHa98] S. 169). Inkonsistenz ist für 25 Prozent und Unübersichtlichkeit der Dialogelemente für 15 Prozent der Probleme verantwortlich. Der Entwicklungsprozess einer ergonomischen Oberfläche muss demzufolge mit einer Aufgabenanalyse beginnen.

3.2.1 Aufgabenanalyse

Dieser Arbeitsschritt wird im allgemeinen Softwareentwicklungsprozeß als Anforderungsanalyse bezeichnet. Im Bereich der Oberflächenentwicklung spricht man jedoch von der **Aufgabenanalyse** (vgl. [Olse98] S. 18). Das Ziel besteht darin, zu erkennen, was der Benutzer mit dem Anwendungssystem erreichen möchte, welche Strategien und Techniken er dabei benutzt und welche Informationen er für ihre Realisierung benötigt.
Da Benutzer und Softwareentwickler zwei verschiedene „Sprachen“ sprechen, ist es wichtig, am Anfang des Projektes eine gemeinsame Kommunikationsbasis zu

bilden (vgl. [Prei99] S. 210ff). Der Entwickler muss sich mit dem Handlungsumfeld des Benutzers beschäftigen, um einen Einblick in das Fachgebiet zu bekommen. Anschließend sollten Definitionen der verschiedenen Fachbegriffe aufgestellt werden. Sprechen beide Seiten die gleiche „Sprache“, kann mit der systematischen Informationssammlung über die zu unterstützenden Aufgaben begonnen werden.

Für die Informationserfassung gibt es verschiedene Möglichkeiten. Die einfachste ist die Beobachtung. Für sie wird keine Vorbereitungsphase benötigt. Ebenfalls positiv ist der persönliche Kontakt zwischen beiden Parteien. Fragebögen und persönliche Interviews bieten die Möglichkeit, eine Vielzahl von Benutzern in die Analyse mit einzubinden. Allerdings muss bei diesen beiden Techniken mit einer größeren Vorbereitungszeit gerechnet werden. Fragebögen eignen sich insbesondere zur statistischen Auswertung. Ein Nachteil sind die unter Umständen langen Rücklaufzeiten. Im Ergebnis der Informationssammlung sollte der Entwickler wissen, welche Aufgaben existieren, wie sie bisher gelöst wurden, welche Probleme dabei auftreten und wie sich der Benutzer zukünftige Lösungen vorstellt. Anschließend sollte der Prozess der Aufgabenzerlegung beginnen. SHNEIDERMAN weißt auf dabei auftretende Schwierigkeiten hin. Werden zu kleine Teilaufgaben gebildet, können Benutzer durch die Vielzahl durchzuführender Aktionen bei der Aufgabenerledigung frustriert werden (vgl. [Shne98] S. 70). Sind die Teilaufgaben zu komplex, werden viele spezielle Optionen benötigt, die den Benutzer verwirren können. Im Anschluss an die Informationssammlung sollten Aspekte der Reihenfolge, Dauer und Häufigkeit einzelner Teilaufgaben bekannt sein. Ebenso wichtig ist eine Unterscheidung in kritische und unkritische Aufgaben. Kritische Aufgaben sind solche, welche bei fehlerhafter Ausführung entscheidend den Arbeitsablauf negativ beeinflussen. Bekannt ist auch, welche Informationen für die Aufgabenerledigung obligatorisch oder optional sind.

Im nächsten Schritt der Aufgabenanalyse werden Sequenzen von Handlungsschritten entworfen. Dabei ist zu definieren, welche Informationen vom Benutzer benötigt werden, welche das System liefern kann und welche Informationen an den Benutzer zurückgegeben werden müssen. Orientierungshilfen sind die oben genannten Aspekte. Häufig auftretende Handlungen sollten besonders schnell erreichbar sein. Bei ihnen ist auch auf eine angemessene Leistungsfähigkeit des Softwaresystems zu achten. Bei jeder Sequenz ist zu analysieren, ob Komplikationen auftreten können und welche Maßnahmen in diesem Fall einzuleiten sind. Mit diesem Wissen können bereits Aussagen über die zu verwendenden Interaktionsformen (vgl. Abschnitt 3.4.2) getroffen werden.

Den Abschluss der Aufgabenanalyse bildet die Auswertung. Bis zu diesem Punkt wurde die Aufgabenanalyse darauf beschränkt, den bisherigen Ablauf zu erfassen. Die Einführung eines neuen Softwaresystems sollte jedoch als Anlass genommen werden, Verbesserungsmöglichkeiten zu erkennen und einzuführen. In den seltensten Fällen ist die Art und Weise, wie die Aufgaben bisher erledigt wurden, optimal. Vor allem die Benutzer, als Experten auf ihrem Gebiet, können dabei wichtige Anstöße geben. Die erweiterten Möglichkeiten, die ein Softwaresystem bieten kann, beispielsweise der Einsatz von leistungsstarken Datenbanken, müssen dem Benutzer jedoch dargeboten werden. Werden diese neuen Techniken eingesetzt, ist eine qualitativ bessere Lösung der Aufgabenerledigung möglich.

3.2.2 Benutzeranalyse

Kontakt zu den Benutzern eines Systems aufzubauen, ist für die Entwicklung von ergonomischen Benutzungsschnittstellen von großer Bedeutung. Nur dadurch kann eine angemessene Benutzeranalyse durchgeführt werden. Dabei muss zwischen Auftraggebern, d.h. dem Kunden und den Benutzern unterschieden werden. Zur Verdeutlichung wird deshalb auch von den **Endbenutzern** gesprochen (vgl. [Prei99] S. 214f). Um die Aufgaben genau zu analysieren müssen die Benutzer des Systems eingebunden werden. Sie kennen nicht nur die Abläufe und Ergebnisse, sondern vor allem die praktischen Probleme bei der Bewältigung der Aufgaben. Demgegenüber stehen die Auftraggeber. Sie können die Aufgaben des Anwendungssystems definieren, wissen jedoch weniger über die praktische Umsetzung. Doch vor allem die Betrachtung von Details führt zu einer ergonomischen Oberfläche. Außerdem hat die Einbeziehung der Benutzer einen wichtigen Nebeneffekt. Wie in der Organisationslehre herausgearbeitet wurde, können Individuen Veränderungen gegenüber ablehnend reagieren. Werden die Benutzer aktiv in den Entwicklungsprozess einbezogen und können das System mit gestalten, sinkt diese ablehnende Haltung. Ihre u.U. vorhandenen Ängste werden dadurch abgebaut. Wird das nicht beachtet, kann das im schlimmsten Fall zum Boykott und damit zum Scheitern des Projektes führen.

Benutzer unterscheiden sich in einer Vielzahl von Eigenschaften. Dazu zählen Alter, Geschlecht, physische Fähigkeiten, Ausbildung, kultureller und ethnischer Hintergrund, Motivation, Ziele und Persönlichkeit, um nur einige zu nennen (vgl. [Shne98] S. 67f). Diese verschiedenen Voraussetzungen müssen im Anwendungssystem berücksichtigt werden, um für alle Benutzer eine ergonomische Oberfläche anbieten zu können. Eine sorgfältige Analyse führt zur Bildung von Benutzergruppen. Übliche Einteilungen gliedern sich beispielsweise in

Anfänger, erfahrene Benutzer und Experten. Eine Klassifikationen nach der Arbeitsaufgabe kann beispielsweise die drei Gruppen Lehrling, Sachbearbeiter und Abteilungsleiter ergeben. Ziel ist es, möglichst viele heterogene Individuen in homogene Gruppen einzuordnen. Die Auswahl der Kriterien hängt von dem konkreten Anwendungsfall ab. Sinnvolle Unterscheidungsmerkmale sind zum Beispiel Ziele oder die Fähigkeiten und Erfahrungen des Benutzers mit dem Anwendungsgebiet. Die Ermittlung der verschiedenen Gruppen kann ähnlich wie bei der Aufgabenanalyse mit Hilfe von Interviews und Fragebögen oder durch Beobachtung erfolgen. HIX und HARTSON weisen darauf hin, dass vor allem die Beobachtung der Benutzer geeignet ist (vgl. [HiHa93] S. 128). Jede der identifizierten Benutzergruppen hat verschiedene Anforderungen an das Anwendungssystem. Da aus ökonomischen Überlegungen heraus nicht alle realisierbar sind, müssen im Vorfeld der Entwicklung Prioritäten gesetzt werden. Im allgemeinen wird man die meisten Zugeständnisse der Benutzergruppe mit den meisten Mitgliedern machen. Ein anderes Kriterium ist die Dauer bzw. Häufigkeit der Nutzung des Anwendungssystems.

Die Bildung von Benutzergruppen kann als Ausgangspunkt der Erstellung von Benutzermodellen angesehen werden. Ein **Benutzermodell** enthält explizite Annahmen über einen konkreten Benutzer, die es ermöglichen, dass das System auf diesen Benutzer zugeschnitten wird (vgl. [Prei99] S. 466). Die Annahmen bestehen im einfachsten Fall aus Attributen mit entsprechenden Werten. Die Erstellung und Verwaltung von Benutzermodellen durch das Anwendungssystem ist die Grundvoraussetzung für adaptive Systeme.

Die folgenden Ausführungen sollen den Sachverhalt verdeutlichen. Betrachtet man die zu lösende Aufgabe als Anwendungsbereich A, so kann S(A) als Anwendungssystem bzw. die Implementierung des Anwendungsbereiches bezeichnet werden (vgl. [Herc94] S. 18f). Das mentale Modell des Benutzers vom Anwendungsbereich wird durch B(A) beschrieben. Das Benutzermodell S(B(A)) ist somit das Modell des Systems vom mentalen Modell des Benutzers. Eine Inkompatibilität zwischen S(A) und B(A) führt bei der späteren Nutzung des Anwendungssystems zu Problemen. Diese Inkompatibilität resultiert im allgemeinen aus einem unzulänglichen Modell des Entwicklers vom mentalen Modell des Benutzers E(B(A)), kurz einer unzureichenden Benutzeranalyse.

3.3 Gestaltungsgrundsätze für Dialoge

Unter dem Dialog zwischen Mensch und Maschine wird der wechselseitige Informationsaustausch zur Erfüllung einer Aufgabe verstanden (vgl. [Geis90] S. 141). Den

hier betrachteten Dialog eines Anwendungssystems kann man als zielorientierte Interaktion zwischen Benutzer und Dialogsystem definieren. D.h., dass eine Benutzungsschnittstelle aus mindestens einem, meist aber aus mehreren Dialogen besteht. Die Initiative für den Informationsaustausch kann sowohl vom Menschen als auch von der Maschine ausgehen und kann während des Dialogverlaufes wechseln. Ziel des Dialoges ist eine Benutzerführung. Der Anwender soll bei der Erledigung seiner Aufgabe unterstützt werden. Ein guter Dialog vermeidet dabei jegliche Über- oder Unterforderung des Menschen. Die vorher durchgeführte Benutzeranalyse hilft bei der Umsetzung dieser Forderung. Ein Dialog sollte selbsterklärend sein (vgl. Abschnitt 3.3.2). D.h., dass der Anwender den Dialog selbstständig, ohne fremde Hilfe benutzen kann. Das bedeutet jedoch nicht, dass auf die Einführung eines Anwendungssystems, die mit einer Schulung verbunden sein kann, verzichtet werden darf. Weiterhin ist durch die Dialoggestaltung sicherzustellen, dass das System vom Benutzer akzeptiert wird. Die Anpassung des Dialoges an die Fähigkeiten des Benutzer sollte ebenso möglich sein (vgl. Abschnitt 3.3.6). Diese kann entweder durch den Benutzer erfolgen, oder durch das System automatisch vorgenommen werden.

Um einen Dialog nach seiner Brauchbarkeit zu bewerten, werden angemessene Kriterien benötigt. Benutzungsschnittstellen sind jedoch „sehr inhomogene und vieldimensionale Gebilde, die sich weitgehend einer direkten, objektiven Messung entziehen.“ ([Herc94] S. 104) Dennoch gibt es eine Reihe von Merkmalen, die eine Bewertung ermöglichen. Die ISO 9241 definiert sieben Grundsätze der Dialoggestaltung (vgl. [ISO9241-10]). Diese sind nicht unabhängig voneinander. So kann es nötig sein, Vorteile des einen Grundsatzes gegenüber denen eines anderen abzuwägen. Im folgenden werden die Grundsätze einzeln beschrieben.

3.3.1 Aufgabenangemessenheit

Die ISO 9241-10 definiert einen Dialog als „aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.“ ([ISO9241-10] S. 4)

Ein Dialog sollte nur die Informationen anzeigen, die für die zu erledigende Aufgabe relevant sind. Damit soll eine Reizüberflutung des Benutzers vermieden. Entwickler bieten oft überflüssige Informationen an, weil beispielsweise die bei der Programmierung verwendete Komponente diese Funktionalität bietet. Die Frage, ob damit die Erledigung der Arbeitsaufgabe erleichtert wird, wird oft nicht gestellt. Die Darstellung der Informationen muss dabei an die Arbeitsaufgabe angepasst werden. So sollten beispielsweise lange Listen nach Vorgabe des Benutzer sortiert werden können. Die Anzeige der Genauigkeit von Zahlen

wird nicht von den technischen Möglichkeiten sondern von der Aufgabe vorgegeben. Aufgabenangemessenheit bedeutet auch, dass das Dialogsystem automatisierbare Arbeitsschritte auch selbständig ausführt. So können Positionsmarken im vorhinein sinnvoll platziert werden oder Startprozeduren automatisch vom System ausgeführt werden. Bei der Dialoggestaltung ist sowohl der Grad an Komplexität der Aufgabe als auch die Qualifikation des Benutzers zu beachten. So müssen beispielsweise bei komplizierten Aufgaben die Hilfetexte entsprechend umfassend die einzelnen Teilprobleme erläutern. Die auf Verlangen des Benutzer angezeigte Hilfe sollte sich mit der gegenwärtig zu bearbeitenden Aufgabe beschäftigen. Innerhalb des Hilfesystems kann dann der Benutzer zu anderen Themenbereichen navigieren (vgl. Abschnitt 3.4.6). Eine weitere Forderung besteht in der Vorgabe von Standardwerten. Diese müssen sinnvoll ausgewählt und durch den Benutzer veränderbar sein. Vor allem bei Datumseingaben tritt häufig der Fall auf, dass das aktuelle Datum eingegeben werden muss. Das Eingabefeld kann in diesem Fall vom Anwendungssystem gefüllt werden. Enorme Zeiteinsparungen lassen sich auch erreichen, wenn häufig wiederkehrende Aufgaben speziell behandelt werden können. Es sollte möglich sein, diese zu automatisieren und damit zu beschleunigen. Im allgemeinen wird das durch den Einsatz von Makros realisiert (vgl. Abschnitt 3.4.8). Werden während der Erledigung einer Aufgabe Daten geändert, sollte dem Benutzer möglich sein, die ursprünglichen Daten wieder herzustellen. So kann beispielsweise die Eingabe in einem Textfeld durch Drücken der Taste ESC wieder zurück gesetzt werden. Deutlich mächtiger als dieser einfache Mechanismus ist eine Undo-Funktion, die ganze Dialogschritte rückgängig machen kann (vgl. Abschnitt 3.3.3). Die letzte Forderung besteht in der Vermeidung unnötiger Arbeitsschritte. So kann beispielsweise das Drucken und Schließen einer Rechnung in einem Schritt durchgeführt werden.

Die Effizienz des Dialoges hängt entscheidend von der Anpassung des Systems an die Aufgabe ab. Die in Abschnitt 3.2.1 beschriebene Aufgabenanalyse ist dafür die Voraussetzung. Die benutzergerechte Gestaltung kann mit der Anpassung an die Aufgabe konkurrieren. In diesen Fällen muss ein Kompromiss gefunden werden.

3.3.2 Selbstbeschreibungsfähigkeit

„Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.“ ([ISO9241-10] S. 5)

Diese Erläuterungen stellen eine Ergänzung der vorher durchgeführten Benutzerschulung dar, keinen Ersatz. Hilfetexte sollen es dem Benutzer erleichtern, sich schnell und problemlos ein mentales Modell des Anwendungssystems aufzubauen. Die Erläuterungen müssen den Kenntnissen des Benutzer angepasst sein. So kann vor allem bei Anfängern die Darstellung von Beispielen hilfreich sein. Die Texte sollten sowohl die Muttersprache des Anwenders als auch die Fachbegriffe des Arbeitsgebietes verwenden. Ein Dialogsystem sollte nach jeder Handlung des Benutzers eine Rückmeldung geben. Wird vom Benutzer ein Arbeitsschritt initiiert, der schwerwiegende Folgen haben kann, sollte zuvor eine Bestätigung verlangt werden. Dabei sollte der Benutzer die Möglichkeit haben, sich dazugehörige Erläuterungen anzeigen zu lassen. Das Dialogsystem sollte sicherstellen, dass der Benutzer immer über den gegenwärtigen Dialogzustand informiert ist. Dazu gehört beispielsweise ein Überblick über vergangene und zukünftige Dialogschritte. Wird eine Eingabe verlangt, sollten dem Benutzer Informationen darüber angezeigt werden. Ein Beispiel ist die Anzeige des erwarteten Formates. Das Hilfesystem (vgl. Abschnitt 3.4.6) sollte die Erläuterungen kontextabhängig anbieten. Das bedeutet, dass in Abhängigkeit der aktuellen Dialogsituation ein an den Anwendungszustand orientierter Text angeboten wird. Merkmale der Dialogsituation stellen beispielsweise das ausgewählte Fenster, der Eingabefokus, oder der aktuelle Bearbeitungsschritt der Aufgabe dar. Hilfetexte können aktive oder passiv sein. Ersteres bedeutet, dass der Benutzer die Initiative übernimmt und durch eine entsprechende Eingabe die Anzeige von Erläuterungen fordert. Aktive Hilfe sind automatisch vom System angebotene Erläuterungen. Ein Beispiel wäre die Anzeige eines Hilfetextes nach einem Eingabefehler.

3.3.3 Steuerbarkeit

„Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“ ([ISO9241-10] S. 6) Das heißt, dass die Kontrolle, soweit es die Aufgabe zulässt, beim Benutzer liegen soll. Parallel durchführbare Aufgaben müssen auch gleichzeitig abzuarbeiten sein. Das System sollte in mehrere unterbrechbare und wieder fortsetzbare Teildialoge gegliedert werden. Auf modale Dialoge, die einen Wechsel in einen anderen Dialog ohne vorherige Beendigung verhindern, sollte beim Systementwurf verzichtet werden. Ausgeschlossen sind kritische Vorgänge, wie zum Beispiel Fehlermeldungen. Bei Systemen mit hohem Sicherheitsbedürfnis ist ebenfalls von zu großer Freiheit für den Benutzer abzusehen und ein systemgesteuerter Dialog vorzuziehen. Die Geschwindigkeit des Dialoges

sollte dem Benutzer nicht vorgeschrieben werden. So wird beispielsweise das Ende der Eingabe in einem Feld mit der Taste ENTER durch den Benutzer signalisiert. Der Benutzer sollte selbst die Interaktionsform auswählen können (vgl. Abschnitt 3.4.2). So bevorzugen Anfänger beispielsweise die Funktionsauswahl über Menüs, während Experten lieber Kommandos verwenden. Die ISO 9241 subsumiert unter dem Begriff Steuerbarkeit auch die Forderung nach Stornierbarkeit von Aktionen. Es sollte wenigstens der letzte Dialogschritt, soweit seine Folgen reversibel sind, zurückgenommen werden können. Diese Technik wird als **Undo** bezeichnet. Durch die Möglichkeit die letzte Aktion rückgängig zu machen, wird der Anwenders angeregt, das Anwendungssystem zu explorieren. Die Angst, einen Fehler zu begehen wird entscheidend gesenkt. Soweit möglich sollte das System nicht nur für den letzten, sondern für alle gemachten Schritte ein Undo ermöglichen. Möchte der Benutzer etwas ausprobieren, ist meist eine gewisse Anzahl von Arbeitsschritten notwendig, um das Ergebnis zu erhalten. Wird diese Technik unterstützt, sollte die direkte Wahl des entsprechenden Undo-Schrittes möglich sein, um mehrere Arbeitsschritte auf einmal rückgängig zu machen. Da sich der Benutzer nur bis zu einen gewissen Grad die letzten Aktionen merken kann, muss in diesen Fällen die Anzeige der Schritte auch eine kurze Erläuterung beinhalten. Die Rücknahme einer Undo-Operation wird mit **Redo** bezeichnet.

3.3.4 Erwartungskonformität

„Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z. B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen.“ ([ISO9241-10] S. 6)

Während sich die Erfahrungen des Benutzers bei der Arbeit mit dem System immer mehr ausweiten, schränkt sich die Menge der Erwartungen immer mehr ein. Stimmen die Erwartungen des Benutzers mit dem tatsächlichen Systemverhalten überein, so ist das Anwendungssystem konsistent. Dazu sollte Dialogverhalten und Informationsdarstellung innerhalb des Dialogsystems einheitlich sein. Hierbei helfen sowohl der Einsatz von Standardkomponenten als auch Style Guides. Eine Änderung des Dialogzustandes sollte auf einheitliche Weise herbeigeführt werden. Dazu zählen der einheitliche Aufruf des Hilfesystems oder der Abbruch eines Dialogs mit der Taste ESC. Der Einsatz eines einheitlichen, dem Benutzer vertrauten Wortschatzes steigert nicht nur die Selbstbeschreibungsfähigkeit sondern auch die Erwartungskonformität. Die Gestaltung von ähnlichen Aufgaben sollte immer gleich sein. Damit kann der Benutzer ein bekanntes Verfahren leicht abgewandelt wieder einsetzen. Zusätzlich ist ein hoher Grad an Konsistenz

von Bedeutung, da er das Lernen vereinfacht und unnötige Belastungen für den Benutzer vermeidet. Es lassen sich drei Arten von Konsistenz unterscheiden (vgl. [Herc94] S. 112). Die Einheitlichkeit der Dialoge innerhalb einer Anwendung wird als innere Konsistenz bezeichnet. Sie muss durch den Dialogentwickler sichergestellt werden. Dabei haben Verwendung, Positionierung, Farbe und Verhalten von Bildschirmelementen einen großen Einfluss. Die äußere Konsistenz beschreibt die Einheitlichkeit von Dialogen verschiedener Anwendungssysteme. In diesem Bereich wirken die Richtlinien und Techniken des zugrunde liegenden Betriebssystems unterstützend. Die metaphorische Konsistenz beschreibt die Einheitlichkeit der Dialoge mit der realen Arbeitswelt (vgl. Abschnitt 2.3). Insbesondere die Aufgabenanalyse hilft dabei, eine entsprechende Metapher zu entwickeln.

3.3.5 Fehlertoleranz

„Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“ ([ISO9241-10] S. 7)

Fehler und Probleme im Umgang mit Software treten alltäglich auf. Etwa 10 Prozent der Computerarbeitszeit verbringen Benutzer damit, diese zu bewältigen (vgl. [Kens+95] S. 217). Damit stellen sie nicht nur eine zusätzliche Belastung für den Benutzer dar, sondern verursachen ferner für das Unternehmen hohe Kosten. Fehler sind immer vom aktuellen Arbeitskontext abhängig und bedürfen daher spezifischer Behandlung. Syntaktische Fehler sind vom System erkennbar. Eine Korrektur kann automatisch oder manuell erfolgen. Wird der Fehler automatisch korrigiert, beispielsweise Tippfehler, muss der Benutzer darauf hingewiesen werden, ohne jedoch den Dialogablauf zu stören. In dem genannten Fall kann die Eingabe korrigiert angezeigt werden, was als Hinweis ausreichend ist. Die automatische Fehlerkorrektur muss abschaltbar sein. Bei manueller Fehlerbeseitigung muss eine Fehlermeldung erfolgen, die in ihrer Ausführlichkeit anpaßbar ist. Das erlaubt unerfahrenen Benutzern sich über den Grund des Fehlers ausführlich zu informieren, während Experten mit diesen Informationen nicht belästigt werden. Zusätzlich sollte der Benutzer bei der Behebung des Fehlers unterstützt werden. Vorhandene Alternativen können dargestellt und erläutert werden. Die Fehlermeldungen selbst sind verständlich, sachlich und konstruktiv zu formulieren. Sie sollten einheitlich strukturiert werden. Dazu gehört sowohl der Aufbau des Fehlertextes als auch seine Darstellung.

Das Ziel beim Entwurf des Anwendungssystems besteht darin, das Auftreten von Fehlern zu vermeiden. So können Eingaben beschränkt werden, indem z.B. keine Ziffern in Textfeldern mit Zahlen als Inhalt erlaubt sind.
Semantische Fehler treten immer dann auf, wenn das mentale Modell des Benutzers von dem tatsächliches Modell des Anwendungssystems abweicht. Sie können vor allem durch erwartungskonforme Systeme und den Einsatz geeigneter Metaphern eingegrenzt werden.

3.3.6 Individualisierbarkeit

„Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.“ ([ISO9241-10] S. 8) Umfassende Individualisierungsmöglichkeiten sind keine Rechtfertigung, auf die ergonomische Gestaltung von Dialogen zu verzichten. Die Möglichkeiten der Anpassung müssen durch den Entwickler sinnvoll begrenzt werden. Damit sollen Beeinträchtigungen durch Individualisierung vermieden werden. Beispielsweise darf es nicht möglich sein, die Lautstärke eines Audiosystems soweit zu erhöhen, dass Hörschäden entstehen können.
Grundsätzlich sollten Techniken zur Anpassung an Sprache und kulturelle Eigenheiten bereit gestellt werden. Dieser Aspekt wird unter dem Begriff Internationalisierung diskutiert. Weiterhin sollten sensomotorische und geistige Fähigkeiten des Benutzers beachtet werden. So ist beispielsweise für farbenfehlsichtige Anwender die Anpassung von Farben ein wichtiges Kriterium. Das Wissen der Benutzer und ihre Erfahrungen im Aufgabenbereich und mit dem Anwendungssystem sind unterschiedlich und unterliegen einem stetigen Wandel. Dementsprechend sollten Individualisierungsmöglichkeiten auch diesen Bereich abdecken. Ein Beispiel ist die Anpassung des Umfanges von Hilfetexten. Die Darstellung von Informationen sollte ebenfalls änderbar sein. So bevorzugen manche Benutzer eine Listendarstellung, während andere die Darstellung in Karteiform präferieren. Auch wenn in der Anwendung der Wortschatz des Arbeitsgebietes verwendet wurde, sollte es dem Benutzer erlaubt sein, eigene Bezeichnungen festzulegen. Die letzten Forderungen sind Anpassung von Kommandos und die Wahl der Interaktionsform. Die Kommandoanpassung umfasst sowohl ihre Bezeichnung als auch die Funktionalität. Als Beispiel können die in Abschnitt 3.3.1 erwähnten Makros genannt werden.

3.3.7 Lernförderlichkeit

„Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen des Dialogsystems unterstützt und anleitet.“ ([ISO9241-10] S. 9)

Lernen als das Erarbeiten einer möglichst geeigneten Vorgehensweise oder Strategie zur Erfüllung einer gegebenen Aufgabe durch wiederholte Übung (vgl. [Joha93] S. 147) ist für die effiziente Nutzung eines Softwaresystems von großer Bedeutung. Umfangreiche Systeme können auch durch einführende Schulungen nicht von Anfang an umfassend erfasst und in ihrer ganzen Funktionalität genutzt werden. Vielmehr sollte der Benutzer unterstützt werden, selbstständig sein Wissen über das System während seiner Nutzung zu verbessern und zu erweitern. Dabei helfen die bereits erwähnten Metaphern, die es ermöglichen, bereits vorhandenes Wissen in den neuen Bereich zu transferieren. Eine Hilfefunktion wirkt ebenfalls unterstützend. Die im Abschnitt 3.3.3 erläuterte Undo-Funktion unterstützt das „Learning-by-doing“. Bei diesem Konzept soll der Anwender während der Erledigung einer Aufgabe lernen. Werden Schritte ausgeführt, die fehlerhafte Auswirkungen haben, können sie jederzeit wieder rückgängig gemacht werden.

Die Lernförderlichkeit eines Anwendungssystems ist auch wichtig, weil Benutzer die Eigenschaft besitzen, Funktionen und Abläufe die länger nicht genutzt wurden, wieder zu vergessen. In diesen Fällen reicht ein kurzes Wiederauffrischen des bereits Gelernten. Kurze Hinweise in Form von Tipps (vgl. Abschnitt 4.2.23) können dabei bereits ausreichen. Ein wichtiger Nebeneffekt besteht darin, dass Lernen bei der Benutzerzufriedenheit ein positiven Einfluss besitzt.

3.4 Entwurfstechniken

Die im vorherigen Abschnitt herausgearbeiteten Gestaltungsgrundsätze müssen bei der Entwicklung einer Benutzungsschnittstelle umgesetzt werden. Erst damit kann eine ergonomische Oberfläche entstehen. Es gibt eine Vielzahl von Prinzipien und Techniken, die dabei helfen können. Diese sind allgemeiner formuliert als konkrete Style Guides. Die Palette der Möglichkeiten ist jedoch zu weit gefächert, als das eine umfassend vollständige Beschreibung aller Aspekte möglich wäre. Aus diesem Grund werden die wichtigsten Prinzipen ausgewählt und diskutiert. Die einzelnen Techniken sind meist von mehreren Gestaltungsgrundsätzen beeinflusst. So unterstützt beispielsweise die Anwendung von Icons die Selbstbeschreibungsfähigkeit und Lernförderlichkeit. Die Auswahl der Interaktionsform wirkt sich dagegen hauptsächlich auf die Steuerbarkeit von Anwendungssystemen aus.

Die hier aufgeführte Reihenfolge der einzelnen Techniken steht nicht im Verhältnis zur ihrer Wichtigkeit. Vielmehr sollten alle Prinzipien beachtet und benutzt werden. Den Grad der Intensität geben die Rahmenbedingungen des Entwicklungsprojektes vor. Zu diesen gehören sowohl die Aufgabe und die Benutzer als auch finanzielle Gesichtspunkte. So wirken Sicherheitsabfragen innerhalb eines Systems erschwerend bei der Benutzung, können aber äußerst nützlich und somit sinnvoll sein. In diesen Fällen müssen Kompromisse eingegangen werden.

3.4.1 Fenstersysteme

Grafische Fenstersysteme werden heute in den meisten Computersystemen als Dialogform genutzt. Durch sie ist eine bessere Abbildung der menschlichen Gedankenwelt möglich, als das auf textueller Kommunikation beruhende Dialogsysteme vermögen (vgl. [Prei99] S. 77). Ein **Fenster** beschreibt dabei einen „separat steuerbaren Bereich auf dem Bildschirm, der zur Darstellung von Objekten und/oder zur Durchführung eines Dialogs mit dem Benutzer verwendet wird.“ ([ISO9241-16] S. 5) Es bietet dem Benutzer eine Sicht auf das Anwendungssystem. Durch die Verwendung mehrerer Fenster und damit unterschiedlicher Sichten kann dem Anwender ein umfassendes Verständnis der Anwendung und ihres aktuellen Zustandes vermittelt werden.

In den meisten grafischen Systemen können eine Vielzahl von Anwendungen simultan ablaufen (vgl. [Olse98] S. 91). Dabei besteht jede aus einem Haupt- und mehreren Unterfenstern. Um eine unnötige Konfusion zu vermeiden, werden grafische Benutzungsschnittstellen auf der Grundlage von **Basis-Window-Systemen** entwickelt. Diese werden durch zwei Aspekte charakterisiert. Zum einem bieten sie Softwaredienste an, die es Anwendungen ermöglichen, sowohl eigene Fenster zu erstellen und zu organisieren als auch diese mit Interaktionselementen zu füllen. Andererseits gestattet das Basis-Window-System dem Benutzer Größe und Position der Fenster zu kontrollieren. Die dafür zuständige Softwarekomponente wird als **Fenstermanager** bezeichnet.

Damit eignen sich Fenstersysteme ideal für die parallele Bearbeitung mehrerer Aufgaben. Diese kann sowohl anwendungsintern als auch anwendungsübergreifend stattfinden. Durch überlappende Darstellung der Fenster können dem Benutzer die vorhandenen Prozesse angezeigt werden. Diese Darstellung unterstützt die Schreibtischmetapher. Genau wie beim Stapeln von Dokumenten, ist nur das zu oberst liegende komplett sichtbar. Entsprechende Auswahlmechanismen, um das aktuelle Fenster festzulegen, müssen durch das Anwendungssystem bereit gestellt werden. Die einfachste Möglichkeit ist das Anklicken mit

einem Zeigegerät wie der Maus. Da Fenster nicht zwingend angezeigt werden, sondern auch komplett verdeckt sein können, müssen innerhalb der Anwendung Alternativen angeboten werden. Eine Auswahl mittels Menü umgeht das genannte Problem.

Sowohl die Größe als auch die Position von Fenstern kann vom Benutzer verändert werden. Ist durch den speziellen Anwendungsfall diese Freiheit nicht sinnvoll, kann sie beispielsweise durch Angabe der maximalen Größe beschränkt werden. Ein wichtiges Kriterium bei Fenstersystemen ist die Platzierung der Fenster. In den meisten Fenstermanagern, wie z.B. Microsoft Windows, sind zwar Strategien integriert, diese sind jedoch unzureichend. So betreffen die Befehle zur Neuanordnung der Fenster alle im derzeitigen Nutzungskontext erreichbaren Fenster. Damit lassen sich Operationen nicht gezielt auf Fenster anwenden, die für eine bestimmte Aufgabe erforderlich sind. Es gibt zwar den Mechanismus einer Eltern - Kind Beziehung, diese Einteilung wird jedoch durch den Entwickler vorgenommen und ist für den Benutzer unzugänglich. Beim Anzeigen eines neuen Fensters sollte vom System sichergestellt werden, dass es wichtige Bildschirmbereiche nicht verdeckt. So darf der Dialog zum Suchen und Ersetzen bei der Sicherheitsabfrage nicht das Objekt überlappen, welches ersetzt werden soll. Fenster sollten sich nach Möglichkeit ihre letzte Position und Größe „merken“. Üblicherweise entwickelt ein Benutzer während der Arbeit mit dem Anwendungssystem eine Fensteranordnung, die er auch zukünftig präferiert. Im entwickelten Prototyp wurden dafür zwei Methoden eingeführt, die diese Arbeit automatisch übernehmen (vgl. Abschnitt 4.2.1). Fenster bieten die Möglichkeit, Daten von einem Fenster in ein anderes zu transferieren. Dies erhöht die Effizienz bei der Aufgabenerledigung, da eine redundante Erstellung der Daten nicht notwendig ist. Im allgemeinen wird dieser Mechanismus mittels einer Zwischenablage realisiert.

Werden mehrere Aufgaben gleichzeitig abgearbeitet, werden entsprechend viele Fenster benötigt. Dabei kann die Bildschirmorganisation unübersichtlich werden, was zur Verwirrung des Benutzer führt. Dieser beschäftigt sich dadurch sehr lange mit der Verwaltung der Fenster. Die Effizienz der Aufgabenerfüllung sinkt. Virtuelle Bildschirme sind eine Technik, um das zu verhindern. Der Bildschirm wird in eine Raummetapher gewandelt. Dabei kann der Benutzer verschiedene Fenster gruppieren und diese einem Raum zuordnen. Zwischen den einzelnen Räumen kann der Benutzer beliebig hin und her wechseln, wobei der jeweils aktuelle Raum bildschirmfüllend dargestellt wird.

KRUSCHINSKI gibt einige grundsätzliche Richtlinien zum Aufbau von Fenstern (vgl. [Krus99] S. 36ff). Ein Fenster sollte nur die für die Teilaufgabe relevanten

Informationen anzeigen. Diese sind allerdings vollständig darzustellen. D.h., dass alle benötigten Informationen direkt im Fenster zugänglich sind. Die Informationsdichte eines Fensters sollte zwischen 25 Prozent und 40 Prozent liegen. Sie wichtet den durch die Interaktionselemente benötigten Platz mit der gesamten Größe des Fensters. Einer der wichtigsten Aspekte der Fenstergestaltung ist die Einhaltung der Konsistenz. Ein gleichartige Struktur hat verschiedene Vorteile. So sinkt beispielsweise sowohl die Lernzeit als auch Suchzeit des Benutzers, was gleichzeitig zu einer Steigerung der Effizienz führt. Als letzter Punkt ist bei der Gestaltung von Fensters auf ein harmonisches Erscheinungsbild zu achten. Dabei sind die wichtigsten Faktoren Proportion, Balance, Vorhersagbarkeit, Symmetrie, Einfachheit, Klarheit und Sequenz. Diese werden an dieser Stelle jedoch nicht eingehend diskutiert. KRUSCHINSKI gibt in seiner Arbeit einen ausführlichen Überblick darüber (vgl. [Krus99] S. 41ff).

3.4.2 Interaktionsformen

Bei der Nutzung von Anwendungssystemen treten bestimmte Aktionen in gleicher oder ähnlicher Form immer wieder auf. Die Aktionen werden als **Interaktionsaufgaben** bezeichnet (vgl. [Prei99] S. 117). Sie drücken aus, *was* der Benutzer machen kann. Die Interaktionsaufgaben können in verschiedener Art und Weise durch den Anwender durchgeführt werden. So kann er beispielsweise zwischen Maus- oder Tastatureingabe auswählen. Die verschiedenen Möglichkeiten werden als **Interaktionsform** bzw. synonym **Interaktionstechnik** bezeichnet. Sie drücken aus, *wie* der Benutzer etwas machen kann. Grundsätzlich lässt sich die Aussage treffen, dass die Interaktionsform sowohl von der zu erledigenden Aufgabe als auch von dem Benutzer abhängt. Im folgenden werden die wichtigsten Interaktionstechniken näher beschrieben. Obwohl hier eine getrennte Betrachtung stattfindet, werden sie in der Praxis meist gemischt verwendet.

3.4.2.1 Deskriptive Interaktion

Die deskriptive Interaktion basiert auf der Verarbeitung von Sprache. Dabei kann sowohl eine Kommandosprache als auch natürliche Sprache eingesetzt werden. Die Kommandosprache war eine der ersten Interaktionsformen von Benutzungsschnittstellen (vgl. [HiHa93] S. 82). Sie besteht aus einer kontextfreien Grammatik, deren Schlüsselwörter mit möglichst wenig Buchstaben auskommen. Aus diesem Grund muss diese Sprache vom Benutzer erlernt werden. Wird sie beherrscht, stellen die Kommandos eine äußerst effiziente Methode dar, mit dem System zu kommunizieren. Unterstützend kann eine AutoComplete-Funktion vom

System angeboten werden (vgl. [Prei99] S. 119). Wird ein Kommando während der Eingabe erkannt, obwohl es noch nicht komplett eingegeben wurde, so kann es vom System automatisch ergänzt werden. Der Einsatz von Aliasnamen, d.h. die Definition eines normalerweise kürzeren Wortes für ein Kommando ermöglicht weitere Effizienzsteigerungen. Ein wichtiger Aspekt bei Kommandosprachen stellt das Hilfesystem dar. SHNEIDERMAN betont, dass ein solches nur schwer anzubieten ist (vgl. [Shne98] S. 73). Es existieren jedoch Möglichkeiten, auch diese Interaktionsform entsprechend zu unterstützen. So sollte der Benutzer möglichst schnell sowohl verfügbare Kommando als auch seine Parameter einsehen können. Bei fehlerhafter Eingabe kann automatisch eine Übersicht entweder der Parameter oder der lexikalisch benachbarten Kommandos angeboten werden. Fehlende Angaben können durch Standardwerte ergänzt werden.

Kommandosprachen besitzen viele Vorteile. Sie erfordern keine speziellen Eingabegeräte und sind universell hinsichtlich der Ausdrucksmächtigkeit. Bei Beherrschung der Syntax können Aufträge aller Art effizient formuliert werden (vgl. [Ober94] S. 119). Die Aufzeichnung eines Protokolls ist sehr leicht umzusetzen. Die Definition von Makros gestaltet sich für den Benutzer unproblematisch. Neben den Vorteilen existieren jedoch auch erhebliche Nachteile. So müssen die Kommandos gelernt werden um aus dem Gedächtnis abrufbar zu sein. Eine flüssige Interaktion erfordert eine schnelle und fehlerfreie Tastatureingabe. Abkürzungen, Standards und Makros können dabei unterstützend wirken, erhöhen jedoch den Lernaufwand. Der Wirkungsbereich von Kommandos umfasst das gesamte System. Aus diesem Grund sind die Auswirkungen eines Kommandos für den Benutzer schwer zu kontrollieren. Der hohe Lernaufwand gepaart mit der konsequenten Ausrichtung auf Effizienz prädestiniert diese Interaktionsform für professionelle Benutzer.

Die Interaktion über natürliche Sprache geht von der gewohnten Sprache des Menschen aus. Sie erfordert deshalb keinen zusätzlichen Lernaufwand. Damit ist sie für Gelegenheitsnutzer bzw. Anfänger gut geeignet. Bei ihnen sinkt vor allem bei umfangreichen Datenmengen die Effizienz der Eingabe. Das Hauptproblem bei natürlich-sprachlichen Systemen besteht jedoch in ihrer Interpretation. Die Bedeutung der Eingabe hängt stark vom Kontext ab. Tritt ein solcher mehrdeutiger Fall auf, muss das System einen Klärungsdialog initiieren, der wiederum die Effizienz mindert. Da die natürliche Sprache mit bis zu 300.000 Wörtern ein extrem umfangreiches Vokabular besitzt, können heutige Anwendungssysteme nur einen kleinen Teil kennen und verarbeiten.

Das Gebiet der deskriptiven Interaktionsformen gewinnt in letzter Zeit wieder an Bedeutung. Grund dafür ist die Einführung der Spracheingabe. Die Steuerung von Computern über Sprache, beispielsweise bei Fahrzeugnavigation oder -telefonie, wird auch in Zukunft weiter an Bedeutung gewinnen.

3.4.2.2 Menüs

Menüs zählen zu den deiktischen bzw. selektionsorientierten Interaktionsformen. Bei diesen wird dem Benutzer die Möglichkeit gegeben, aus einem Angebot von Operationen bzw. Objekten auszuwählen (vgl. [ISO9241-14] S. 5). Damit wird das Konzept der Wiedererkennung unterstützt. Dieses basiert darauf, dass ein Mensch bei der Erkennung deutlich leistungsfähiger als bei der Erinnerung ist.

Menüs sind eine Interaktionstechnik, die schnell verstanden wird. Vor allem für Gelegenheitsbenutzer und Anfänger sind sie gut geeignet. Zu den Vorteilen gehören die leichte Erlernbarkeit, die hohe Anwendungsneutralität sowie die Vermeidung von Syntaxfehlern. Zusätzlich ist der Eingabeaufwand sehr gering. Menüs unterstützen das entdeckende Lernen , da sie bisher nicht verwendete und nicht erinnerte Alternativen sichtbar machen (vgl. [Ober94] S. 129). Nachteile sind vor allem die langsame Auswahl und der hohe Platzbedarf auf dem Bildschirm. Menüs sind nur für kleine Auswahlmengen geeignet. Ein weiterer Nachteil ist die Parameterübergabe bei Operationen. Diese sind nur durch aufwendige Kaskadierung, d.h. die Bildung von Untermenüs, oder durch einen zusätzlichen Dialog möglich. Die partielle Darstellung eines Menüs kann zu Orientierungs- und Navigationsproblemen führen.

Menüs sind eine wichtige Interaktionsform, die in fast jeder Anwendung genutzt wird. Aus diesem Grund wird ihnen in der Richtlinie ISO 9241 ein eigener Teil gewidmet.

Das Hauptproblem bei der Verwendung eines Menüs ist ihre Strukturierung. Das Ziel besteht darin, dass der Benutzer so schnell wie möglich den gesuchten Menüeintrag findet. Da meist die Anzahl der verfügbaren Optionen zu hoch ist, um effizient in einem einzigen Menü angezeigt zu werden, ist die Bildung eines hierarchischen Menübaums erforderlich (vgl. [ISO9241-14] S. 7). Jede zusammen angezeigte Gruppe von Menüeinträgen wird als Menü bzw. Untermenü bezeichnet. Unter der **Tiefe** eines Menüs versteht man die vorhandene Anzahl von Untermenüs. Die **Breite** eines Menüs gibt die Anzahl von Menüpunkten der aktuellen Ebene an.

Üblicherweise werden Menüpunkte nach ihrer inhaltlichen Zusammengehörigkeit gruppiert. Das Ziel sollte eine breite Menühierarchie sein, da die Suche deutlich schneller als in tiefen

Hierarchien ist. Hier tritt ein Zielkonflikt ein, da der Entwickler in Versuchung gerät, inhaltlich relativ unabhängige Optionen in einem Menü zu gruppieren. Dem ist jedoch entgegenzuwirken, da bei unlogischer Strukturierung der Zeitvorteil der breiten Hierarchie verloren geht. Um den Entwickler bei diesen Entscheidungen zu unterstützen, wurden umfangreiche Untersuchungen durchgeführt. SHNEIDERMAN fasst diese zusammen und gibt einige Empfehlungen (vgl. [Shne98] S. 257). Grundsätzlich sollte ein Menü aus mindestens drei, maximal jedoch 20 Einträgen bestehen. Bei der Verwendung von relativ vielen Menüeinträgen ist sind Separatoren bzw. Trennlinien einzusetzen (vgl. [Prei99] S. 103). Diese führen zu Untergruppen und beeinflussen die Suchzeit ebenfalls positiv. Die durchschnittliche Tiefe von Menüs sollte zwischen zwei und drei liegen. Werden mehr Ebenen genutzt, so kann das eine Desorientierung des Benutzers nach sich ziehen. Ein Problem tritt auf, wenn bei der Entwicklung des Systems die Anzahl der Menüeinträge noch nicht bekannt ist. In diesen Fällen sollten Maßnahmen getroffen werden, welche die Bildung von schlechten Menühierarchien vermeiden. So kann beispielsweise eine Aufteilung in mehrere Spalten erwogen werden, oder dem Benutzer selbst die Möglichkeit der Anpassung gegeben werden (vgl. Abschnitt 3.4.8).

Ein weiteren wichtigen Anhaltspunkt für die Strukturierung bieten die verbreiteten Konventionen. Diese sind nach Möglichkeit einzuhalten, da sie vielen Benutzern bekannt sind. So befindet sich beispielsweise das Dateimenü jeweils links, während das Hilfemenü grundsätzlich rechts angeordnet ist. Außer der Strukturierung sind Platzierung und der Zeitraum, in dem das Menü sichtbar ist, festzulegen (vgl. [Ober94] S. 127).

Das Konzept der Beschleunigungstasten ist ebenfalls zu unterstützten. Beschleunigungstasten (engl.: accelerator key, shortcut key) sind eine Kombination von Tasten, um eine Menü-Option aufzurufen, ohne dass das entsprechende Menü auf dem Bildschirm angezeigt wird (vgl. [ISO9241-14] S. 4). Damit kann der Zugriff auf einen Menüeintrag wesentlich beschleunigt werden. Die Beschleunigungstaste wird im Menüeintrag rechts vom beschreibenden Text angezeigt. Das System sollte es dem Benutzer gestatten, für Menüeinträge selber Tastenkürzel zu vergeben. Das bedeutet jedoch nicht, dass der Entwickler auf sinnvolle Vorgaben verzichten kann. Für die Hilfefunktion ist beispielsweise die Funktionstaste F1 üblich. Diese Beschleunigungstasten sind auch aktiv, wenn das Menü nicht geöffnet ist. Zur schnellen Navigation in Menüs bietet sich jedoch auch eine Selektion über Anfangsbuchstaben von Menüeinträgen an. Der zu drückende Buchstabe wird in diesem Fall unterstrichen dargestellt.

Häufig werden oft benutzte Menüeinträge in Werkzeugleisten (auch: Symbolleisten) zusammengefasst, die durch den Benutzer beliebig ein- und ausgeblendet werden können. Dabei werden die Menüeinträge durch Icons repräsentiert. Diese sollten auch im Menüeintrag genutzt werden, wobei sie links vom beschreibenden Text platziert werden. Damit wird die Fähigkeit der Wiedererkennung des Anwenders genutzt und ebenfalls eine Beschleunigung bei der Selektion erreicht. Von einer dynamischen Modifikation von Menüs wird abgeraten. So könnten oft genutzte Menüeinträge dynamisch in ihrer Position im Menü nach oben positioniert werden. Kaum gebrauchte Einträge werden entsprechend weiter unten angeordnet. Untersuchungen haben jedoch gezeigt, dass durch die ständige Neuorientierung des Benutzers der erhoffte Geschwindigkeitsvorteil ausblieb.

Außer dem globalen Menü werden in Anwendungen oft lokale Menüs angeboten. Sie werden als Pop-up-Menüs bezeichnet. Diese werden üblicherweise mit der rechten Maustaste aktiviert und erscheinen dynamisch in der Nähe des ausgewählten Objektes. Sie bieten sowohl von der aktuellen Selektion abhängige, als auch globale Optionen. Die Nutzung solcher Menüs ist insofern problematisch, als das sie nicht explizit sichtbar sind. Aus diesem Grund müssen grundsätzlich alle Optionen auch anderweitig aktivierbar sein.

In den letzten Jahren wurden einige interessante Ansätze entwickelt, bestimmte Nachteile von Menüs abzuschaffen. Ein Problem bei der Nutzung von Kontextmenüs besteht darin, dass der aktuelle Bildschirminhalt teilweise verdeckt wird. Abhilfe lässt sich durch Nutzung transparenter Menüs schaffen (vgl. [Prei99] S. 105ff). Bei diesen kann der Benutzer durch den Hintergrund des Menüs „durchsehen“. Im Idealfall lässt sich die Stärke der Transparenz durch den Benutzer beeinflussen. Ein Nachteil dieser Menüs ist, dass sie etwas schwerer lesbar sind, als normale Menüs. Kreisförmige Menüs sind der Versuch, die Suchzeit weiter zu verringern. Vorteilhaft ist dabei, dass alle Menüeinträge von der Kreismitte gleich weit entfernt sind. Zeitgewinn bringt auch, dass ein Benutzer sich die Richtung, in der sich der Menüeintrag befindet besser merken kann, als dessen Position in einem linearem Menü. Während Kreismenüs sich bisher nicht durchsetzen konnten, werden Abreißmenüs vor allem in Grafikanwendungen benutzt. Diese Menüs sind auf dem Bildschirm frei plazierbar. Sie können je nach Bedarf aus- bzw. eingeklappt werden. Damit sind sie schnell im Zugriff, überdecken jedoch den Bildschirminhalt nicht.

3.4.2.3 Direkte Manipulation

Die direkte Manipulation ist „eine Dialogtechnik, durch die der Benutzer den Eindruck erhält, die Objekte am Bildschirm direkt zu bearbeiten, z.B. indem er mit Hilfe eines

Zeigeinstrumentes auf sie zeigt, sie verschiebt und/oder ihre physikalischen Eigenschaften (oder Werte) verändert.“ ([ISO9241-16] S. 4). THIMBLEBY nennt als Hauptmerkmal der direkten Manipulation die sofortige, konsistente und eindeutige Darstellung der manipulierbaren Daten (vgl. [HaTh90] S. 130). Das bedeutet, das verfügbare Objekte und Werkzeuge der Anwendung ständig visualisiert werden. Damit soll dem Benutzer das Gefühl gegeben werden, analog zu Realität arbeiten zu können. Auf Grund der Komplexität eines Anwendungssystems kann in den wenigsten Fällen alles gleichzeitig dargestellt werden. Eine Möglichkeit die Komplexität zu reduzieren, stellt der Mechanismus der Modi dar. Ein Modus definiert einen Zustand des Systems, in dem nur Teile der Daten und Funktionen angezeigt werden. Die direkte Manipulation wird heute in fast allen grafischen Benutzungsschnittstellen als Interaktionstechnik angewandt. In der aktuellen Literatur wird sie oft zusammen mit dem Begriff Virtual Reality (auch: Virtual Environment) genannt. Letzteres wird dabei als konsequente Weiterentwicklung der direkten Manipulation angesehen.

Die direkte Manipulation bietet eine Vielzahl von Vorteilen. Anfänger erlernen hier die Grundfunktionalität schneller als in Systemen, die als Eingabe eine komplexe Syntax verlangen. Auch Gelegenheitsbenutzer können sich die Grundkonzepte besser merken. Werden an Objekten Veränderungen durchgeführt, so ist das Ergebnis direkt erkennbar. Entspricht es nicht den Erwartungen des Benutzers, so kann er es über die Undo-Funktion sofort rückgängig machen (vgl. [Ober94] S. 136). Anschließend kann ein erneuter Versuch unternommen werden. Dieses Rückkopplungsprinzip wird oft mit dem WYSIWYG-Prinzip verknüpft (vgl. Abschnitt 3.4.5). Durch den Verbund von Objekt und Operation, verringert sich die Fehlerrate, da der Benutzer nur Operationen auswählen kann, die für das Objekt auch sinnvoll sind. Damit werden auch deutlich weniger Fehlermeldungen benötigt. Zusätzlich erleben Benutzer weniger Angstgefühle, da die Systeme verständlicher sind und Aktionen gefahrlos ausprobiert werden können. Zutrauen und Sicherheit steigern sich ebenfalls, da der Benutzer als Initiator auftritt und damit das Systemverhalten vorhersehen kann.

Es gibt jedoch auch Einschränkungen. Vor allem bei Routineaufgaben kann die Effizienz der Aufgabenerledigung sinken. Da die Objekte immer direkt mit dem Zeigegerät anzusprechen sind, steigt mit der Anzahl der auszuwählenden Objekte auch der Interaktionsaufwand. Das Problem tritt auch auf, wenn Objekte anhand ihrer Eigenschaften ausgewählt und manipuliert werden sollen. Soll beispielsweise aus tausend Objekten eine Auswahl anhand von drei Eigenschaften vorgenommen werden, so muss bei konsequenter Umsetzung der direkten Manipulation jedes Objekt ausgewählt und bewertet werden. Es ist offensichtlich, das damit

ein unzumutbarer Interaktionsaufwand entsteht. Ein Lösungsansatz besteht darin, kleinere Teildialoge zu entwickeln, die eine Filterung von Objekten nach verschiedenen Eigenschaften durchführen. Die Ergebnismenge des Filters kann dann einfach komplett selektiert und manipuliert werden. Als Beispiel kann hier der Dateisuche unter MS Windows genannt werden. In dem oben genannten Beispiel können deskriptive oder deiktische Interaktionsformen deutlich effizienter sein. Ein weiterer Nachteil besteht darin, dass alle Operationen des gewählten Objektes visualisiert werden müssen. Das ist jedoch nicht immer möglich, da eine Überflutung des Benutzers die Folge ist. Aus diesem Grund wird nur eine gefilterte Auswahl von Operationen angeboten. Damit wird unter Umständen nicht die volle Funktionalität kennengelernt und genutzt. Nachteilig ist auch der Umstand, dass die grafische Symbolik kaum normiert ist. Ein großes Problem kann auftreten, wenn die Grenzen der verwendeten Metapher vom Benutzer nicht erkannt werden. Dies führt dann zu falschen mentalen Modellen und damit zu Fehlhandlungen.
Durch eine intensive Aufgabenanalyse ist festzustellen, in welchen Bereichen des Dialogs die direkte Manipulation nachteilig sein kann. Hier ist dann dem Benutzer eine alternative Interaktionsform anzubieten.

3.4.3 Antwortverhalten

Unter dem Antwortverhalten versteht man die zeitliche Reaktion des Anwendungssystems auf Benutzereingaben. Das Antwortverhalten ist für interaktive Systeme von großer Bedeutung. Treten hierbei unerwartete Verzögerungen auf, verunsichert das den Benutzer. Mit steigender Häufigkeit dieses Verhaltens kann das den Benutzer verärgern und sogar frustrieren. Andererseits kann eine zu schnelle Systemreaktion den Benutzer zu hastigem arbeiten animieren. Die Folge sind schlechterer Lernerfolg, oberflächliches Lesen von Systemausgaben und Eingabefehler. Der Zeitgewinn durch das schnellere Antwortverhalten geht damit wieder verloren (vgl. [Shne98] S. 352). Ein Studie mit 4.448 Computerbenutzern ergab, dass das Antwortverhalten die wichtigste Einflussgröße auf die Zufriedenheit darstellt (vgl. [Goul+97] S.232). Eine höhere Benutzerzufriedenheit führt zu größerer Motivation und damit zu einer effizienteren Aufgabenerledigung.
Die wichtigste Teilgröße des Antwortverhalten eines Computersystems ist die Antwortzeit (engl. response time). Diese gibt den Zeitraum zwischen dem Ende der Eingabe des Benutzers und dem Beginn der Ausgabe des Systems an.

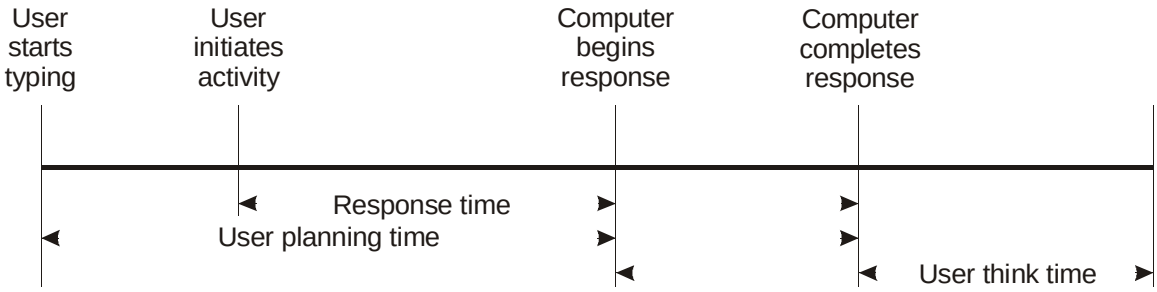


Abbildung 4 Modell des Antwortverhalten von Computersystemen (vgl. [Shne98] S. 353)

Wie in Abbildung 4 zu sehen ist, plant der Benutzer schon während der Eingabe der Daten weitere Schritte. Dieser Planungsprozess wird auch innerhalb der Antwortzeit des Computersystems fortgesetzt. Bei komplexen Aufgaben kann eine längere Antwortzeit zu einer präziseren Planung und damit zu weniger Fehlern führen. Bei einfachen Aufgaben wird der Benutzer durch eine lange Antwortzeit behindert. Dabei wirkt sich vor allem der Informationsverlust des Kurzzeitgedächtnisses aus. Die verlorenen Informationen müssen erneut beschafft werden, was zusätzliche Zeit benötigt und Kosten verursacht. Das bedeutet, dass eine optimale Antwortzeit sowohl durch den Benutzer, als auch durch die Komplexität der Aufgabe bestimmt wird.

Empirische Untersuchen haben einige Faktoren herausgearbeit, welche die Bestimmung einer günstigen Antwortzeit beeinflussen. Vor allem Erfahrungen mit ähnlichen Systemen rufen entsprechende Erwartungen bei dem Benutzer hervor. MILLER hat heraus gefunden, dass Benutzer bei einer Antwortzeit zwischen zwei und vier Sekunden eine Abweichung von 8 Prozent bemerken (vgl. [Shne98] S. 359). Ist die Antwortzeit länger als erwartet, wird der Benutzer ungeduldig. Die Akzeptanz gegenüber dem Anwendungssystem sinkt. Ist sie jedoch überraschend kurz, so wird der Benutzer skeptisch, ob alles korrekt abgelaufen ist. Wird beispielsweise der aktuelle Arbeitsstand durch den Benutzer gesichert, so erwartet dieser eine kurze, aber merkbare Antwortzeit. Das völlige Fehlen dieser Verzögerung kann in dem Benutzer das Gefühl einer Fehlfunktion wecken. Einen Einfluss auf die Antwortzeit haben auch persönliche Merkmale des Benutzers. So akzeptieren Anfänger eher längere Wartezeiten als Experten. Weitere Faktoren sind Alter, momentane Belastung oder die Tageszeit. Die untere Grenze für die Antwortzeit legt die Rechenleistung des benutzten Computersystems fest. Im allgemeinen sollte die Antwortzeit unter einer Sekunde liegen. Die Obergrenze liegt bei drei bis vier Sekunden. Ab dieser Zeitspanne wird durch die ISO 9241 eine entsprechende Systemmeldung verlangt. Diese informiert den Benutzer über den aktuellen Zustand des

Systems. Verbreitete Techniken sind dabei Fortschrittsbalken oder Mauszeiger mit sich zeitlich verändernder Darstellung. Bei Microsoft Windows ist das beispielsweise eine sich drehende Sanduhr. Letzteres wird genutzt, wenn der für die Funktion benötigte Zeitraum nicht im voraus berechnet werden kann.

Im Prototyp wurde für diesen Anwendungsfall die Klasse **`ProgressBar`** entwickelt. Diese zeigt ein Fenster mit einem Fortschrittsbalken an. Ist eine genaue Anzeige des Systemzustandes auf Grund fehlender Granulierung der Prozessschritte nicht möglich, so kann eine Zeitspanne angegeben werden, in welcher der Balken 100 Prozent erreicht. Für die Ermittlung dieser Zeitspanne kann der Prozess während der ersten Ausführung überwacht und die tatsächlich benötigte Zeit gemessen werden. Bei wiederholter Abarbeitung des Prozesses kann diese als Vorgabe für den Fortschrittsbalken genutzt werden. Sind Messungen nicht sinnvoll, da die Prozessdauer bei jeder Ausführung schwankt, sollte die Anzeige durch einen sich verändernden Mauszeiger realisiert werden.

Die Antwortzeit lässt sich nur schwer messen, da sie sowohl von hardware- als auch softwaretechnischen Einflüssen abhängt. Statistische Messungen an Prototypen ermöglichen es jedoch, Aussagen zu treffen. Die Zielgröße für die Evaluierung von Antwortzeiten stellt die Produktivität dar. Sie wird durch die Anzahl der korrekt durchgeführten Aufgaben pro Zeiteinheit bestimmt. Damit wird sie durch eine höhere Fehlerrate negativ beeinflusst. Das Optimum ist erreicht, wenn die Kosten für eine weitere Verbesserung der Antwortzeit höher sind, als die Einsparungen durch höhere Produktivität. Damit muss die durch das System mögliche minimale Antwortzeit nicht immer die günstigste sein. Tendenziell lässt sich jedoch feststellen, dass die zu langen Antwortzeiten den größeren Problembereich darstellen.

3.4.4 Icons

Icons stellen einen Grundbestandteil grafischer Benutzungsschnittstellen dar. Sie sind symbolische Darstellungen von Objekten bzw. Operationen. Im deutschen Sprachgebrauch werden sie synonym als Piktogramme bezeichnet. Die wörtliche Übersetzung aus dem lateinischen lautet *geschriebenes Bild*. Der DUDEN definiert **Piktogramme** als „formelhaftes, graphisches Symbol mit international festgelegter Bedeutung.“ ([Stau87] S. 5) Nach ISO 11581 ist ein Icon „eine Graphik auf einem Sichtanzeigegerät, das eine Funktion eines Rechnersystems repräsentiert.“ ([ISO11581-1] S. 6) Das wichtigste Merkmal von Piktogrammen ist ihre abstrakte Darstellung. Damit sind sie gegenüber Fotos oder Bildern nicht so anschaulich. Ihre Bedeutung, d.h. was sie repräsentieren sollen, wird vom Benutzer

hingegen besser erkannt. Durch die grafische Darstellung fallen Icons dem Anwender leicht auf. Deshalb werden wichtige bzw. oft benötigte Funktionen oder Objekte mit Hilfe von Icons dargestellt.

Icons entsprechen durch ihre abstrahierte Darstellung einer symbolhaften Sprache. Solche Sprachen werden in allen Bereichen des Lebens genutzt. So ist die Beschilderung im Straßenverkehr oder von Bahnhöfen und Flugplätzen ein Beispiel für die Verwendung von Piktogrammen. Damit können Kenntnisse aus diesen Wissensbereichen auch bei den Icons für die Benutzungsschnittstelle eines Anwendungssystems angewendet werden. Durch den bildhaften Charakter können Piktogramme in verschiedenen Kulturbereichen verstanden werden. Icons sind demzufolge nicht nur bei Vorhandensein von verschiedenen Benutzergruppen ideal anzuwenden. Sie tragen auch zur Internationalisierung des Softwaresystems bei. Bei der Entwicklung ist allerdings zu beachten, dass beispielsweise bestimmte Farben in manchen Ländern eine unterschiedliche Bedeutung besitzen.

Die Entwicklung von Icons bzw. Icon-Sätzen ist ein aufwendiger Prozess, der nicht unterschätzt werden darf. Ein Icon-Satz besteht aus einer Gruppe von inhaltlich zusammengehörigen Icons. Diese Zusammengehörigkeit sollte auch durch die grafische Darstellung deutlich gemacht werden. Bestimme Aufgabenbereiche finden sich in vielen Anwendungssystemen wieder. Ein Beispiel dafür sind die Funktionen Undo, Ausschneiden und Einfügen. In diesen Fällen existieren bereits Piktogramme, die dem Benutzer bekannt sind. Um eine anwendungsübergreifende Konsistenz zu erreichen, sollten sie berücksichtigt werden (vgl. [Stau87] S. 99). So wird für die Funktion Drucken im allgemeinen ein Icon mit einem Drucker genutzt. Innerhalb eines Icon-Satzes sollten gleiche Farben sowie einheitliche Symbolgrößen und Ränder benutzt werden. Trotz der Gleichartigkeit innerhalb des Satzes müssen die einzelnen Piktogramme klar voneinander unterscheidbar sein. Durch einen klaren Kontrast zum Hintergrund wird das Erkennen der Icons ebenfalls erleichtert. Die verschiedenen Icon-Sätze sind innerhalb eines Anwendungssystem konsistent zu gestalten. Dabei ist die Definition einer Farbtabelle hilfreich. Die Erstellung bzw. Produktion von Piktogrammen wird idealer Weise mit Hilfe der Benutzer durchgeführt (vgl. [Stau87] S. 95). So können bereits vorhanden mentale Modelle Vorgaben über die Darstellung liefern. Es ist zu beachten, dass der Benutzer die Funktion des Icons erkennen muss, nicht der Designer. Nach der Entwicklung von Piktogrammen ist im Rahmen der Evaluation sicher zustellen, dass die Symbole richtig gedeutet werden. STAUFER gibt in seinem Buch einen detaillierten Überblick über mögliche Methoden. Im wesentlichen wird die Erkennbarkeit und

Verständlichkeit bewertet. Die ISO 11581 nennt mit Unterscheidbarkeit, Erlernbarkeit, Leserlichkeit und Wiedererkennbarkeit weitere Kriterien, die eine Beurteilung des Icon erlauben (vgl. [ISO11581-1] S. 6f). Quantifiziert werden diese Eigenschaften durch Zeitmessungen, Zählung von Fehlern und Übereinstimmung zwischen der erkannten semantischen Bedeutung und der tatsächlichen Funktion des Piktogrammes (vgl. [Stau87] S. 119). Der Gesamtprozess der Piktogrammentwicklung wird in Anhang A.8 dargestellt.

Die Entwicklung und Verwaltung von Icons kann bei umfangreichen Projekten ein nicht unerhebliches Ausmaß annehmen. Aus diesem Grund existieren Tools, die diese Aufgaben übernehmen. Moderne Softwareentwicklungsumgebungen bieten in der Regel einen integrierten Icon-Editor an.

Einen weiteren großen Vorteil, den Icons bieten, ist die Möglichkeit, auch Zustände bzw. Modi anzeigen zu können. Dazu wird ihr Aussehen geändert. Häufig wird diese Technik bei Ein/Aus Zuständen eingesetzt. Meist wird durch eine Änderung des Hintergrundes bzw. der Farben der jeweilige Zustand angezeigt. Es besteht auch die Möglichkeit, das Symbol selbst zu ändern. Allerdings ist zu beachten, dass eine zu große Abweichung Nachteile bei der Wiedererkennung nach sich zieht. Unter Umständen wird dann vom Benutzer der Zusammenhang zwischen beiden Zuständen nicht mehr erkannt, was zu Einbußen bei der Effizienz führen kann.

Ein Nachteil von Icons ist, dass auch momentan nicht verfügbare Aktionen bzw. Objekte angezeigt werden. Um diesen Effekt abzuschwächen, werden diese Icons üblicherweise grau dargestellt, um ihre Inaktivität anzuzeigen. Weiterhin wird für die Erkennung von Piktogrammen ein Vorwissen des Benutzers vorausgesetzt. Eine weitere Schwachstelle ist die Mehrdeutigkeit. Es wurde zwar hinreichend beschrieben, wie dieser Nachteil möglichst vermieden werden kann, auszuschließen ist er jedoch nie.

Icons werden auch zusammen mit Schaltern oder in Menüs verwendet. Auch hier gelten die oben gemachten Aussagen. Ist an anderer Stelle ein Zugriff auf die gleichen Funktionen oder Objekte möglich, so sind jeweils die gleichen Icons zu verwenden. Ein Beispiel dafür ist die Verwendung von Symbolleisten für den Schnellzugriff von wichtigen Menüpunkten.

3.4.5 WYSIWYG-Prinzip

Das WYSIWYG-Prinzip (What you see is what you get) bedeutet, dass die Benutzereingaben auf dem Bildschirm so dargestellt werden, wie sie später auf dem Drucker erscheinen würden

(vgl. [Prei99] S. 120ff). Ursprünglich für Textverarbeitungen entwickelt, wird dieses Prinzip heute in vielen Softwaresystemen integriert.

Führt ein Benutzer eine Manipulation an einem Objekt durch, so werden die Auswirkungen und damit das Ergebnis sofort sichtbar. Diese sofortige Rückkopplung führt dazu, dass Fehler schneller erkannt werden. Vor allem vom System erkennbare Fehler können dem Benutzer unmittelbar angezeigt werden. Dieser kann dadurch sofort zielgerichtete Gegenmaßnahmen einleiten, ohne weitere Operationen auszuführen, die auf den Fehler aufbauen und damit unsinnig wären. Ein weiterer Vorteil dieser Technik ist, dass zusätzliche Informationen, die die Darstellung betreffen, nicht auf dem Bildschirm erscheinen. So kann auf die explizite Anzeige von Formaten oder Positionen verzichtet werden, da der Benutzer diese implizit aus der Darstellung erkennen kann. Damit wird eine größere Übersichtlichkeit erreicht.

Das WYSIWYG-Prinzip hat jedoch auch Nachteile. Ein Problem stellt die nie ganz exakte Übereinstimmung zwischen Bildschirmausgabe und Ausdruck dar. Diese beruht auf den physikalischen Charakteristika der unterschiedlichen Geräte. Vor allem bei farbigen Grafiken tritt dieses Problem auf. Die konsequente Umsetzung des WYSIWYG-Prinzip kann sich bei der Erledigung bestimmter Aufgaben auch hinderlich auswirken. So ist zum Beispiel bei der Verfassung eines längeren Textes die ständige Darstellung der eingefügten Grafiken störend, da sie sowohl Platz auf dem Bildschirm als auch Computerressourcen beanspruchen. Ferner können für bestimmte Aufgaben Informationen über die Darstellung der Objekte für den Benutzer wichtig sein. Beispiele dafür sind die Gliederung eines Dokumentes oder die hierarchische Struktur eines CAD-Modells.

Aus den genannten Gründen ergibt sich die Forderung, dass WYSIWYG-Ansichten nicht die einzige im Softwaresystem angebotene Sicht sein darf. So können diese Ansichten mit zusätzlichen Informationen angereichert werden, was jedoch das Konzept durchbrechen würde. Eine andere Lösung ist das gleichzeitige Angebot von Struktur- und WYSIWYG-Ansicht. Beide Sichten müssen in diesem Fall jedoch synchronisiert werden. Der einfachste Ansatz ist, es dem Benutzer zu ermöglichen, zwischen den verschiedenen Sichten wählen zu können. Dabei entspricht jede Sicht einem Modus. Das WYSIWYG-Prinzip wird häufig mit der Technik der direkten Manipulation verknüpft (vgl. Abschnitt 3.4.2.3).

3.4.6 Hilfesystem

Es gibt praktisch kein Anwendungssystem, welches umfassend verständlich ist. Aus diesem Grund müssen dem Benutzer Möglichkeiten zur Unterstützung geboten werden. Das können

Schulungen durch Experten, Handbücher oder Hilfesysteme sein. Im Rahmen dieser Arbeit werden letztere genauer diskutiert.

Als **Hilfesystem** wird jedes Programm bezeichnet, das bei der Benutzung eines interaktiven Systems durch explizite Erklärungen hilft (vgl. [Herc94] S. 161). Es erklärt den Systemzustand, d.h. vorhandene Objekte, ihre Eigenschaften und Beziehungen untereinander. Zusätzlich wird die Funktionalität beschrieben, also die Operationen mit denen die Eigenschaften von Objekten manipuliert werden können. Hilfesysteme bieten eine Reihe von Vorteilen (vgl. [Shne98] S. 425f). Sie sind, im Gegensatz zu Handbüchern, während der Arbeit mit dem Anwendungssystem ständig verfügbar. Durch Aktualisierungen, beispielsweise mit Hilfe des Internet, können sie kostengünstig immer auf dem neuesten Stand gehalten werden. Der Einsatz von Suchmaschinen ermöglicht das Anbieten spezifischer, aufgabenabhängiger Informationen. Innerhalb des Hilfesystem können multimediale Komponenten wie Grafik, Ton oder Animation verwendet werden. Es existieren jedoch auch Nachteile. So sind längere Texte vor allem auf Monitoren mit niedriger Auflösung nicht so gut lesbar, wie auf Papier gedruckter Text. Dabei können längere Bearbeitungszeiten von 15 bis 30 Prozent auftreten. Die niedrigeren Auflösungen können auch dazu führen, dass Grafiken und Bilder schlechter dargestellt und damit fehlinterpretiert werden. Bei Anfängern können Probleme bei der Anwendung des Hilfesystem auftreten, da sie mit der Benutzungsschnittstelle des Hilfesystems nicht vertraut sind. Ein weiterer Nachteil besteht darin, dass das Hilfesystem Platz auf dem Bildschirm benötigt. Dadurch können Teile des Anwendungssystems verdeckt werden, was zu einer höheren Belastung des Kurzzeitgedächtnisses des Benutzers führt. Aus diesen Gründen sollten sowohl Online-Hilfe als auch Handbücher angeboten werden. Moderne Konzepte ermöglichen es, beides aus einem einzigen Datenpool zu erzeugen, so dass nur ein geringer zusätzlicher Aufwand entsteht.

Sieht man Hilfesysteme als eine besondere Form von Anwendungssystemen, so lassen sich die gemachten Aussagen über ergonomische Aspekte auch in diesem Bereich anwenden. Da jedoch Aufgaben und Ziele eines Hilfesystems bekannt sind, können einige speziellere Aussagen getroffen werden.

Wie ein Dialog kann auch das Hilfesystem nach Art der Aktivierung unterschieden werden. Passive Hilfe wird durch den Benutzer initiiert. Sie ist bei fast allen Softwaresystemen integriert. In den Anfängen der Hilfesysteme waren dies meist **Online user manuals,** die lediglich die Handbücher in elektronischer Form anbieten (vgl. [Shne98] S. 411). Diese Hilfesysteme waren schlecht zu nutzen, da die angebotenen langen Texte sehr unüberschaubar

waren. Eine erste Verbesserung war die Einführung von Zugriffsstrategien. Sie ermöglichen dem Benutzer durch eine Auswahlfunktion gezielt auf Teile der Hilfeinformationen zuzugreifen. Solche Hilfesysteme werden als **Online help facility** definiert. Bei ihnen ist mindestens ein statischer Index vorhanden. Implementierte Suchfunktionen, die mit natürlich-sprachlichen Anfragen umgehen können, stellen ein leistungsstärkeres Instrument für den Benutzer dar. Das Anfrageergebnis stellt einen dynamisch erzeugten Index dar. Dieser zeigt auf die entsprechenden Hilfetexte, die selbst aber weiterhin statisch sind. Das Problem bei natürlich-sprachlichen Anfragen ist die Schwierigkeit, die Anfrage syntaktisch und semantisch korrekt zu erfassen. So können Tipp- oder Schreibfehler ein korrekte Auswertung verhindern. Ebenfalls problematisch stellt sich die Auswertung des inhaltlichen Zusammenhangs bei Eingabe von Wortgruppen dar. Die angesprochenen Probleme sind im Rahmen des Information Retrieval Thema intensiver Forschungen.

Aktive Hilfesysteme übernehmen selbst die Initiative (vgl. [Herc94] S. 165ff). Dazu ist eine Überwachung des Dialogs zwischen Benutzer und Anwendungssystem notwendig. Dabei sind einfache Fehler, wie das Drücken einer falschen Taste einfach zu erkennen. Fehlerhafte Parameter oder unzulässige Nutzung von Operationen können ebenfalls mit relativ wenig Aufwand erkannt werden. Deutlich schwieriger ist die Erkennung von logischen Fehlern. Hierzu muss bereits explizites Wissen über Anwendungsbedingungen vorhanden sein. Wünschenswert, aber ebenfalls schwierig ist das Feststellen einer suboptimalen Nutzung des Anwenders. Das Hilfesystem muss dafür aufwendige Mechanismen zur Wissensrepräsentation beinhalten. Diese Mechanismen sollen die Ziele des Benutzers erkennen und die von ihm ausgeführten Aktionen auf Optimalität prüfen. Werden Defizite festgestellt, aktiviert sich das Hilfesystem und weist den Anwender auf alternative Lösungswege hin. Diese Art der Hilfesysteme steht heute noch am Anfang ihrer Entwicklung. Aktive Hilfe ist vor allem bei unerfahrenen Benutzern einzusetzen. Experten benötigen diese Art der Hilfestellung nicht, werden u.U. sogar dadurch behindert (vgl. [ISO9241-13] S. 10). Aus diesem Grund muss der Benutzer die aktive Hilfe aus- und wieder einschalten können.

Im allgemeinen bieten Hilfesysteme ihre Informationen statisch an. D.h., dass die Inhalte vom aktuellen Zustand der Anwendung unabhängig sind. Das ist jedoch für den Benutzer unbefriedigend. Möchte dieser z.B. vom Hilfesystem Alternativen zur Fortführung des aktuellen Arbeitsstandes angezeigt bekommen, ist dies mit statischen Hilfeinformationen schwer zu lösen. Die Anfrage nach zur Zeit verfügbaren Operationen im Anwendungssystem stellt ein weiteres Anwendungsbeispiel für diesen Problembereich dar. Teilweise lässt sich

das Problem mit Zugriffsstrategien umgehen, günstiger ist jedoch die dynamische Bereitstellung von Hilfeinformationen. Damit wird dem Benutzer ermüdendes Durchsuchen des Informationsraumes abgenommen. Ein dynamisches Hilfesystem muss bereits während der Entwicklung eingeplant werden, da es eine Repräsentation des Anwendungszustandes benötigt. Die dynamischen Hilfesysteme sind auch ein guter Ausgangspunkt, individuelle Hilfeinformationen anzubieten. So kann in Abhängigkeit der Benutzergruppe ein bestimmter Detaillierungsgrad der Hilfeinformation angeboten werden. Noch weiter geht der Ansatz, die bereits in Anspruch genommenen Hilfeleistungen eines einzelnen Benutzer zu protokollieren und bei weiterer Hilfeanforderung zu beachten. Der Ansatz der individuellen Hilfe unterstützt auch die dynamische Änderung und Erweiterung von Hilfeinformationen durch den Benutzer. So ist es wünschenswert, dem Anwender die Möglichkeit zu geben, eigene Texte und Hilfethemen hinzuzufügen (vgl. [ISO9241-13] S. 11).

Hilfesysteme sollten ihre Informationen grundsätzlich in einem eigenen Darstellungsbereich ausgeben. Das ermöglicht die parallele Nutzung von Anwendungs- und Hilfesystem. Außerdem wird damit der Ansatz unterstützt, Hilfesysteme anwendungsunabhängig zu entwickeln. Einerseits kann damit eine eigenständige, für Hilfesysteme typische Schnittstelle entwickelt werden, andererseits können damit die Hilfesysteme in verschiedensten Anwendungen wiederverwendet werden. Damit ist das Hilfesystem dem Benutzer auch und vor allem gerade bei neuen Anwendungssystemen bereits vertraut. Die Kostensenkung bei der Entwicklung ist ein positiver Nebeneffekt.

Zur Umsetzung von Hilfesystemen eignen sich im besonderen Maße Hypertext- bzw. Hypermediasysteme. Hypermediasysteme dienen zur Erkundung eines Informationsraum. Sie basieren auf multimedialen Datenbanken (vgl. [Komm+98] S. 3) Diese bestehen aus vernetzten Informationen, die in Text-, Grafik- und Tonform vorliegen können.

3.4.7 Fehlerbehandlung

Bei der Nutzung von Softwaresystemen treten, obwohl nicht gewünscht, Fehlersituationen auf. Diese definieren sich dadurch, dass der Benutzer das beabsichtigte Ziel nicht erreicht. Fehler können aufgrund von Funktionsstörungen des Anwendungssystems oder durch den Benutzer selbst verursacht werden (vgl. [ISO9241-13] S. 8). Eine Fehlhandlung des Benutzers weist auf eine mangelnde Anpassung zwischen Mensch und Computersystem hin. Im Rahmen dieser Arbeit wird auf die zweite Gruppe von Fehlern näher eingegangen. Generelle Aussagen über Darstellung und Behandlung lassen sich jedoch auch auf Systemfehler übertragen.

Grundsätzlich existieren zwei Strategien um mit einem Fehlerproblem umzugehen (vgl. [Kens+95] S. 219). Bei der **Fehlervermeidung** wird versucht, durch gutes Softwaredesign die Fehlerrate klein zu halten. Das **Fehlermanagement** zielt darauf ab, die Auswirkungen eines Fehlers möglichst gering zu halten. Dazu gehört sowohl die Vermeidung von Folgefehlern und negativen Effekten als auch die Minimierung des Fehlerbewältigungsaufwandes.

Die handlungstheoretische Fehlertaxonomie ist eine der bekanntesten Klassifizierungen von Fehlern (vgl. [Kens+95] S. 218 und [BrRu94] S. 199ff). Dabei werden sieben verschiedene Fehlhandlungen identifiziert. An dieser Stelle wird eine einfachere Aufteilung in lediglich zwei Klassen vorgenommen. Die Erste beruht auf der Unaufmerksamkeit des Benutzers. Dazu gehören die Gewohnheits-, Unterlassens- und Erkennensfehler der eben angesprochenen Fehlertaxonomie. Die bei hochautomatisierten Operationen auftretenden Bewegungsfehler sind ebenfalls dieser Klasse zuzuordnen. Die zweite Klasse geht auf ein mangelndes Verständnis des Benutzers zurück. Hierzu gehören vor allem Denk- und Urteilsfehler.

Die meisten Fehler treten bei Routinehandlungen auf und gehören ersterer Klasse an. Das liegt daran, dass bei „automatisierter“ Abarbeitung von Aktionen die Aufmerksamkeit des Benutzer nachlässt. Es finden meist mehrere mentale Prozesse gleichzeitig statt, wobei es zu Interferenzen zwischen diesen kommen kann. Beispielsweise kann ein zu weites Vorausplanen zur Folge haben, dass der aktuelle Stand der Handlung vergessen wird. Im Rahmen der Fehlervermeidung kann diesen Fehlern durch das Einfügen von Kontrollpunkten entgegen gewirkt werden. Solche Kontrollpunkte können beispielsweise zusätzliche Eingabebestätigungen sein. Damit wird der Benutzer bewusst zur Aufmerksamkeit gezwungen. Kontrollpunkte sollten vor allem vor kritischen Aktionen eingefügt werden, d.h. Aktionen die bei unbedachter Ausführung schwerwiegende Folgen haben können. Bei Routinehandlungen können im allgemeinen effizient Fehlervermeidungsstrategien eingesetzt werden. BRODBECK und RUPIETTA nennen als wichtigste Techniken konsistente Dialoggestaltung, Einschränkung von Freiheitsgraden, Sicherheitsrückfragen und Sicherheitskopien (vgl. [BrRu94] S. 205ff).

Fehler aufgrund von mangelndem Verständnis sind durch das Anwendungssystem schwer zu erkennen. Hier muss durch ein gutes ergonomisches Design eine Vermeidung angestrebt werden. Vor allem der Einsatz einer adäquaten Metapher kann dabei unterstützend wirken. Ein umfassendes Hilfesystem kann dem Benutzer dabei helfen, sich vor der Ausführung von Operationen genauer über den Problembereich zu informieren.

Es ist jedoch unwahrscheinlich, eine vollständige Fehlervermeidung zu erreichen. Dies liegt auch daran, dass es durchaus Handlungen gibt, bei denen der Aufwand zur Bewältigung eines Fehlers geringer ist als die Vermeidung des Fehlers selbst. Die Beseitigung von Fehlern wird durch ein Fehlermanagement umgesetzt. Dieses umfasst die Fehlerentdeckung, Fehlererklärung und Fehlerbehebung. Grundsätzlich können Fehler sowohl vom Benutzer als auch vom System erkannt werden. Kann ein Fehler nur durch den Anwender entdeckt werden, sollten ihm entsprechende Diagnosewerkzeuge wie WYSIWYG-Editoren oder Vorschau- und Simulationsfunktionen zur Verfügung gestellt werden (vgl. [ISO9241-13] S. 9). Der Klärungsdialog wird in diesem Fall durch den Benutzer initiiert und stellt im allgemeinen das Hilfesystem dar. Wird der Fehler durch das System erkannt, sollte dies einen Klärungsdialog in Form einer Fehlermeldung initiieren (vgl. Anhang A.6). Diese muss vermitteln, was falsch ist, die Ursache nennen und Korrekturmaßnahmen beschreiben. Die Behebung des Fehlers kann ebenfalls sowohl durch das System selbst oder den Benutzer erfolgen. Die automatische Korrektur darf sich jedoch nicht der Kontrolle des Anwenders entziehen. Im Rahmen des Fehlermanagement sollte eine Undo-Funktion angeboten werden.

In einer empirische Studie wurden bei der Nutzung einer Adresslösung 305 Nutzungsprobleme festgestellt (vgl. [Kens+95] S. 222ff). In der anschließenden Auswertung konnten für 210 (68,9%) Fehlervermeidungs- und für 295 (96,7%) Fehlermanagementvorschläge entwickelt werden. Der Fehlerbehandlung sollte demzufolge vor allem im Rahmen der Evaluierung eine entsprechend hohe Priorität eingeräumt werden.

3.4.8 Individualisierung

Ein Softwaresystem wird für eine Vielzahl von unterschiedlichen Benutzern entwickelt. Wie bereits in Abschnitt 3.2.2 beschrieben, können diese in Benutzergruppen gegliedert werden. Eine Anpassung der Benutzungsschnittstelle an diese Gruppen wird bereits während der Entwicklung des Softwaresystems vorgesehen. Die konkrete Arbeitssituation kann jedoch trotz intensiver Aufgaben- und Benutzeranalyse nicht von vornherein umfassend beschrieben werden (vgl. [Oppe94] S. 236). Daraus ergibt sich, dass die angebotene Schnittstelle im Einzelfall nicht optimal ist. Um maximale Effizienz zu ermöglichen, muss die Benutzungsschnittstelle anpaßbar sein. Die dabei auftretenden Aspekte werden unter dem Begriff Individualisierung von Benutzungsschnittstellen zusammengefasst. OPPERMANN weist auf einen weiteren Grund für Individualisierung hin. So beinhaltet ein vollkommenes Fehlen von Freiheitsgraden die Gefahr der Abstumpfung und Dequalifizierung des Benutzers. Ein zu

hoher Grad Individualisierung ohne Strukturierungshilfe führt dagegen zu Orientierungslosigkeit und ist demzufolge ebenfalls zu vermeiden.

Grundsätzlich lassen sich zwei Arten von Individualisierung bestimmen. Wird die Anpassung vom Benutzer selbst vorgenommen, spricht man von **adaptierbaren** Benutzungsschnittstellen (vgl. [Herc94] S. 175ff). Die zweite Möglichkeit besteht in der Anpassung durch das Softwaresystem selbst. In diesem Fall liegt eine **adaptive** Benutzungsschnittstelle vor. Ähnlich wie bei den Hilfesystemen, befindet sich diese Form noch in einem frühen Entwicklungsstadium.

Eine Individualisierung kann auf verschiedenen Ebenen stattfinden. Die einfachste Form ist die Anpassung des Anwendungssystems in seinem Erscheinen. D.h., der Benutzer wählt nach seinen Präferenzen Farben, Schriftarten, Sprache, Positionen einzelner Bildschirmelemente u.a. aus (vgl. Abbildung 5). Das Anwendungssystem sollte am Anfang sinnvolle Voreinstellungen anbieten. Diese Einstellungen können dann je nach Bedarf in einem einfachen Dialog durch den Benutzer geändert werden. Die Festlegung von Synonymen stellt ebenfalls eine einfache Form der Individualisierung dar. So können bestimmte Funktionsbezeichnungen bei Benutzern falsche Assoziationen auslösen. Durch einfaches Umbenennen kann dieses Problem gelöst werden. Eine weitere Form der Individualisierung ist die Vorgabe von Standardwerten durch den Benutzer. Dies ist vor allem bei Eingaben in Masken zweckmäßig. Durch geschickte Vorgaben lassen sich erhebliche Effizienzsteigerungen erreichen. Eine sehr mächtige Art der Individualisierung ist die eigene Definition von Operationen. Im alltäglichen Arbeitsablauf werden bestimmte Sequenzen von Operationen immer wieder ausgeführt. Fasst man diese einzelnen Schritte zu einem einzigen zusammen, erhält man ein **Makro**. Für das Erstellen von Makros wird der entsprechende Arbeitsablauf aufgezeichnet und mit einem Namen versehen. Der Benutzer kann nun unter diesem Namen die einzelnen Operationen mit einer einzigen Aktion auslösen. Kann man die Makros parametrisieren wird dieser Mechanismus noch leistungsfähiger. In umfangreichen Anwendungssystemen werden heute ganze Makrosprachen integriert. Das bedeutet, dass der Benutzer eigene Programme schreiben kann. Der Übergang vom reinen Makro bis zur Programmierung ist also fließend.

Oft wird vom Softwaresystem ein Mechanismus angeboten, der die Einstellung von Modi erlaubt. So kann der Benutzer durch eine einzige Einstellung eine Vielzahl an Individualisierungsmöglichkeiten durch das System vornehmen lassen. Beispielsweise wird in

einem Anfänger-Modus üblicherweise der Funktionsumfang eingeschränkt und die Interaktionsinitiative hauptsächlich auf Seite des Softwaresystems liegen.

Die verschiedenen Formen der Individualisierung verlangen vom Benutzer entsprechende Fähigkeiten. Zu ihrer Realisierung werden Dialoge benötigt, die die Komplexität des Anwendungsprogramms steigern. Es ist bei der Benutzer- und Aufgabenanalyse klar festzustellen, welcher Grad der Individualisierung sinnvoll und dem Benutzer zumutbar ist.

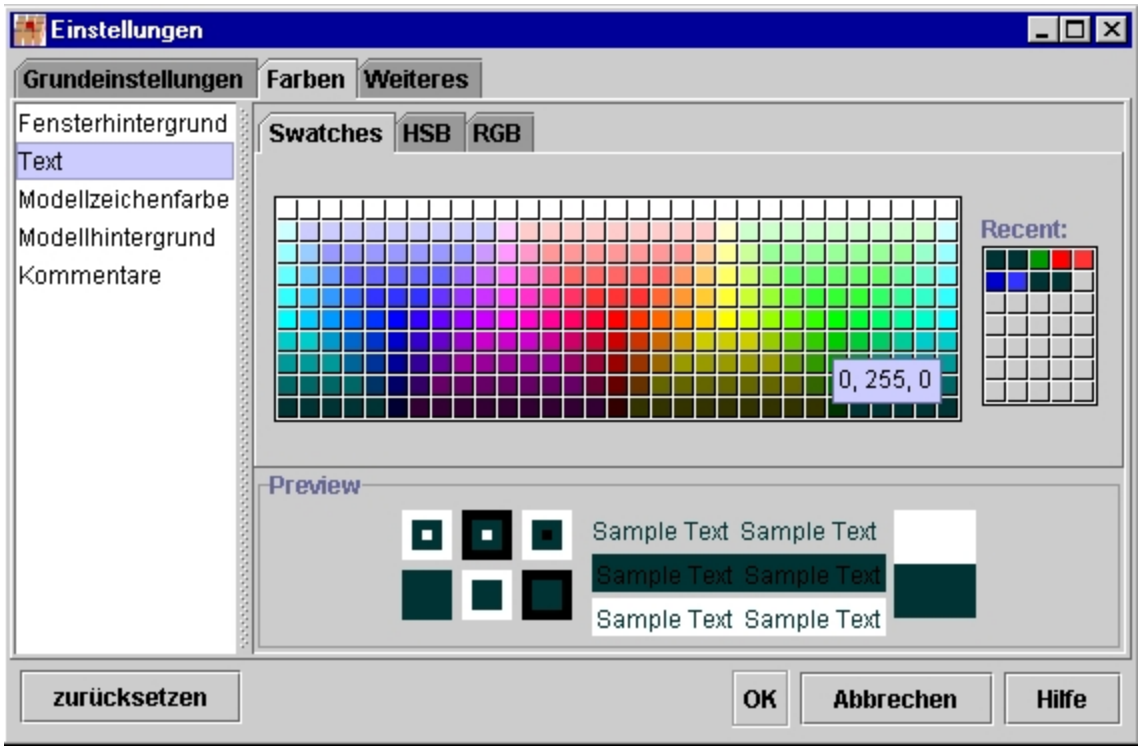


Abbildung 5 Dialog einer einfachen Möglichkeit der Individualisierung im Prototyp

Im dem hier gezeigten Beispiel kann der Benutzer bestimmte Farben einstellen. Diese einfache Form der Individualisierung ist heute in fast jedem Softwaresystem verfügbar. Für den Farbdialog wurde eine Standardkomponente benutzt. Die hier englischen Beschreibungen müssen noch durch Deutsche ausgetauscht werden.

4 Style Guide

Style Guides sind Dokumente, die den Entwicklungsprozess für ergonomische Software mittels Vorgaben, Empfehlungen und Hinweisen unterstützen. Sie beschreiben die Prinzipien und Regeln, die der Benutzungsschnittstelle zugrunde liegen. Synonym dazu wird auch der Begriff Design Guideline genutzt. Style Guides ersetzen jedoch keine Design-Spezifikation (vgl. [GöMa00] S. 43), da sie keine Erklärungen über den logisch konsistenten Bedienungsablauf des Dialoges geben. Im benutzerorientierten Entwicklungsprozess unterstützen sie den dritten Arbeitsschritt (vgl. Anhang A.8). Dieser besteht im Entwurf der Gestaltungslösungen. Die Style Guides kommen dabei vor allem bei späteren Iterationsschritten, die sich mit der Feingestaltung des Prototyp beschäftigen, zum Einsatz.
Style Guides verstehen sich als Leitfäden, die lediglich empfehlenden Charakter besitzen. Im Gegensatz dazu sind **Standards** Dokumente, die direkte Anweisungen für die Gestaltung von Benutzungsschnittstellen geben. Diese müssen unbedingt befolgt werden. Standard sind sehr allgemein formuliert, was die Überprüfung ihrer Einhaltung erschwert (vgl. [HiHa93] S. 18).
Style Guides sind im allgemeinen in zwei Bereiche gegliedert. Der erste befasst sich mit der Beschreibung der Bildschirmelemente. Dazu gehört sowohl das Aussehen (engl. look), als auch das Verhalten des Objektes (engl. feel). Der zweite Bereich gibt Empfehlungen, in welchen Anwendungsfällen und in welcher Art und Weise die entsprechenden Bildschirmelemente zu nutzen sind. Fasst man die beiden Teile des ersten Bereiches zusammen, so entsteht der Begriff **Look and Feel**, mit dem die Charakteristik einer Benutzungsschnittstelle beschrieben wird.
Style Guides unterscheiden sich in der Konkretheit ihrer Gestaltungskriterien. So ist die Richtlinie ISO 9241 durchaus als Style Guide anzusehen. Aufgrund ihrer absoluten Allgemeingültigkeit ist sie als alleinige Grundlage für die Gestaltung der Oberfläche einer Anwendung nicht geeignet. Sie gibt vor allem Auskunft darüber, was kein ergonomisches Design ist, kann demzufolge auch zur Evaluierung einer Oberfläche eingesetzt werden. Konkreter als die ISO 9241 sind die Style Guides der verschiedenen Betriebssysteme. Diese fußen auf der offiziellen Norm, sind aber bereits detaillierter, da betriebssystemabhängige Techniken und Paradigmen bekannt sind. So ist beispielsweise im MS Windows Style Guide festgelegt, dass rechts oben im Fenster ein kleines *x* das Schließen dieses Fensters ermöglicht. In der ISO 9241 wird lediglich gefordert, dass ein Fenster jederzeit geschlossen werden kann.

Der Java Look and Feel Design Guideline hat sich zum Ziel gesetzt, plattformunabhängigen Anwendungen wie sie mit Java entwickelt werden können, ein konsistentes Aussehen und Verhalten zu verleihen (vgl. [Java01]). Er wird mittels der Java Foundation Classes (JFC) umgesetzt. Die JFC sind eine Sammlung von standardisierten Dialogelementen. Dieses Toolkit wurde auch im Prototyp verwendet. Trotz des einheitlichen Look and Feel werden plattformabhängige Grundkonzepte nicht durchkreuzt. So bleibt das Aussehen der Fenster oder standardisierte Tastenbelegungen nach wie vor betriebssystemabhängig.

HIX und HARTSON fassen die bis jetzt erwähnten Style Guides unter dem Begriff **Commercial Style Guides** zusammen. Diese werden von einzelnen Organisationen entwickelt und publiziert. Sie sind im allgemeinen mit einem kommerziellem Toolkit verbunden (vgl. [HiHa93] S. 22). Obwohl diese Style Guides schon deutlich konkreter sind, reichen sie in der Regel nicht aus, um den Entwicklungsprozess optimal zu unterstützen. Sie bilden lediglich die Grundlage, auf der ein anwendungsabhängiger Style Guide entwickelt werden muss. In diesem werden für ein bestimmtes Projekt spezifische Gestaltungshinweise definiert, da erst in diesem Fall ist der genaue Nutzungskontext bekannt. So kann auf anwendungsspezifische Merkmale wie beispielsweise die Zielgruppe entsprechend Rücksicht genommen werden. Damit soll die Anwendung einen größtmöglichen Grad an Ergonomie erhalten.

Durch die Nutzung der verschiedenen Style Guide entsteht eine mehrstufige Filterung der Gestaltungsmöglichkeiten für den Entwickler. Diese ist in der folgenden Grafik abgebildet.

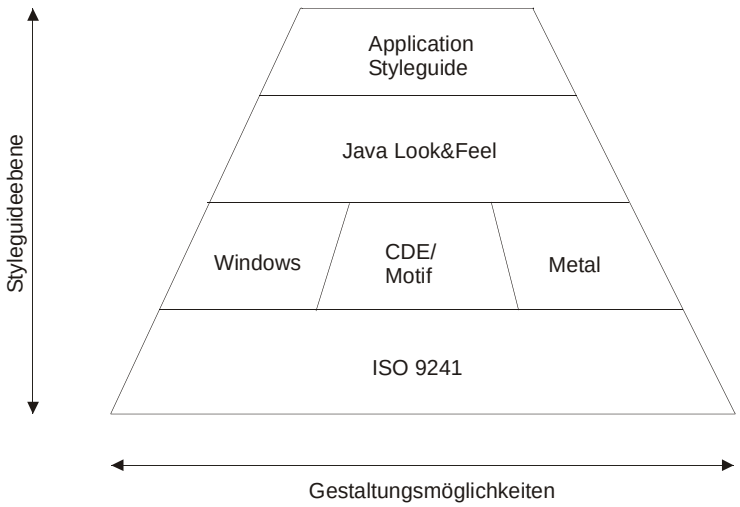


Abbildung 6 Mehrstufige Filterung der verschiedenen Style Guides

Style Guides sind ein wichtiges Hilfsmittel für die Entwicklung von Oberflächen. Sie können jedoch nie alle Fragen des Designs beantworten. Ein Nachteil von Style Guides besteht darin, dass sie zwar sehr gut beschreiben, was für Interaktionselemente genutzt werden können. Sie geben jedoch wenig Aufschluss, in *welchem* Fall diese angewendet werden müssen. Außerdem besteht die Gefahr, dass sie bei zu konkreter Ausprägung die Kreativität des Entwicklers einschränken. Die Nutzung von Style Guides befreit den Oberflächendesigner nicht von der Pflicht, eine ausführliche Evaluation der Benutzungsschnittstelle durchzuführen.

4.1 Entwicklung des Style Guide

Die Entwicklung eines Application Style Guide findet in einer frühen Phase des Projektes statt. Er sollte auf Grundlage von bereits vorhandenen Style Guides und nach Analyse des Nutzungskontextes unter Anleitung eines Experten entstehen. Während der Projektentwicklung muss der anfangs unvollständige Application Style Guide stetig erweitert und aktualisiert werden. HIX und HARTSON schlagen eine Gliederung in fünf Bereiche vor (vgl. [HiHa93] S. 25). In der *Einleitung* werden grundsätzliche Paradigmen erklärt. Hier wird beispielsweise festgelegt, ob eine grafische oder textuell orientierte Oberfläche entwickelt werden soll. Der zweite Bereich beschäftigt sich mit den *Ein- und Ausgabegeräten*. Der nächste Teil des Style Guide zeigt *repräsentative Bildschirmdarstellungen*. Diese dienen als Vorlage bzw. Muster. Sie sind vor allem für die Sicherstellung des konsistenten Aussehens von Bedeutung. Der vierte Bereich beschreibt alle *Interaktionselemente*. Es wird zu jedem einzelnen sowohl Aussehen als auch Verhalten erklärt. Falls nötig, muss beschrieben werden, in welchem Fall das Element benutzt wird. Der letzte Teil des Application Style Guide beinhaltet alle auftretenden *Meldungen*. Dazu gehören sowohl Fehler- und Warnmeldungen als auch andere Rückinformationen.

Der hier entwickelte Application Style Guide umfasst hauptsächlich den vierten Teil der eben beschriebenen Gliederung. Die im Anhang A.4 bis A.7 gezeigten Abbildungen gehören zu dem dritten Bereich. Die drei fehlenden Bereiche sollten noch ergänzt werden. Der im folgenden näher beschriebene Style Guide baut auf dem Java Look and Feel Design Guideline (vgl. [Java01]) auf. Er umfasst lediglich die Erweiterungen und anwendungsabhängigen Spezifikationen, die getroffen wurden, um eine konsistente Anwendung entwickeln zu können. Hauptsächlich werden Empfehlungen für Layout, Bezeichnung, Farben u.ä. für alle visuellen Klassen gegeben. Weicht das Verhalten vom Java Look and Feel Design Guideline ab, wird es ebenfalls beschrieben. Das Ziel ist eine Reduzierung der Möglichkeiten für den

Designer, ohne ihn jedoch beim kreativen Entwurf einzuschränken. Damit soll die Entwicklung einer konsistenten Anwendung sichergestellt werden. Zusätzlich zu den hier ausgearbeiteten Richtlinien sollten die im Abschnitt 3.4 beschriebenen Entwurfstechniken beachtet werden. Diese unterstützen vor allem bei der Entwicklung der inneren Struktur und Organisation der Anwendung. Es existieren noch eine Vielzahl von Zusammenstellungen einzelner Regeln, die zu einer ergonomischen Oberfläche beitragen können. Eine bekannte Sammlung sind beispielsweise die acht goldenen Regeln von Shneiderman (vgl. [Shne98] S. 74f).

4.2 Beschreibung der visuellen Klassen

Im folgenden Abschnitt werden die visuellen Klassen näher spezifiziert. Im ersten Teil wird jeweils kurz erläutert, in welchem Anwendungsfall das Bildschirmelement benutzt werden kann. Die Einschränkungen und Empfehlungen für Aussehen und Verhalten werden im zweiten Teil diskutiert.

4.2.1 GMEJFrame

Diese Klasse bildet die Grundlage aller Fenster, die im Prototyp dargestellt werden. Sie ist von **`JFrame`** abgeleitet und besitzt mehrere zusätzliche Methoden. Diese sollen sowohl die Arbeit für den Entwickler vereinfachen, als auch in ergonomischer Sicht Vorteile bringen. Es wurde die Methode ***`setSizeAndPosition`***() entwickelt (vgl. Anhang **0**). Diese dient dazu, sowohl Position und Größe der letzten Darstellung wieder herzustellen. Sie definiert ebenfalls, welche Buttons für die Standardfunktionen OK, Hilfe und Abbruch zu benutzen sind. Das ist für die Klasse von Bedeutung, da sie die Funktionstasten ENTER, F1 und ESC mit Ausführung dieser Funktionen belegt. Damit wird eine einheitliche Steuerung der Fenster gewährleistet (vgl. Abschnitt **3.4.1**).

Es ist mindestens ein Panel in das Fenster einzubringen. Das erste Panel ist dabei gleichbedeutend mit dem Attribut *`contentPane`*. Die Evaluierung eines Fensters kann mit Hilfe der Checkliste in Anhang B.1 erfolgen.

4.2.2 JPanel

Die Klasse **`JPanel`** dient zur Gliederung bzw. Organisation eines Fensters. Sie stellt einen Arbeitsbereich innerhalb eines Fensters dar. Dieser Fensterbereich wird im englischen mit *panel* bezeichnet. Aufgrund einer fehlenden adäquaten Übersetzung wird im folgenden der englische Begriff genutzt. Panele können verwendet werden, um semantisch

zusammengehörige Bildschirmelemente auch als Gruppe erscheinen zu lassen. Damit verringert sich die Suchzeit für den Benutzer (vgl. [Krus99] S. 40). Zusätzlich kann durch die Verwendung von Panelen ein konsistentes Aussehen innerhalb der verschiedenen Fenster einer Anwendung sichergestellt werden. Panele besitzen drei funktionale Merkmale:

- Sie können Bildschirmelemente aufnehmen, man spricht deshalb auch von einer Containerklasse. Dabei kann ein Panel wieder andere Panele aufnehmen, was zu einer Anordnungshierarchie führt.
- Sie besitzen ein Layout, d.h. eine Vorschrift, wie die inneren Elemente angeordnet werden (vgl. Abschnitt 4.2.9).
- Sie können einen Rand annehmen. In Ausnahmefällen kann auf die Nutzung eines Randes verzichtet werden. Um die einheitliche Benutzung von Rändern zu ermöglichen, wurde die Klasse **`GMEBorder`** entwickelt (vgl. 4.2.8).

Für die Verwendung von Rändern wurden eindeutige Regeln festgelegt. Die Auswahl des zu benutzenden Randes hängt dabei sowohl von der Anordnung des Panels als auch von der inhaltlichen Beziehung zu anderen Panelen ab. So besitzt beispielsweise das erste Panel jedes Fensters den Randtyp null. Die derzeit definierten Randtypen wurden im Anhang abgebildet.
Folgende Regelungen werden für die Verwendung der Ränder vorgegeben:

- Das jeweils erste Panel in einem **`GMEJFrame`**, **`JTabbedPane`** oder einem **`JSplitPane`** hat den Randtyp null.
- Berühren sich zwei Panele, ist zwischen diesen Beiden ein Abstand von insgesamt 6 Pixel zu einzuhalten, indem in beiden Panelen ein Abstand von 3 eingetragen wird. Bei den vier Außenseiten eines Panels ergeben sich damit maximal 15 Kombinationen die den Randtypen null bis 15 entsprechen.
- Ist der Inhalt des Panels deutlich vom Rest des Formulars zu unterscheiden, kann eine Umrandung mit einer Überschrift erfolgen. Diese entspricht dem Randtyp zwei.

4.2.3 JScrollPane

Die Klasse **`JScrollPane`** generiert bei Bedarf automatisch Bildschirmlaufleisten (engl. scroll bar), die bei in der Größe anpaßbaren Bildschirmelementen benötigt werden, um diese komplett darzustellen. Üblicherweise wird diese Klasse mit einem Fenster verknüpft. Wird das Fenster verkleinert, ermöglichen die Laufleisten das Verschieben des sichtbaren Bereiches. Die Klassen **`JTree`**, **`JTable`** und **`JEditorPane`** sollten ebenfalls grundsätzlich mit einem **`JScrollPane`** verknüpft sein. Diese Klassen verändern während

der Benutzung die Größe des Bereiches, in dem sie ihre Informationen darstellen. Die Größe des Bildschirmelementes bleibt davon jedoch unberührt. Damit kann beim Entwurf keine Aussage getroffen werden, ob die vorgegebene Größe während der späteren Benutzung ausreichend ist. Für den Rand gelten die gleichen Regelungen wie für ein **`JPanel`**.

4.2.4 JTabbedPane

Die Klasse **`JTabbedPane`** ist immer dann zu verwenden, wenn die Informationsmenge eines Fensters zu hoch ist, um auf einmal dargestellt zu werden. Eine Trennung wird durch die Darstellung von Karteikarten, die durch die Klasse gekapselt werden, erreicht. Durch das Anklicken auf einen Reiter kann zwischen ihnen gewechselt werden. Die Reiter sollten immer oberhalb der Karteikarten angezeigt werden. Es wird grundsätzlich nur eine Karteikarte angezeigt. Aus diesem Grund sollten inhaltlich zusammengehörige Elemente auf dem selben Panel angeordnet sein. Es empfiehlt sich diese Klasse in ein mit einem **`JScrollPane`** zu verknüpfen, da der Inhalt der unterschiedlichen Karteikarten auch verschiedene Dimensionen besitzen kann. Das erste Element einer Karteikarte ist, abgesehen von Ausnahmen, ein **`JPanel`**. Für dieses gelten die Anweisungen aus Abschnitt 4.2.2.

4.2.5 JSplitPane

Die Klasse **`JSplitPane`** ist einzusetzen, wenn eine nicht unbedeutende Menge an Informationen abhängig von einer vom Benutzer veränderbaren Variable anzuzeigen ist. Die Klasse teilt den Bildschirmausschnitt in zwei unterschiedlich große Teile. Üblicherweise wird die vom Benutzer zu verändernde Variable in Form einer Liste angezeigt. Dafür eignen sich im besonderen die Klassen **`JList`** und **`JTree`**. Sie sind im linken Teil des **`JSplitPane`** anzuordnen. Die rechte Seite enthält die davon abhängigen Informationen. Die Klasse **`JSplitPane`** wurde im Prototyp in mehreren Fenstern verwendet, um die Konsistenz bei der Gliederung zu erhöhen. Die Abbildung 5 illustriert die Anwendung der Klasse.

Beide Bereiche des **`JSplitPane`** sind grundsätzlich mit einem JPanel bzw. JScrollPane zu versehen. Die Eigenschaft *`dividersize`* ist mit 10 zu belegen. Sie gibt die Breite des Balken zwischen den beiden Bereichen an. Für das Attribut *`dividerposition`*, das der Breite des linken Teiles entspricht, ist eine geeignete Größe zu finden. Im Prototyp wurde üblicherweise ein Bereich zwischen 150 und 250 ausgewählt.

4.2.6 JToolBar

Ein **`JToolBar`** ist immer dann zu verwenden, wenn über Icons eine schnelle Auswahl von Funktionen ermöglicht werden soll. Dazu wird ein Container ähnlich einem Panel bereitgestellt, der die Buttons aufnimmt und anordnet. Man bezeichnet dieses Bildschirmelement auch als **Symbolleiste**. Zusätzlich wird die Funktionalität angeboten, die Leiste aus dem Fensters durch ziehen mit der Maus herauszulösen. Gesteuert wird diese Funktion über das Attribut *`floatable`*.

Der **`JToolBar`** sollte grundsätzlich oben in horizontaler Ausrichtung im Fenster platziert werden. Ist ein Menü implementiert, dann wird es direkt unter dem Menü angeordnet. Das Attribut *`floatable`* sollte auf *false* gestellt werden. Es ist im Einzelfall zu prüfen, ob ein Herauslösen des Symbolleiste aus dem Fenster sinnvoll ist. Werden mehrere **`JToolBar`** nebeneinander platziert, so ist der Randtyp 16 (vgl. Abschnitt 4.2.8) zu verwenden.

4.2.7 Border

Die Klasse **`Border`** wird verwendet, um Bildschirmelemente mit einem Rand bzw. Rahmen zu versehen. Dabei können Abstände zwischen einzelnen Elementen hergestellt werden, wodurch engere Zusammenhänge zwischen Informationen visualisiert werden. Durch Abstände zwischen einzelnen Teilen wird eine Gliederung des Fensters erreicht.

Grundsätzlich gibt es zwei verschiedene Ausprägungen dieser Klasse. In der ersten Form wird lediglich Platz um das zu umrahmende Bildschirmelement reserviert. Als Alternative gibt es die Möglichkeit zusätzlich eine Linie inklusive Überschrift um die Bildschirmelementgruppe zu platzieren. Die Farbe der Überschrift ist dabei auf schwarz zu setzen. Durch die Umrahmung wird eine noch größere Unabhängigkeit dieser Elemente von den restlichen im Fenster befindlichen Elementen vermittelt. Die Klasse **`Border`** sollte nur in der Klasse **`GMEBorder`** verwendet werden.

4.2.8 GMEBorder

Diese Klasse stellt alle Ausprägungen der Klasse **`Border`** für den Prototypen zur Verfügung. Damit wird eine einheitliche Verwendung von Rändern in allen Fenstern gewährleistet. Soll ein Rand mittels der Methode ***`setBorder`*** einem Bildschirmelement zugewiesen werden, dann ist als Parameter anzugeben:

`GMEBorder`.***`getBorder`***(**`GMEBorder`**.`<<` *`BORDERTYP`* `>>`).

Die zur Zeit definierten Randtypen sind in Anhang C.3.5 definiert. Diese können jederzeit erweitert werden.

4.2.9 Nutzung von Layoutmanagern

Der LayoutManager ist ein Interface, welches abgeleiteten Klassen ermöglichen soll, Position und Größe von Komponenten innerhalb eines Containers festzulegen. Ändert ein Container, beispielsweise ein Fenster, seine Größe, so positioniert der LayoutManager entsprechend den Vorgaben die Bildschirmelemente neu.

Der LayoutManager wurde in verschieden Klassen implementiert. So teilt **`BorderLayout`** beispielsweise den Bildschirm in die fünf Bereiche *Oben*, *Unten*, *Links*, *Recht* und *Mitte*. Die Klasse wurde in den meisten Fällen des Prototyp benutzt. Existieren in einem Fenster Bildschirmelementgruppen die sich auch bei Änderung der Fenstergröße gleich verhalten sollen, wird **`GridLayout`** benutzt. Um eine tabellarische Anordnung von Elementen zu erreichen kann die Klasse **`GridBagLayout`** genutzt werden. Bei dieser sind vorhandene Spalten mit einem Abstand von insgesamt 10 Pixeln zu versehen.

Wann welcher Layoutmanager zu verwenden ist, kann nicht konkret vorgegeben werden. Wichtig ist, dass der gesamte Bildschirminhalt auch bei Änderung der Fenstergröße sinnvoll positioniert bleibt. Zusätzlich sollten die allgemeinen Gestaltprinzipien unterstützt werden. Ob die richtigen LayoutManager eingesetzt wurden, muss bei jedem Fenster separat geprüft werden.

4.2.10 JMenu, JPopupMenu

Ein Menü stellt die Menge an verfügbaren Optionen des aktuellen Fensters dar. Ist in einem Fenster ein Menü implementiert, muss es alle Funktionen bereitstellen. Menüs wurden im Abschnitt 3.4.2.2 ausführlich behandelt. Sind nur wenige Funktionen verfügbar, muss kein Menü implementiert werden. Es reicht dann aus, ein Symbolleiste mit diesen Funktionen anzubieten.

Menüpunkte können mit einem Piktogramm versehen werden. Dieses muss das gleiche sein, dass an anderer Stelle bei gleicher Funktion verwendet wird. Soll kein Piktogramm angezeigt werden, dann muss ein Leeres angegeben werden, um eine einheitliche vertikale Ausrichtung der einzelnen Menüpunkte zu erreichen. Jeder Menüpunkt sollte mit einer Beschleunigungstaste versehen werden. Die einzelnen Menüpunkte sind nach logischen Kategorien zu organisieren. D.h., es sind logische Funktionsgruppen zu bilden. Die Reihenfolge innerhalb dieser sollte konsistent und nach Wichtigkeit geordnet sein. Ist eine Wichtung nicht möglich, kann auf eine konventionelle und danach auf die alphabetische Reihenfolge zurückgegriffen werden. Für häufig verwendete Menü-Optionen sollte eine Funktionstaste bereit gestellt werden. Konventionelle Belegungen, wie z.B. F1 für Hilfe,

sollten übernommen werden. Optionen, die momentan nicht verfügbar sind, sollten den Status *disabled* erhalten. Damit ist es dem Benutzer möglich, alle Optionen zu explorieren, und bei Nichtverfügbarkeit über das Hilfesystem zu erfahren, wie diese Option aktiviert und benutzt werden kann. Menü-Optionen sollten einen kurzen erklärenden Text am unteren Rand des Fensters einblenden, da eine Option mit einem oder zwei Worten meist nur unzureichend erklärt ist. Bei Ihrer Bezeichnung selbst ist darauf zu achten, dass der Benutzer möglichst genau weiß, was mit dieser Option erreicht werden kann.

Kontextmenüs, die bei allen Klassen für Mengen, wie z.B. **`JList`** und **`JTree`** empfohlen sind, enthalten stets eine Untermenge des aktiven, globalen Menüs. Sie zeigen für das aktuell ausgewählte Bildschirmelement eine Auswahl an Optionen und werden durch Klicken der rechten Maustaste aufgerufen. Funktionen, die nur im Kontextmenü erreichbar sind, sollten vermieden werden. Ansonsten gelten auch für Kontextmenüs die gleichen Regelungen wie für globale Menüs.

4.2.11 JLabel

Die Klasse **`JLabel`** dient zum Darstellen eines einfaches Textes. Sie wird im allgemeinen genutzt, um Textfelder, Listen u.ä. zu spezifizieren. Die Position des Label sollte oberhalb des dazugehörigen Bildschirmelements sein. Damit wird vermieden, dass bei Längenänderungen des Textes, die z.B. durch Übersetzen in andere Sprachen auftreten können, Teile des Label überdeckt werden. Da Java in einem solchen Fall am Ende des Textes drei Punkte anfügt, die nochmals Platz benötigen, kann u.U. der Sinn des Elementes für den Benutzer nicht mehr erkannt werden. Wird dennoch eine Anordnung links des Bildschirmelementes gewählt, ist durch den Einsatz eines entsprechenden LayoutManagers (vgl. Abschnitt 4.2.9) sicherzustellen, dass solche Nebeneffekte nicht auftreten. Der Schrift des Labels ist weder kursiv noch fett. Ihre Farbe ist schwarz.

4.2.12 JButton

Die Klasse **`JButton`** dient dazu, einen Button auf dem Bildschirm darzustellen. Mit seiner Hilfe kann eine Funktion direkt ausgelöst werden. Buttons werden im Prototyp in drei verschiedenen Ausprägungen genutzt, die jeweils abhängig von der Position und der Semantik sind. Wird ein Button in einem **`JToolBar`** platziert, so wird er als Icon benutzt. Er hat dann die feste Größe von 27x27 Pixel. Der Button besteht lediglich aus einem Piktogramm, ein Text wird nicht angegeben.

Wird der Button unterhalb des Arbeitsbereiches des Fensters platziert, dient er als Zugriff auf die Sekundärfunktionen des aktuellen Fensters. Im allgemeinen sind das die Funktionen Bestätigen, Abbrechen, Hilfe und Zurücksetzen. Die Buttons haben in diesem Fall eine variable Breite und die feste Höhe von 27 Pixel. Ein Piktogramm ist im Gegensatz zum Text keine Pflicht. Das Standardlayout sieht vor, die Buttons je nach Vorhandensein rechtsbündig in der Reihenfolge Hilfe, Abbrechen und Bestätigen anzuordnen (vgl. Abbildung 7). Zusätzliche Button wie Zurücksetzen oder ähnliches werden linksbündig angeordnet. Der Standardbutton, welcher mit der Taste ENTER ausgelöst wird, bekommt zur Visualisierung automatisch den Randtyp 15 (vgl. Abschnitt 4.2.8) von der Klasse **`GMEJFrame`** zugewiesen.

Abbildung 7 Darstellung des Layout der Button in einem Fenster

Der Text der einzelnen Button muss je nach Anwendungsfall angepasst werden. Es empfiehlt sich die jeweilig ausgelöste Funktion zu beschreiben. Allgemeine Beschreibungen wie „OK“ sollten möglichst vermieden werden.
Werden Buttons im Arbeitsbereich des Fenster angeordnet, so ist ihre Höhe auf 21 Pixel festzulegen. Ist ein Text angegeben, so ist die Breite variabel. Wird nur ein Piktogramm angezeigt, ist die Breite ebenfalls auf 21 Pixel zu setzen.
Eine Abwandlung der Klasse **`JButton`** ist die Klasse **`JToggleButton`**. Dieser Button hat die Möglichkeit, Ein/Aus Zustände zu visualisieren. Der Schriftschnitt des beschreibenden Textes des Button ist grundsätzlich Fett, die Farbe ist schwarz.

4.2.13 JTextField

Die Klasse **`JTextField`** ermöglicht die Eingabe von Daten. Grundsätzlich ist immer abzuwägen, ob für den jeweiligen Anwendungsfall das Textfeld ideal ist. Ist die Eingabe von begrenzter Länge, der Inhalt jedoch frei wählbar, dann sollte dieses Element genutzt werden. In anderen Fällen können auch Kombinationsfelder (**`JCombobox`**), Listen (**`JList`**) oder mehrzeilige Texteingabe (**`JTextArea`**) die bessere Wahl sein. Diese Klasse **`JTextField`** kann auch genutzt werden, Informationen anzuzeigen. Dabei ist lediglich die Eigenschaft *`editable`* auf *false* zu setzen. Der Vorteil gegenüber der Anzeige mit einem **`JLabel`**

besteht darin, dass der Benutzer die Information markieren, kopieren und anderweitig verwenden kann.

4.2.14 JDateField

Die Klasse **`JDateField`** (vgl. Anhang C.1.3) wird verwendet, wenn ein Datum ein- oder ausgegeben werden muss. Sie wurde von **`JTextField`** abgeleitet, um eine bessere Unterstützung des speziellen Formates zu erreichen. Aus diesem Grund gelten für die Klasse die gleichen Bedingungen wie für die Klasse **`JTextField`**.

Werden weitere Eingaben eines bestimmten Datentyps öfters benötigt, sollte auch für diese eine Klasse abgeleitet werden. Das erleichtert sowohl die Implementation, als auch die spätere Benutzung durch eine bessere Fehlerbehandlung. Zusätzlich wird sichergestellt, dass spezielle Datenformate immer in der gleichen Form verwendet werden.

4.2.15 JTextArea, JEditorPane

Sowohl die Klasse **`JTextArea`** als auch die Klassen **`JEditorPane`** werden nicht verwendet. An Ihrer Stelle wird grundsätzlich **`JTextPane`** genutzt.

4.2.16 JTextPane

Die Klasse **`JTextPane`** ermöglicht die Anzeige und Eingabe von mehrzelligem Text. Dabei wird ein *plug-in Editor Kit* benutzt, das an den anzuzeigenden Text angepasst werden kann. Bereits vorhanden sind *RTF, HTM Styled Text* und *Default Editor Kit*. Für die Anzeige von HTML-Seiten ist das **`JTextPane`** ausreichend. Für die Eingabe von normalen Text, wie er beispielsweise für die Dokumentation gebraucht wird, muss mindestens das *RTF Editor Kit* implementiert werden. Die Funktionen für Schriftartwechsel u.a. müssen mittels einer Symbolleiste oder einem Menü dem Benutzer zugänglich gemacht werden. Wird dem Bildschirmelement **`JTextPane`** kein **`JScrollPane`** zugewiesen, so muss ihm grundsätzlich der Randtyp 18 (vgl. Abschnitt 4.2.8) zugewiesen werden.

4.2.17 JCombobox

Ein Kombinationsfeld ermöglicht die Auswahl aus einer Liste von Objekten. Es ist keine Mehrfachauswahl möglich. Falls erforderlich kann die Eingabe einer neuen Option ermöglicht werden. Kombinationsfelder bieten sich ideal an, um beispielsweise Statusanzeigen zu visualisieren. Die Hintergrundfarbe ist auf weiß zu setzen. Die Höhe ist wie bei den übrigen einzeiligen Elementen auf 21 Pixel festzulegen.

4.2.18 JList

Eine Liste visualisiert ein Menge an Objekten, welche normalerweise in Textform dargestellt sind. Die Objekte können jedoch auch grafisch angezeigt werden. Listen ermöglichen eine Mehrfachauswahl. Es ist immer zu prüfen, ob für die dargestellten Objekte ein Kontextmenü angeboten werden soll. Die Klasse **`JList`** muss grundsätzlich mit einem **`JScrollPane`** verknüpft sein, um ein Scrollen zu ermöglichen. Die Bezeichnung von Listen wird oberhalb angeordnet. Werden Elemente eingefügt, muss der neue Eintrag anschließend selektiert sein. Wird ein Eintrag gelöscht, sollte der nächste gültige Eintrag selektiert sein. Bei Listen, die einen größeren Umfang erreichen können, sollte eine Sortier- sowie eine Suchfunktion angeboten werden. Wird dem Bildschirmelement **`JList`** kein **`JScrollPane`** zugewiesen, so muss ihm grundsätzlich der Randtyp 18 (vgl. Abschnitt 4.2.8) zugewiesen werden.

4.2.19 JTree

Auch die Klasse **`JTree`** dient zum Anzeigen einer Menge von Objekten. Im Unterschied zu einer einfachen Liste, stehen hier die Objekte in einer hierarchischen Ordnung. Zusätzlich ist meist der Typ von Objekten einer Ebene gleichartig, während er sich in untergeordneten Hierarchien ändern kann. Ist dies der Fall, sollte vor der textuellen Beschreibung ein Piktogramm zur Identifikation dargestellt werden. Das folgende Beispiel zeigt drei Ebenen, wobei die Objekte jeder Ebene einen anderen Typ besitzen.

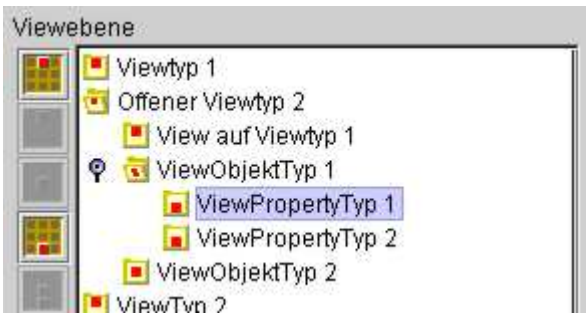


Abbildung 8 Baumdarstellung von verschiedenartigen Objekten

Besitzt das angezeigte Objekt untergeordnete Elemente, so sollte das durch ein zusätzliches Piktogramm angezeigt werden. Auch bei diesem Bildschirmelement wird die Bezeichnung oberhalb angeordnet. Beim Einfügen oder Löschen von Einträgen verhält sich die Klasse **`JTree`** wie **`JList`**. Wird ein Element untergeordneter Ebene eingefügt, muss der Pfad zu diesem Element ebenfalls mit geöffnet werden. Auch der Klasse **`JTree`** sollte ein

JScrollPane zugewiesen werden. Ansonsten ist dem Bildschirmelement der Randtyp 18 (vgl. Abschnitt 4.2.8) zuzuweisen.

4.2.20 JTable

Tabellen stellen zweidimensional organisierte Informationsmengen dar. Eine explizite Bezeichnung durch einen Label ist nicht unbedingt nötig, da die Spaltenüberschriften meist ausreichend Aufschluss über den Inhalt der Tabelle liefern. Üblicherweise wird in Zellen Text mittels der Klasse **JLabel** dargestellt. Hier sollte der Schriftschnitt von fett auf normal zurückgesetzt werden. Werden andere Klassen innerhalb der Tabelle abgebildet, so sind die in diesem Style Guide aufgestellten Regeln zu beachten. Der Benutzer sollte die Möglichkeit haben, Tabellen nach ausgewählten Spalten zu sortieren. Diese sollten auf Verlangen auszublenden und untereinander auszutauschen sein. Die vom Benutzer vorgenommen Änderungen in der Darstellung der Tabelle sollten gespeichert werden, um beim nächsten Programmstart die gleiche Anzeige wieder herstellen zu können.

4.2.21 JCheckBox

Die Klasse **JCheckBox** wird genutzt, um zwischen zwei möglichen Zuständen auszuwählen. Sie ist die vereinfachte Form von zwei voneinander abhängigen **JRadioButton** und besitzt die gleiche Funktionalität wie ein **JToggleButton**. Der Vorteil diesem gegenüber ist die Möglichkeit, eine längere Beschreibung mit darzustellen. Diese sollte einen normalen Schriftschnitt und die Farbe schwarz besitzen.

4.2.22 JRadioButton

Die Klasse **JRadioButton** wird benutzt, um eine endliche Anzahl von sich ausschließenden Optionen anzuzeigen und gleichzeitig einen schnellen Zugriff zu gewähren. Müssen mehr als fünf Optionen visualisiert werden, sollte eine nicht editierbare Klasse **JList** bzw. **JCombobox** vorgezogen werden. Die Klasse **JRadioButton** muss grundsätzlich einer Instanz von **ButtonGroup** zugeordnet werden. Die Schriftart der Bezeichnungen ist weder kursiv noch fett. Ihre Farbe ist schwarz.

4.2.23 Tooltips

Tooltips stellen Hinweise dar, die angezeigt werden, wenn der Benutzer eine kurze Zeit mit dem Mauszeiger auf einem Bildschirmelement verweilt. Es sollte für jedes sichtbare Element ein entsprechender Text vorgesehen sein. Dabei sollte das Bildschirmelement inhaltlich beschrieben werden. Wie bei den beschreibende Texten der Klasse **JLabel** ist auf

allgemeine Floskeln zu verzichten. Die Beschreibung: „Bitte Text eingeben“ gibt wenig Aufschluss, welche Art von Text eingegeben werden soll.

4.3 Icons

Icons ermöglichen die grafische Darstellung von Optionen oder Objekten. Für Icons gibt es zwei Standardgrößen. Die Größe 16 x 16 Pixel wird für Bäume, Fenster, Listen u.ä. genutzt. Für Button wird die Größe 24 x 24 Pixel empfohlen. Die Entwicklung einer Icon-Familie wird ausführlich im Abschnitt 3.4.4 diskutiert.

5 Evaluation ergonomischer Oberflächen

Die Evaluierung von interaktiven Systemen dient dazu, die Benutzbarkeit einer Schnittstelle zu testen und zu verbessern (vgl. [Prei99] S. 232). Evaluation im Sinne von Qualitätssicherung und Qualitätsverbesserung stellt einen wichtigen Aspekt bei der Gestaltung ergonomische Software dar. Mangelhafte Oberflächen können nicht nur zu Einbußen bei der Effizienz der Aufgabenerledigung, sondern auch zu Fehlern führen. Diese können abhängig vom Aufgabenbereich des Anwendungssystems erhebliche Konsequenzen nach sich ziehen. Aus diesem Grund muss das Ausmaß der Evaluation immer dem Softwareprojekt angepasst werden. Studien belegen, dass schon mit relativ geringen Einsatz bis zu 80 Prozent des Verbesserungspotential erkannt und umgesetzt werden können (vgl. [ReMo95] S. 248). Soll ein höherer Prozentsatz erreicht werden, steigen die Kosten überproportional.

Die ISO 14598 beschreibt den Evaluierungsprozeß für Software. Da die Norm dafür gedacht ist, ein gesamtes Softwareprojekt zu beurteilen, kann sie auch bei Benutzungsschnittstellen eingesetzt werden. Grundsätzlich werden drei verschiedene Ausgangssituationen betrachtet. So kann der Evaluierungsprozeß entweder im Rahmen der Entwicklung oder Anschaffung von Software oder unabhängig von beidem durchgeführt werden (vgl. [ISO14598-1] S. 6). Die entsprechenden Anpassungen des Prozesses werden in den Teilen drei bis fünf der Norm speziell behandelt. Den folgenden Ausführungen liegt der im Teil eins spezifizierte allgemeine Evaluierungsprozeß zu Grunde. Dieser wird in vier Phasen gegliedert (vgl. Anhang A.10). Im ersten Schritt werden die Anforderungen festgelegt. Dabei wird der Zweck der Evaluierung definiert. Handelt es sich beispielsweise um ein Endprodukt, so entscheidet die Evaluierung zwischen Auslieferung oder Überarbeitung der Software. Zu den Anforderungen gehört ebenfalls die Spezifikation eines quality model. Ein **Qualitätsmodell** (engl. quality model) besteht aus einer Menge von Eigenschaften einer Software und deren Beziehungen untereinander (vgl. [ISO9126-1] S. 21). Die ISO 9126-1 beschäftigt sich umfassend mit Qualitätsmodellen. Das dort vorgestellte Modell quality in use kann als Ausgangspunkt für die Evaluierung einer Benutzungsschnittstelle gesehen werden (vgl. Anhang A.12). Nach der Definition der Anforderungen beginnt der zweite Schritt des Evaluierungsprozesses. Da die vier beispielhaft genannten Eigenschaften Leistungsfähigkeit, Produktivität, Sicherheit und Zufriedenheit nicht direkt messbar sind, müssen dafür Indikatoren gefunden werden. ZEIDLER und ZELLNER sehen in der Definition dieser scharfen,

softwareergonomischen Kriterien ein Hauptproblem der Evaluierung (vgl. [ZeZe92], S.: 189). Wurden geeignete Kriterien gefunden, kann die Wichtung der einzelnen Merkmale sowie die Skala für die Messung festgelegt werden. Anhang A.11 zeigt einen Vorschlag der ISO 14598-1 für eine mögliche Einteilung. Der dritte Prozessschritt umfasst die Entwicklung eines Evaluierungsplanes. Dazu gehört die Auswahl der Testpersonen, die Festlegung des Verantwortlichen, die Art der Datenerfassung und -auswertung. Es können beispielsweise Daten durch Fragebögen, Interviews oder Videoaufnahmen gesammelt werden. In dieser Phase werden auch die maximalen Kosten festgelegt. Im abschließenden vierten Schritt findet die eigentliche Evaluierung statt. Hier werden die einzelnen Kriterien in den Skalen erfasst, mit den Vorgabewerten verglichen und Auswertungen durchgeführt. Wurden Probleme bei der Gebrauchstauglichkeit festgestellt, werden diese anschließend analysiert. In den meisten Fällen existieren mehrere Lösungsansätze. Diese müssen diskutiert und entsprechend umgesetzt werden. Dabei sind die Kosten für die Realisierung zu beachten. Vor allem die Erfassung der Daten wird in vielen Publikationen ausführlich beschrieben. Eine erfolgreiche Evaluierung ist jedoch ohne sorgfältige Vorbereitung kaum durchzuführen. Aus diesem Grund sollten alle Schritte des Evaluierungsprozesses durchlaufen werden.

Grundsätzlich kann die Evaluierung sowohl durch externe MCI-Experten, das Designerteam oder durch Anwender erfolgen. Experten sind mit vielen offiziellen Style Guides vertraut. Sie besitzen umfangreiche Erfahrungen mit Benutzungsschnittstellen und können viele der vorhanden Probleme an einem Prototypen schnell erkennen. Ein Nachteil dieser Evaluationsmethode besteht darin, dass Experten die praktischen Probleme bei der Aufgabenerfüllung nicht kennen (vgl. [ReMo95] S. 252). Dieses Problem besteht in abgeschwächter Form auch bei der Evaluierung durch das Designerteam. Allerdings können Designer den aktuellen Entwicklungsstand kontinuierlich mit den definierten Style Guides und allgemeinen Gestaltungsprinzipien (vgl. Abschnitt 3.3) abgleichen. Auch die in dieser Arbeit entwickelten Checklisten (vgl. Anhang B) nutzen Sie, um Arbeitsschritte sofort zu prüfen. Beide Formen der Begutachtung, bei der methodisches Vorgehen zur Identifikation von Problemen führt, werden mit dem Begriff **heuristische Evaluation** bezeichnet (vgl. [Prei99] S. 242). NIELSEN definiert sie als systematische Prüfung der Benutzungsschnittstelle auf Gebrauchstauglichkeit (vgl. [Niel93] S. 155f). Dabei zeigt er anhand von Studien, dass bereits fünf Testpersonen ca. 75 Prozent der Probleme erkannt haben.

Bei der **empirischen Evaluierung** wird der Test durch die Benutzer durchgeführt. Dabei unterscheidet man zwischen subjektiven und objektiven Evaluationsmethoden. Bei beiden Verfahren werden die Probanden mit dem Prototyp konfrontiert. Erfolgt die anschließende Bewertung durch den Benutzer persönlich, so nennt man das Ergebnis subjektiv. Wird die Bewertung anhand von Prüflisten durchgeführt, ergibt sich eine objektive Evaluierung. Inhalt dieser Prüflisten sind beispielsweise Fehlerraten oder Lernaufwand.
Auch umfangreiche Maßnahmen der Evaluation geben keine Garantie für die Entwicklung einer ergonomische Benutzungsschnittstelle. Wichtig ist, dass schon während der Entwicklung flankierende Maßnahmen, wie der Einsatz von Werkzeugen oder die Abstellung eines Designerteams für die Entwicklung der Oberfläche eingeführt werden. Da alle Evaluationsmethoden zumindest einen Prototypen benötigen, sollte auf dessen schnelle Entwicklung großen Wert gelegt werden. Vor allem Schichtenkonzepte, die die Oberfläche von der Anwendung selbst trennen, ermöglichen die parallele Entwicklung eines solchen. Es sei an dieser Stelle beispielhaft auf das 1983 entwickelte Seeheim-Modell hingewiesen.

5.1 Prototyping

Die Benutzung von Prototypen ist auch außerhalb der Informatik weit verbreitet. Aus diesem Grund gibt es verschiedenste Ansätze für eine Definition. Aus Sicht des Entwicklers von Benutzungsschnittstellen ist ein **Prototyp** die Simulation von Aussehen und Verhalten eines Softwaresystems (vgl. [HoHi97] S. 368). Damit kann bereits während der Entwicklung einer Anwendung ein konstruktiver Dialog zwischen Benutzern, Programmierern und Oberflächendesignern in Gang gesetzt werden. Dieser lenkt die grundsätzlich iterative Entwicklung von Prototypen. Studien ergaben, dass durch den engen Kontakt zu den Benutzern die entwickelten Oberflächen leichter zu erlernen und zu benutzen sind (vgl. [HiHa93] S. 258). Hinzu kommt, dass Anwender die Brauchbarkeit eines vorhandenes System besser bewerten können als Anforderungen an dieses zu beschreiben. Ein weiterer positiver Aspekt besteht darin, dass die Benutzer sich mit dem neuen System bereits vertraut machen können. Dadurch verkürzt sich die Lernphase nach der Produkteinführung. Durch die Einbeziehung der Benutzer steigt gleichzeitig die Akzeptanz gegenüber dem Anwendungssystem. Daraus folgt, dass Prototypen in einer kurzen Zeitspanne für Demonstrationszwecke einsetzbar sein müssen. Aus diesem Grund können sie nur einen begrenzten Umfang der Gesamtfunktionalität enthalten. Durch ständige Evaluierung durch Benutzer und Entwickler und anschließende Erweiterung und Anpassung des Prototypen wird

dieser nach und nach in das Endprodukt überführt. Um den Funktionsumfang einzugrenzen, können entweder horizontale oder vertikale Prototypen entwickelt werden. Diese können nur Teile des Gesamtsystems darstellen. Vor allem in frühen Projektphasen ist es oft nicht möglich, einen komplett integrierten Prototypen zu erstellen.

5.1.1 Horizontaler Prototyp

Bei dieser Art von Prototypen wird die Tiefe, die die Funktionalität des Systems widerspiegelt, ausgeblendet. Die Oberfläche ist im allgemeinen komplett dargestellt. Typische Beispiele sind komplette Fenster mit Menüeinträgen, Listen, Kombinationsfeldern und anderen Bildschirmelementen. Eine Dateneingabe ist genauso wenig möglich, wie die Auswahl einer Systemfunktion. Der Benutzer bekommt damit einen Gesamteindruck, wie die Anwendung später aussehen wird und welche Funktionen an welchen Stellen bereit gestellt werden. Damit kann beispielsweise die globale Metapher des Systems evaluiert werden. Eine Bewertung der Effizienz ist nicht möglich, da keine Aufgaben durchgespielt werden können. Es lassen sich lediglich begrenzte Aussagen über die Effektivität machen. Horizontales Prototyping wird meist für sehr zeitig entwickelte Prototypen eingesetzt.

5.1.2 Vertikaler Prototyp

Bei dem vertikalen Prototypen wird die Funktionalität eines kleinen Teilbereiches komplett realisiert. Damit kann der Benutzer diese Teilaufgabe umfassend bearbeiten. Sowohl Dateneingabe, -verarbeitung und -ausgabe sind möglich. Der Prototyp stellt insofern bereits eine fortgeschrittene, dem Endprodukt nahe Realisierung dar. Damit lässt sich eine konkrete Evaluierung der umgesetzten Funktionen vornehmen.

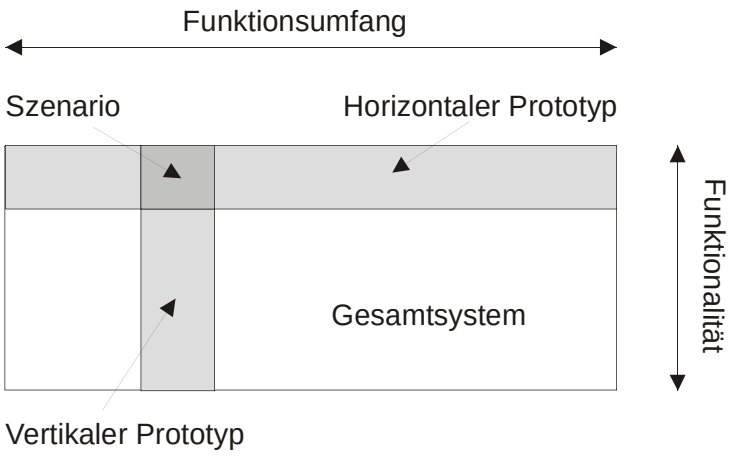


Abbildung 9 Dimensionen des Prototyping (vgl. [Niel93] S. 94)

5.1.3 Szenario

Werden beide Strategien gemischt eingesetzt, nennt man den so entstehenden Prototypen Szenario. In diesem ist eine minimale Funktionalität implementiert. Der Benutzer kann sowohl Oberflächendarstellung als auch Funktionalität bewerten. Es werden jedoch keinerlei Freiheiten im Sinne von alternativen Lösungsmöglichkeiten für den Benutzer angeboten. Das Szenario bestimmt die Umstände und Zeitabläufe der modellierten Aufgabe. Damit kann bereits in einer frühen Phase die Nutzung einer bestimmter Funktionen beschrieben und getestet werden, ohne einen kompletten Prototyp entwickeln zu müssen.

5.1.4 Entwicklung des Prototypen

Bei der Entwicklung von Prototypen steht das Hauptziel, schnell ein Ergebnis zu erzielen im Vordergrund. Dabei ist der Grad an Ausgereiftheit eines Prototypen nicht gleichzeitig der Grad an aussagekräftigen Ergebnissen (vgl. [Prei99] S. 236). In Studien wurde gezeigt, dass Benutzbarkeitsprobleme sowohl an einem realen System als auch an Papierskizzen gleichermaßen erkannt wurden. Ganz im Gegenteil laden unfertig aussehende Skizzen vielmehr dazu ein, über Varianten nachzudenken. Fertig scheinende Prototypen sind für eine kreative Diskussion nicht geeignet. Das legt nahe, am Anfang eines Projektes, durchaus mit Skizzen o.ä. zu arbeiten. In späteren Phasen folgt die Entwicklung eines Systems, um eine reale Demonstration zu ermöglichen. Dabei gibt es einige Punkte zu beachten, die eine Beschleunigung des Prototyping ermöglichen (vgl. [Prei99] S. 235). Speicher- oder Zeitoptimierung sind im allgemeinen zu vernachlässigen. Um eine effiziente Entwicklung zu ermöglichen, sollte ein leistungsfähiges Computersystem genutzt werden, dass durchaus deutlich besser als das spätere Zielsystem sein kann. Ein Prototyp kann auch mit fehlerhaftem Kode hilfreich für die Weiterentwicklung sein, da davon der Test nur wenig beeinflusst wird. Sonderfälle sollten im allgemeinen nicht im Prototyp realisiert werden, sondern erst im Endprodukt. Viel Zeit kann auch durch den Einsatz von Pseudodaten eingespart werden. Dabei wird die Komplexität der realen Daten gesenkt indem Datensätze geringeren Umfangs benutzt werden.

Prototyping, als Vorgang der Entwicklung eines Prototyps, sollte jedoch nicht nur schnell sondern auch flexibel und einfach gestaltet sein. Zum einen ist es wichtig, die Kosten für Prototyping gering zu halten, zum zweiten soll vermieden werden, dass sich der Entwickler zu sehr mit dem Prototyp identifiziert und gegen Veränderungen inflexibel wird.

Um die oben genannten Anforderungen zu unterstützen, wurde in der Vergangenheit ein Vielzahl von Werkzeugen entwickelt. Damit soll der Aufwand für die Programmierung gesenkt oder ganz beseitigt werden. Die Entwicklung von Prototypen unter Einsatz von Werkzeugen wird unter dem Begriff *Rapid Prototyping* subsumiert. Eine Erweiterung stellt das *Interactive Prototyping* dar. Bei diesem kann der Benutzer den Prototyp während des Testlaufs manipulieren.

5.2 Werkzeuge

Werkzeuge sind im hier betrachteten Kontext Anwendungssysteme, die die Entwicklung von Software unterstützen. Sie sind auf verschiedenen Abstraktionsebenen – von systemnaher Programmierung bis zu benutzungs- und anwendungsnaher Modellierung – angesiedelt (vgl. [VoNe98] S. 12). Ein genereller Trend geht zu den objektorientierten Systemen mit vorgefertigten Dialogelementen hin. Das Baukastenprinzip, das in den meisten Werkzeugen die zugrunde liegende Metapher ist, ermöglicht eine einfache Anpassung der Elemente durch Änderung von Parametern. Durch die Vererbungstechniken können den Dialogelementen neues Verhalten und Aussehen verliehen werden. In modernen Werkzeugen werden die statischen Aspekte einer Benutzungsschnittstelle grafisch entwickelt. Dabei wird die WYSIWYG-Technik angewandt, bei der der Entwickler sofort sehen kann, wie der Dialog in der Anwendung später aussieht. Der Programmieraufwand wird dadurch entscheidend gesenkt. Werden Dialogelemente in ihrer Funktionalität erweitert oder verändert, so können diese in den meisten Werkzeugen integriert und benutzt werden. Ein Beispiel dafür ist das Konzept der Java Beans. Die einheitliche Verwendung von Dialogelementen führt auch anwendungsübergreifend zu einem hohem Grad an Konsistenz. Zusätzlich unterstützen Style Guides die Konformität. Dieser hohe Grad an Standardisierung hat jedoch auch Nachteile. So sind viele Komponenten weit verbreitet und bei Benutzern akzeptiert. Änderungen bzw. Innovationen an diesen Elementen werden als störend empfunden. Durch die fehlende Toleranz des Anwenders können sich diese nicht durchsetzen.

Die einfachste Unterstützung bilden User Interface Toolkits (UI-Toolkits). Ein **UI-Toolkit** stellt sich dem Programmierer als Bibliothek von Prozeduren oder Klassen mit standardisierten Dialogelementen dar (vgl. [VoNe98] S. 13). Durch Aufruf der Prozeduren oder Instanziierung der Klassen werden Interaktionselemente erzeugt. Entsprechend den Anforderungen können diese Elemente auf dem Bildschirm angeordnet, miteinander kombiniert und konfiguriert werden. Die Begrenzung der Darstellungsmöglichkeiten durch

Parameter führt zu einem einheitlichen Look and Feel. Objektorientierte UI-Toolkits bieten die Möglichkeit, neue Klassen zu bilden. Durch diese kann neues Aussehen und Verhalten für bestimmte Elemente erzeugt werden. **Application Frameworks** sind erweiterte und spezialisierte objektorientierte UI-Toolkits. Ein **Framework** ist eine Sammlung von kooperierenden Objekten, die eine integrierte Lösung eines bestimmten Anwendungsgebietes anbieten und durch den Entwickler oder Benutzer angepasst werden können (vgl. [Buzb98]). Damit liefert der Framework bereits einen Lösungsvorschlag, der durch Unterklassenbildung der vorgegeben Elemente konkretisiert wird.

Im Gegensatz zu den bisher besprochenen Werkzeugen bewegen sich **User Interface Builder** oberhalb des Programmiersprachenniveau (vgl. [VoNe98] S. 15). Ein UI-Builder gibt dem Entwickler die Möglichkeit, die Benutzungsschnittstelle grafisch zu konstruieren. Er stellt die Elemente eines Toolkits durch Menüs oder Paletten dar. Diese können durch den Entwickler direkt-manipulativ ausgewählt und positioniert werden. Die Modifikation der Eigenschaften, wie beispielsweise Farbe, Höhe und Breite wird auch grafisch unterstützt. Ein UI-Builder setzt auf einem UI-Toolkit auf. Die durch den Entwickler konstruierten Bildschirmdarstellungen werden in Kode gewandelt. Dieser kann durch das unterstützte Toolkit interpretiert werden, womit wieder ein einheitliches Look and Feel erzeugt wird. Der Vorteil der UI-Builder besteht darin, dass sich der Entwickler die Prozeduren, Klassen und Parameter des Toolkits nicht merken muss. Damit können sie auch von Designern ohne Programmierkenntnisse genutzt werden. UI-Builder bieten in den meisten Fällen einen Testmodus an. In diesem kann die entwickelte Oberfläche evaluiert werden, ohne zuvor eine länger dauernde Kodegenerierung und Übersetzung durchzuführen.

Eine noch umfangreichere Unterstützung bieten **User Interface Management Systeme** (UIMS). Diese beinhalten sowohl ein UI-Toolkit als auch einen UI-Builder. Zusätzlich bieten sie eine eigene Sprache an, um das Dialogverhalten formal zu spezifizieren. Analog zu einem Datenbank Management System (DBMS) wird während der Laufzeit der Anwendung daraus die Oberfläche generiert. Einige Systeme steigern die Leistung während der Ausführung, indem sie die Spezifikation vor der Ausführung in eine effizienter interpretierbare Form übersetzen. Moderne UIMS passen ihre eigene Sprache an die Objektstruktur des benutzten UI-Toolkits an. Neuere Systeme enthalten eine Vielzahl von einzelnen Werkzeugen. Sie können beispielsweise einen Icon-Designer oder Bildeditor integrieren. In diesen Fällen spricht man auch von User Interface Development Environments (UIDE).

Zur Veranschaulichung der verschiedenen Werkzeugebenen dient die in Abbildung 10 gezeigte Architektur. Jede einzelne Schicht nutzt die direkt zugänglichen darrunterliegenden Schichten.

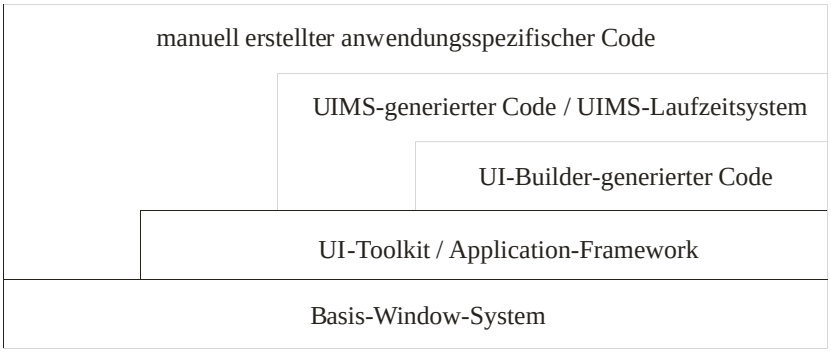


Abbildung 10 Schichtenarchitektur von UI-Werkzeugen (vgl. [VoNe98] S.17)

Wie die Abbildung andeutet, sind anspruchsvolle Benutzungsschnittstellen nur realisierbar, wenn man für bestimmte Teilbereiche direkt auf das Basis-Window-System zugreift. Dies zeigt auch die Begrenztheit aktuell verfügbarer Werkzeuge.

Bei der Entwicklung von Software gibt es viele Gebiete, die durch Werkzeuge unterstützt werden können. Jedes Werkzeug hat sowohl Stärken als auch Schwächen, die aus den verschieden Zielen resultieren. So sind beispielsweise UIMS für industrielle Anwendungen besonders auf zeitliche Aspekte ausgerichtet. Aus diesem Grund müssen für jedes Entwicklungsprojekt die Anforderungen an das zu verwendende Werkzeug definiert werden. Diese Kriterien werden von den Zielen und Rahmenbedingungen des Softwareprojektes beeinflusst. Im Ergebnis entsteht eine Katalog, der die Auswahl von geeigneten Werkzeugen ermöglicht.

Im folgenden werden vier Werkzeuge getestet und ihre Brauchbarkeit für Prototyping bewertet. Die Auswahl wurde durch folgende Bedingungen eingegrenzt. Da der GME 2001 auf der Basis von Java entwickelt wird, sollte das Werkzeug ebenfalls diese Sprache unterstützen. Der Prototyp soll nicht nur horizontal sondern auch vertikal ausgerichtet sein. Das heißt, dass teilweise Funktionalität implementiert werden muss. Damit entfallen alle Werkzeuge, die lediglich eine statische Oberfläche darstellen können. Bei der abschließenden Bewertung standen vor allem Aspekte des Prototyping und damit der Geschwindigkeit im Vordergrund. Die Ergebnisse des Vergleichs wurden im Anhang A.1 in einer Tabelle zusammengefasst. An dieser Stelle befindet sich auch die Spezifikation des benutzten

Testsystems. Dieses ist laut Herstellerangaben der zu prüfenden Werkzeuge als ausreichend beschrieben.

5.2.1 VisualAge for Java, Entry Edition 3.0

IBM bietet mit seinem VisualAge for Java eine integrierte Entwicklungsumgebung für die Programmiersprache Java an. Das Produkt wird in drei Versionen ausgeliefert. Die hier betrachtete Entry Edition wird kostenlos vertrieben. Sie beinhaltet alles um einfache Applikationen bzw. Applets zu programmieren. Allerdings darf bei ihr ein Projekt maximal aus 750 Klassen bestehen. Die Professional und Enterprise Edition sind kostenpflichtig und erweitern den Funktionsumfang. Alle Versionen stehen für die Plattformen MS Windows, Solaris und Unix zur Verfügung. Das JDK ist fest im Programm integriert. D.h. dass neuere Versionen erst genutzt werden können, wenn IBM dafür ein Update released. Die Version 3.0 wird mit dem Java Development Kit 1.2 sowie mit der Java Foundation Class Version 1.0.3 ausgeliefert.

Die Installation gestaltet sich sehr einfach und benötigt ca. 200 MB Festplattenkapazität. Nach dem Programmstart, der knapp eine Minute dauert, erscheint als erstes ein Wizard, der durch die Erstellung eines neuen Projektes führt. Die empfohlenen 64 MB Arbeitsspeicher erscheinen zu wenig, da das Programm bereits ca. 50 MB Speicher belegt. Hinzu kommt, dass für jedes neue Bearbeitungsfenster weiter fünf MB belegt werden. Nachdem das neue Projekt erstellt wurde, befindet man sich im Workbench, dem Arbeitsplatz. Dieser ist Ausgangspunkt für alle Aktivitäten. Er listet alle Projekte, Packages, Klassen und Interfaces auf. Von hier aus können Arbeitsstände importiert und exportiert sowie die Versionierung durchgeführt werden. Damit ermöglicht er die Navigation durch das gesamte Projekt. Der Import bzw. Export wird benötigt, da IBM in VisualAge for Java ein Repository für die Speicherung aller Daten benutzt. D.h., dass sowohl alle Quellkodes als auch die übersetzten Bytekode in einer eigenen Datenbank gesichert werden. Um die Daten im Repository anzusehen, hat IBM den Repository Explorer integriert. Mit ihm kann durch die gesamte Datenbank navigiert werden. Projekte, Klassen, Interfaces können von hier aus in den Workbench transferiert werden. Erst dann ist eine Bearbeitung bzw. Nutzung dieser möglich. Das Repository unterstützt ebenfalls eine Versionsverwaltung. Es ist möglich, einem Projekt, einzelnen Klassen oder Interfaces Versionen zuzuteilen und zu verwalten. Dies stellt jedoch nur einen einfachen Mechanismus und kein durchgängiges Konfigurationsmanagement dar.

Nach Anlegen oder Öffnen eines Projektes kann man neue Klassen mit Hilfe eines Wizard erstellen. Diese werden in einem Bearbeitungsfenster modifiziert, in dem die eigentliche

Programmierung stattfindet. Das Hauptaugenmerk liegt dabei auf dem *Visual Composition Editor*. In diesem werden alle visuellen Komponenten angezeigt. Auf der linken Seite befindet sich eine Palette aller im Moment verfügbaren Beans. Standardmäßig sind alle AWT und Swing Komponenten eingestellt. Weitere Standardelemente sowie eigene Entwicklungen können hinzugefügt werden. Die einzelnen Komponenten werden mittels Drag and Drop in dem Arbeitsbereich platziert. Mit Hilfe eines Eigenschaftseditor können ihre Eigenschaften (engl.: property), wie Größe, Namen, Ausrichtung usw. geändert werden. IBM hat in dem Visual Component Editor das Konzept der *Connections* umgesetzt. Damit kann jedem Ereignis (engl. event), welches von einem Bildschirmelement ausgelöst wird, eine entsprechende Aktion zugewiesen werden kann. Dazu klickt man mit der Maus auf das entsprechende Ereignis der Quellkomponente und anschließend auf eine Methode des Zielobjektes. Rein theoretisch ist es damit möglich, ganze Anwendungen zu programmieren, ohne eine einzige Zeile Kode zu schreiben. Über das Symbol *Run* kann die programmierte Anwendung gestartet werden. Positiv überrascht die schnelle Ausführungsgeschwindigkeit. Das liegt vor allem in der Maßnahme begründet, dass Klassen beim Speichern im Repository gleichzeitig übersetzt werden und damit auch als Bytekode vorliegen.

VisualAge for Java bietet einen integrierten Debugger. Threads können gestoppt und wieder fortgesetzt werden, Variablenwerte angesehen und modifiziert werden. Breakpoints werden ebenfalls unterstützt. Zusätzlich ist die Nutzung relationaler Datenbanken möglich. Die Internationalisierung wird mit dem von IBM implementierten *Java Internationalization Framework* unterstützt. Das Programm ermöglicht die Anbindung von externen Konfigurationsmanagementsystemen. Der größte Nachteil von VisualAge for Java liegt bei dem fest integriertem JDK. Da der GME 2001 mit dem JDK 1.3 implementiert werden soll, eignet sich das Produkt für die Entwicklung des Prototypen nicht.

5.2.2 Forte for Java, Community Edition 1.0

Sun microsystems selbst bietet mit Forte for Java eine Integrierte Entwicklungsumgebung an. Wie IBM gibt es auch dieses Produkt in drei Versionen. Die Community Edition ist für nicht kommerzielle Nutzung ohne Einschränkung frei benutzbar. Ein Teamunterstützung ist in dieser Version nicht enthalten. Die Internet Edition sowie die Enterprise Edition sind kostenpflichtig. Auch Sun microsystems unterstützt die Plattformen Solaris, MS Windows und Unix.

Die Installation gestaltet sich ähnlich einfach bei VisualAge for Java. Der einzige zusätzliche Schritt ist die Abfrage des Installationspfades des JDK. Da Forte for Java nicht explizit an

eine bestimmte Version des JDK gebunden ist, wird es auch ohne ein solches ausgeliefert. Dadurch lässt sich ein Update auf eine neue Version des JDK durchführen, ohne eine neue Version von Forte for Java zu benötigen. Mit nur 30 MB Festplattenbedarf benötigt es am wenigsten Platz von den hier getesteten Tools. Nach dem Start des Programms, erscheinen Haupt-, Explorer- und Eigenschaftsfenster. Bereits zu diesem Zeitpunkt werden knapp 60 MB Arbeitsspeicher belegt.

Anschließend kann man mit einem neuen Projekt beginnen oder den Einstieg mit dem Hilfesystem suchen. Der *Getting Started Guide* beschreibt kurz die Erstellung einer kleinen Applikation.

Startet man ein neues Projekt können anschließend neue Klassen wie beispielsweise Fenster, Java Beans oder JSP über den *Template Chooser* angelegt werden. Dieser stellt eine einfachere Form eines Wizard dar. Er ermöglicht lediglich die Auswahl aus vorgefertigten Mustern. Navigiert wird in Forte for Java mit Hilfe über des Explorerfensters. In diesem erscheinen alle visuellen sowie nicht visuellen Komponenten. Im Hauptfenster kann man zwischen den fünf Workspaces Editing, GUI-Editing, Browsing, Running und Debugging wechseln. Alle Fenster sind grundsätzlich einem Workspace zugewiesen. Der Editing Workspace dient ausschließlich zum modifizieren der Java Quelldateien. Forte for Java verzichtet auf die Benutzung eines speziellen Repository und speichert alle Daten in normalen Java-Quelldateien. Den Designmodus stellt der Workspace GUI-Editing bereit. Hier wird der aktuell ausgewählte Container, z.B. ein Fenster, grafisch dargestellt. In diesen können Komponenten über die Komponentenpalette eingefügt werden. Die Komponentenpalette kann durch den Benutzer selbst erweitert werden. Über den Eigenschaftseditor werden die Attribute wie Farbe, Position oder Name gesetzt. Ebenso können hier den Ereignissen Methoden zugeordnet werden. Übersetzt und startet man das Projekt, wird automatisch in den Workspace Running gewechselt. Hier findet sich das typische Outputfenster sowie ein Execution View Fenster. Dieses listet alle aktiven Threads der Java-Applikation auf. Im Workspace Debugging kann man sein Projekt nach Fehlern durchsuchen. Er ermöglicht das Setzen von Breakpoints sowie die Anzeige der überwachten Variablen und Ausdrücke.

Eine gute Unterstützung bei der Eingabe bietet die *dynamic code completion*. Nach dem Eintippen eines Klassennamens öffnet sich eine Liste, die alle dazugehörigen Methoden und Eigenschaften anzeigt. Zum einen verkürzt sich die Zeit beim Schreiben, zum anderen wird die Anzahl der Tippfehler gesenkt. Durch den modularen Aufbau des Produktes kann es

ständig ergänzt und erweitert werden. Dabei müssen diese Module nicht von Sun microsystems sein.

Beim Ausführen eines Projektes steigt der Ressourcenverbrauch stark an. So werden selbst bei kleinen Anwendungen 130 MB Speicher genutzt. Daraus resultieren relativ lange Wartezeiten während der Entwicklung. Auch der Wechsel von Workspaces kann sehr lange dauern. Damit ist Forte for Java für die Entwicklung von Prototypen ungeeignet.

5.2.3 VisualCafé 4.0 Standard Edition

Die Firma WebGain bietet ebenfalls eine IDE für Java an. Auch dieses Produkt gibt es in drei Varianten, die hier getestete Standard Edition, eine Expert Edition sowie eine Enterprise Edition. Die Standard Edition ist für den privaten Gebrauch frei. WebGain unterstützt lediglich die Plattform MS Windows. Die entwickelten Java Anwendungen können auch auf anderen Plattformen genutzt werden.

Die normale Installation benötigt ca. 500 MB Festplattenspeicher und ist mit über 13.000 Dateien sehr umfangreich. Die Konzentration auf nur eine Plattform hat positive Auswirkungen auf den Ressourcenverbrauch des Produktes. So startet das Programm innerhalb von acht Sekunden und belegt lediglich 20 MB Hauptspeicher. Beim Anlegen eines neuen Projektes kann man sich zwischen verschiedenen Vorlagen entscheiden. Diese können durch eigene Projekte erweitert werden.

Anschließend befindet man sich im Workspace *Edit.* Dieser besteht aus dem Hauptfenster, in dem sich die Komponentenpalette befindet, dem Projektexplorer, dem *Formdesigner* sowie dem *Property-List* Fensters. Im Projektexplorer werden alle Objekte, Klassen und Dateien des aktuellen Projektes angezeigt. Er dient zum navigieren. Der Formdesigner stellt alle visuellen und nicht visuellen Komponenten des aktuellen Container dar. Die Darstellung der zweiten Gruppe ist für die Nutzung des Connection Wizard sinnvoll. Das Property-List Fenster zeigt alle Eigenschaften eines Objektes an. Hier können beispielsweise Größe, Farbe oder Positionen eingestellt werden. Der Connection Wizard verbindet, ähnlich wie bei VisualAge for Java, Ereignisse mit bestimmten Aktionen. Die Zuordnung wird mit der Maus durchgeführt. VisualCafé bietet für häufig benötigte Aktionen vorbereitete Methoden an. Benötigte Parameter können entweder automatisch durch Eigenschaften anderer Objekte oder manuell übergeben werden. Bei Anwendung des Connection Wizard können sehr schnell Anwendungen mit umfangreicher Funktionalität entwickelt werden. Dabei muss kein Java Quellkode eingegeben werden.

Der zweite Standard Workspace heißt *Debug*, mit dem ein Testen des Projektes ermöglicht wird. Es werden Fenster zur Variablenüberwachung eingeblendet. VisualCafé gestattet es dem Benutzer eigene Workspace zu erstellen, umzubennen sowie zu löschen. Hinzu kommen umfangreiche Möglichkeiten die Entwicklungsumgebung anzupassen. Die Speichernutzung steigt während der Projektausführung auf ca. 50 MB. Damit benötigt das Produkt von den getesteten Werkzeugen den wenigsten Speicher.
Dieser Vorteil wird hauptsächlich durch die Plattformabhängigkeit erkauft. Diese ist aber vor allem beim Oberflächendesign von Nachteil. Es ist unbedingt zu empfehlen, ein Projekt auf allen Plattformen zu testen, auf denen es später angewendet werden soll. Ein weiterer Nachteil ist die Abhängigkeit vom mitgelieferten JDK. Damit ist auch dieses Tool nicht für die Entwicklung des Prototypen GME 2001 geeignet.

5.2.4 JBuilder 4.0 Foundation

Borland/Inprise bietet mit dem JBuilder ebenfalls eine Java Entwicklungsumgebung an. Auch dieses Produkt gibt es in drei verschiedenen Varianten. Die Foundation Edition ist kostenlos zu erhalten. Die Professional sowie die Enterprise Edition erweitern den Umfang der Foundation um Funktionen wie Datenbankzugriff, Teammanagement oder Versionsverwaltung.
Das Programm richtet bei der problemlos verlaufenden Installation das JDK Version 1.3 ein. Dieses kann jedoch gegen eine beliebige Version ausgetauscht werden. Nach dem Programmstart, ist das Hauptfenster sichtbar. Die Speicherbelastung, ist mit 30 MB relativ gering. Der JBuilder besteht im Gegensatz zu den anderen Werkzeugen nur aus einem Fenster, in dem alle Elemente wie Projektbrowser, Strukturbrowser, GUI-Builder und Editor eingebettet sind.
Der Projektbrowser ermöglicht die Navigation durch das gesamte Projekt. Er zeigt alle verfügbaren Klassen an. Der Strukturbrowser zeigt die innere Organisation der aktuell ausgewählten Klasse an. Dies ist vor allem für Fenster von Vorteil. Im Arbeitsbereich kann zwischen verschiedenen Sichten gewechselt werden. Diese sind analog zu den Workspaces der anderen Werkzeugen. Der Editor ist wie auch in forte for Java mit einer *dynamic code completion* ausgestattet. Bei visuellen Klassen kann die Darstellung im GUI-Builder erfolgen. Zusätzlich wird ein Eigenschaftseditor und die Komponentenleiste angezeigt. Mittels Drag and Drop können die Bildschirmelemente in dem Fenster platziert werden. Die Manipulation ihrer Attribute erfolgt mit Hilfe des Eigenschaftseditor. Dieser zeigt auch alle Ereignisse des momentan aktiven Elementes an. Diesen kann direkt Java Quellkode zugewiesen werden, der

jedoch programmiert werden muss. Weitere Sichten im Arbeitsbereich sind die Dokumentation und die Versionsverwaltung. Die Versionsverwaltung ist ähnlich einfach wie bei VisualAge for Java gehalten. Die Dokumentation wird in HTML-Form verwaltet.
Der JBuilder bietet eine Vielzahl von Individualisierungsmöglichkeiten. So kann man beispielsweise Quellkode als Vorlage (engl.: template) anlegen. Diese kann mit einem Ereignis einer Klasse verknüpft werden. Damit kann ein ähnliches Konzept wie das des Connection Wizards in VisualCafé aufgebaut werden.
Auch während der Ausführung bleibt der Speicherbedarf mit ca. 55 MB relativ gering. Lediglich im Debug-Modus wird die Entwicklungsumgebung deutlich langsamer. Vorteile sind die Unabhängigkeit sowohl von der Version des JDK als auch von der Plattform des Betriebssystems. Die grafische Gestaltung von Fenstern ist einfach und schnell durchzuführen. Eine Erweiterung mit Funktionalität ist ohne Schwierigkeiten durchführbar.
Aus den genannten Gründen wird der Prototyp GME 2001 mit Hilfe des Werkzeuges JBuilder 4.0 entwickelt.

5.3 Entwicklung des Prototypen GME 2001

Im Rahmen der Diplomarbeit wurde ein Prototyp für den Generischen Modelleditor 2001 (GME 2001) entwickelt. An diesem Projekt wird seit Februar 2000 an der TU Dresden am Lehrstuhl für Systementwicklung intensiv gearbeitet (vgl. [Grei00a]). Das Ziel ist die Entwicklung eines innovativen Modellierungswerkzeuges.
Der Prototyp ist in dem Bereich *Division Client* des Projektes angesiedelt. Dieser Teil beschäftigt sich mit der Entwicklung der Clients. Diese sind auf die Servermodule aufgesetzte Anwendungen, die die Kommunikation zwischen Modellierern und Datenbank (Repository) übernehmen. Damit fällt in diesen Bereich auch die Entwicklung einer Benutzungsschnittstelle. Der hierfür entwickelte Prototyp soll vor allem die herausgearbeiteten ergonomischen Aspekte beachten. Der GME 2001 wird mit der Programmiersprache Java implementiert. Daraus ergibt sich, dass auch der Prototyp für die Oberfläche in Java umgesetzt wird. Die Benutzung einer einheitlichen Sprache ist zwar nicht zwingend erforderlich, vereinfacht aber die iterative Entwicklung. Für die Umsetzung wird das Werkzeug JBuilder 4.0 verwendet. Es ist eine der schnellsten Entwicklungsumgebungen für Java-Applikationen. Zusätzlich kann das aktuelle Java Development Kit 1.3 verwendet werden, mit dem bereits vorhandene Teile des GME 2001 implementiert wurden.

5.3.1 Beschreibung des E^3-Ansatz

Der Generische Modelleditor 2001 basiert auf dem E^3-Ansatz. Dieser Ansatz verfolgt das Ziel, Modellbeschreibungen zu standardisieren. Unter einem **Modell** wird eine formale Sprache zur Beschreibung von Schemata verstanden. **Schemata** sind formale Abbildungen von realen Sachverhalten bzw. Problembereichen (vgl. [GrEs01] S. 5). Ein **Meta-Modell** ist eine formale Sprache zur Beschreibung von Modellen (vgl. [FeSi98] S. 119). Eine Gruppe von Bestandteilen oder der Ausschnitt eines Schemas wird als **View** bezeichnet. Die Darstellung einer View wird als **Präsentation** definiert.

Das **E^3-Modell** ist ein Meta-Modell, das die einheitliche Definition von Modellen ermöglicht. Es gliedert sich in zwei Teile, der Typebene und der Instanzenebene. Die **Typebene** nimmt alle Modelle der Beschreibungsmodelle auf (vgl. [GrEs01] S. 8). Die Modelle werden im GME 2001 im **Typeneditor** erstellt und bearbeitet. Die **Instanzenebene** nimmt alle Ausprägungen dieser Modelle, also alle Schemata auf. Diese werden im GME 2001 im **Schemaeditor** bearbeitet. Die Typ und Instanzenebene werden vertikal in die drei Teile Kontextebene, Viewebene und Präsentationsebene unterteilt. Das Modell wird zur Verfeinerung auch horizontal in drei Ebenen gegliedert. Daraus ergeben sich insgesamt 18 Berührungspunkte, die als **E^3-Elemente** bezeichnet werden (vgl. Anhang A.2). Auf die genaue Beschreibung der einzelnen Elemente und ihre Beziehungen untereinander wird hier nicht eingegangen.

5.3.2 Ziele des Prototypen

Bei der Erstellung des Prototypen ergeben sich zwei Hauptgebiete. Die erste Aufgabe umfasst die Beschreibung der Oberfläche des gesamten GME 2001. Hierzu wurde der Prototyp horizontal ausgerichtet. Dabei standen verschiedene Anforderungen im Vordergrund:

- Unterstützung eines Konfigurationsmanagement
- Teamfähigkeit
- verteilte Architektur

Die zweite Aufgabe betrifft die Entwicklung des Typeneditors. Bei diesem wurde auch Funktionalität implementiert. Dazu wurde der Prototyp zusätzlich vertikal ausgerichtet. Besondere Beachtung muss dem Umstand beigemessen werden, dass es in der neuen Spezifikation des E3-Modells möglich ist, Views auf Views zu bilden (vgl. Anhang A.3). Der Schemaeditor wurde im Prototyp nicht implementiert.

5.3.3 Spezifikation des Prototypen

Die Entwicklung des Prototypen begann mit der Erfassung der nötigen Informationen. Für den Arbeitsablauf war die bereits vorhandene Version 0.91 des GME hilfreich. Diese beinhaltet sowohl den Typeneditor, als auch den Schemaeditor. Die Anforderungen, die im Abschnitt 5.3.2 beschrieben wurden, erfordern jedoch eine umfassende Projektverwaltung. Der Benutzer wurde in der Anforderungsspezifikation des GME 2001 (vgl. [Grei00b]) als Modellierer definiert. Dieser hat ein grundsätzliches Verständnis über den Ablauf eines Modellierungsprojektes. Aus diesem Grund wurde auch diese Metapher im Workspace eingesetzt. Der Benutzer erhält den Eindruck, ein Projekt durchzuführen, in dessen Rahmen Modelle und Schemas erstellt bzw. bearbeitet werden. Diese Projektsicht wird in vielen Werkzeugen eingesetzt, was zu einer hohen Erwartungskonformität führt. Um eine entsprechende funktionale Oberfläche zu erhalten, wurden im zweiten Schritt Use case Diagramme sowie Klassendiagramme entwickelt (vgl. Anhang C.1). Diese wurden dann mit Hilfe des Tools JBuilder 4.0 in einen Prototypen umgesetzt.

Dabei wurde vor allem dem Workspace und dem Typeneditor Beachtung geschenkt. Beide Teile werden deshalb noch näher spezifiziert. Das Hilfesystem wurde durch einen HTML-fähigen Browser dargestellt (vgl. Abschnitt 3.4.6). Diese enthält keine Möglichkeiten zum Information Retrieval. Eine entsprechende Funktionalität muss noch ergänzt werden. Die Individualisierungsmöglichkeiten wurden angedeutet, jedoch nicht implementiert. Auch sie müssen noch ergänzt werden. Die im Prototyp verwendeten Icons sind zum größten Teil aus dem GME 0.91. Sie müssen entsprechend den Empfehlungen und Richtlinien dieser Arbeit (vgl. Abschnitt 3.4.4) noch überarbeitet werden.

5.3.3.1 Workspace

Der Workspace übernimmt die Verwaltung des gesamten Modellierungsprojektes. Er ist in die Bereiche Projektbrowser, Arbeitsbereich und Aufgabenmanagment unterteilt (vgl. Anhang A.4). Der Projektbrowser enthält alle Objekte des aktuell ausgewählten Projektes. Dazu gehören die Projektdaten, alle Schemata und Modelle, die Dokumentation, die beteiligten Mitarbeiter, Scripte und die Change Requests. Letzteres sind Änderungsanträge, die erst bei einem fortgeschrittenen Projektstatus erforderlich sind. Zusätzlich wurde die Kategorie sonstiges eingeführt, die als Ordner für alle zum Projekt gehörenden Dokumente dient. Das können beispielsweise vom Auftraggeber gelieferte Spezifikationen vorhandener Style Guides o.a. sein. Jedes Projekt wird durch die Klasse **`Project`** abgebildet. Die Spezifikation befindet sich im Anhang C.3.1. Der Arbeitsbereich

passt seinen Inhalt entsprechend der Auswahl im Projektbrowser an. Hier werden die Detailinformationen zum gewählten Objekt angezeigt. Diese Informationen können an dieser Stelle direkt geändert werden. Voraussetzung ist eine entsprechende Berechtigung beim Benutzer. Das Aufgabenmanagement wird im Workspace durch eine Tabelle dargestellt. Diese zeigt für den angemeldeten Benutzer alle verfügbaren Einträge an. Eine Aufgabe wird im Prototyp durch die Klasse **`Task`** (vgl. Anhang C.3.2) abgebildet. Die Tabelle kann durch den Benutzer nach Verlangen entsprechend sortiert werden. Überfällige Aufgaben werden farblich hervorgehoben. Die Einführung des Aufgabenkonzeptes bringt zwei Vorteile. Zum einen kann der Benutzer seinen Arbeitsablauf selbst strukturieren. Zum anderen kann durch die Erstellung von Aufgaben, sowohl durch den GME 2001 selbst als auch durch andere Mitarbeiter, der Prozess gesteuert werden. Wechselt beispielsweise der Modellierer den Status eines Schemas von edit auf stable so kann der GME 2001 automatisch die Aufgabe „Schema testen“ generieren. Sind mehrere Tester im Projekt vorhanden, so muss der Modellierer nur noch den entsprechend verantwortlichen Mitarbeiter auswählen. Die so generierte Aufgabe erscheint anschließend nicht im eigenen sondern im Workspace des Testers. Zusätzlich kann im Aufgabenmanager zu jedem Mitarbeiter eine Liste aller Aufgaben eingesehen werden. Der Aufgabenmanager wird in einem eigenen Fenster dargestellt. Damit können Engpässe durch den Projektleiter erkannt und beseitigt werden. Bei dieser Funktion sind jedoch Datenschutzbestimmungen zu beachten.

Für die Erstellung eines neuen Projektes, Modells, Schemas oder einer neuen Aufgabe wurden Wizards (vgl. Anhang A.5) implementiert. Diese unterstützen den Benutzer bei der Erfassung relevanter Daten. In den Wizards nicht erfasste Daten können zu einem späteren Zeitpunkt im Arbeitsbereich modifiziert und ergänzt werden.

Der aktuelle Arbeitsstand wird immer lokal gespeichert. Für die Überführung der lokalen Daten in das Repository bzw. zurück stellt der Workspace die zwei Funktionen Checkin und Checkout zur Verfügung. Beide Operationen werden ebenfalls mittels eines Wizards unterstützt. Vor allem für den Checkin, also die Überführung der lokalen Daten in das Repository, wird dieser benötigt. Ist die einzubringende Konfiguration bereits vorhanden, so muss der Benutzer über die weitere Behandlung entscheiden. Wird weder die Funktion branch noch die Funktion merge durchgeführt so muss lediglich eine neue Versionsnummer gebildet werden. Beim branch wird eine Verzweigung der Version vorgenommen. Es entstehen damit zwei Elemente mit einem gemeinsamen Vorgänger. Beim merge werden die lokalen Elemente

mit den im Repository vorhandenen zu einem einzigen vereint. Es entsteht ein neues Element mit zwei Vorgängern.

Die jeweilig verfügbaren Versionen werden sowohl textuell, d.h. in Listen und Bäumen angezeigt, als auch als durch einen Konfigurationsgraph visualisiert. Dies ist im Prototyp nur durch entsprechende Bildschirmelemente angedeutet, jedoch nicht implementiert.

Der Workspace ermöglicht ebenfalls den Zugriff auf das Benutzermanagement, das Konfigurationsmanagement und die statistische Auswertung. Die Funktionen wurden nur angedeutet, jedoch nicht implementiert. Die statistische Auswertung soll vor allem zur Nachkalkulation der Projektkosten sowie als Erfahrungspool für zukünftige Projekte dienen.

Der Workspace gestattet es dem Benutzer ein neues Modell anzulegen. Dabei wird eine neue Instanz der Klasse **`Model`** erzeugt. Diese sammelt alle Informationen über das Modell (vgl. Anhang C.3.3). Der eigentliche Vorgang der Modellierung findet im Typeneditor statt. Dieser wird vom Workspace aus unter Angabe des relevanten Modells gestartet.

5.3.3.2 Typeneditor

Der Typeneditor dient zur Entwicklung von Modellen. Um die Gliederung des E3-Modells widerzuspiegeln, wurden die Kontext-, View- und Präsentationsebene nebeneinander dargestellt. Jede Ebene wird durch einen Baum visualisiert. Damit lassen sich die Verfeinerungen innerhalb der Ebene ebenfalls abbilden (vgl. Abbildung 11). Jedes Element in einem dieser Bäume entspricht einem E3-Element.

In der Kontextebene lassen sich ObjektTypen und ObjektPropertyTypen anlegen und löschen. Die neu erstellten Elemente dieser Ebene werden grundsätzlich grau dargestellt. In der Viewebene lassen sich nur ViewTypen anlegen. Diese können hierarchisch gegliedert werden. Damit können View- auf ViewTypen abgebildet werden. ViewObjektTypen und ViewPropertyTypen werden nur indirekt angelegt bzw. entfernt. Dazu wird der entsprechende ObjektTyp bzw. ObjektPropertyTyp selektiert und dem ViewTypen zugeordnet. Ist eine Zuordnung erfolgt, so wird das Element in der Kontextebene nicht mehr grau, sondern schwarz dargestellt. Daraus ergibt sich, dass alle grau dargestellten Elemente der Kontextebene in dem selektierten ViewTypen verfügbar sind. Wird ein untergeordneter ViewTyp selektiert, so werden alle dem übergeordneten ViewTyp zugeordneten ObjektTypen bzw. ObjektPropertyTypen grau dargestellt. Die restlichen Elemente sind in dem untergeordneten ViewTypen nicht verfügbar und werden ausgeblendet. Ein PräsentationsObjektTyp ist ebenfalls immer genau einem ViewTypen zugeordnet. Aus diesem

Grund ändert sich die Menge der angezeigten PräsentationsTypen in Abhängigkeit der Selektion in der Viewebene. Die folgende Grafik verdeutlicht diesen Sachverhalt.

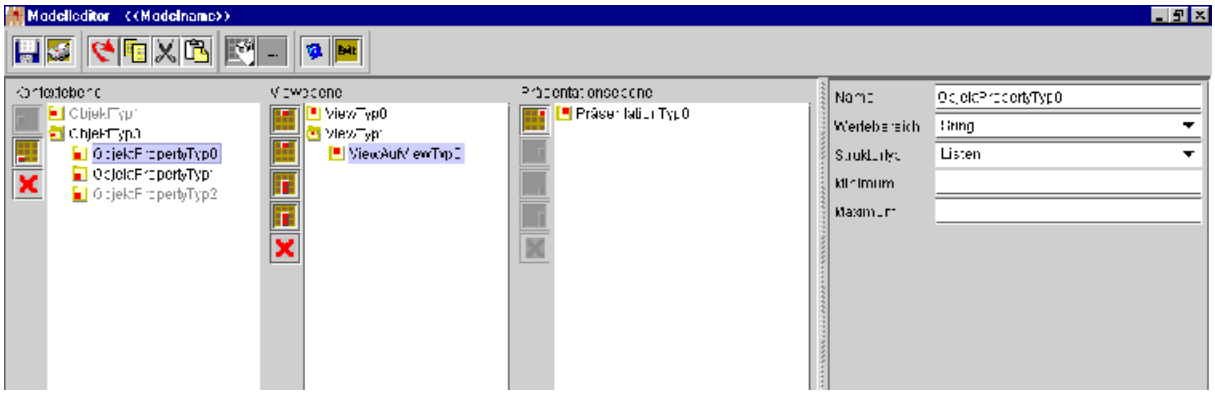


Abbildung 11 Typeneditor

Die Abbildung 11 stellt den Typeneditor während der Benutzung dar. Aus der farblichen Gestaltung der Kontextebene lässt sich erkennen, dass ObjektTyp1 und ObjektTyp3 inklusive ObjektPropertyTyp0 bis ObjektPropertyTyp2 dem ViewTyp1 zugeordnet wurden. Der ObjektTyp0 und ObjektTyp2 werden hier nicht dargestellt, da diese für den selektierten ViewAufViewTyp0 nicht verfügbar ist. Diesem Typ wurde der ObjektTyp3 und zwei ObjektPropertyTypen zugeordnet, die damit in möglichen untergeordneten ViewTypen sichtbar sind. Dem ViewAufViewTyp2 ist ein PräsentationsTyp zugeordnet. Rechts von der Präsentationsebene ist der Eigenschaftseditor eingeblendet. Die in der Symbolleiste angezeigten Piktogramme weisen derzeit drei verschiedene Stilarten auf. Im Rahmen der iterativen Entwicklung muss diese ergonomische Schwäche noch beseitigt werden.

Die E3-Elemente besitzen unterschiedliche Eigenschaften, die im Eigenschaftseditor bearbeitet werden können. Dieser kann je nach Wunsch ein- bzw. ausgeblendet werden. Das Hinzufügen, Entfernen und Bearbeiten von Elementen wird über die jeweils links vom Baum abgebrachten Buttons realisiert. Die aktuelle Selektion in den verschiedenen Bäumen steuert ihre Verfügbarkeit. So kann ein PräsentationsObjektTyp nur angelegt werden, wenn in der Kontextebene ein ObjektPropertyTyp selektiert ist und dieser in der aktuell ausgewählten View eingebunden ist. Sind die Voraussetzungen nicht erfüllt, wird der Button disabled, d.h. der Benutzer kann ihn nicht anklicken. Der Zustand wird durch eine graue Darstellung des Button signalisiert.

Im unteren Teil des Typeneditor soll eine grafische Darstellung des E3-Modells angezeigt werden. Denkbar ist eine automatische Generierung von Struktur- oder Kontextdiagrammen. Die Darstellung wurde mit einer Grafik angedeutet, eine funktionale Implementierung existiert im Prototyp nicht.

6 Zusammenfassung und Ausblick

Das Gebiet der Softwareergonomie ist sehr komplex. Eine umfassende Erfassung ist innerhalb eines Werkes kaum mehr möglich. Dies liegt vor allem an dem interdisziplinären Ansatz. So sind die kognitiven Fähigkeiten des Benutzers die Grundlage, auf der die weiteren Aspekte aufbauen. In diesem Bereich der Psychologie sind trotz intensiver Forschungen noch erhebliche Wissenslücken aufzufinden. So ist beispielsweise der Prozess des Denkens nach wie vor nicht vollständig geklärt. In dieser Arbeit wurde sich darauf beschränkt, eindeutig geklärte und wesentliche Aspekte der Psychologie zu nennen und kurz auf ihre Bedeutung für die Softwareergonomie einzugehen. Für ein genaueres Studium sei auf entsprechende Literatur verwiesen.

Der in Abschnitt 3.1 beschriebene benutzerorientierte Gestaltungsprozess stellt nur einen Teil des gesamten Softwareentwicklungsprozeß dar. Er ist an die Besonderheiten bei der Entwicklung von Benutzungsschnittstellen angepasst. Auf Techniken, die seine Einbindung in das Gesamtvorhaben unterstützen, wird dabei nicht eingegangen. PREIM diskutiert dieses Thema eingehender und weist auf entsprechende Schwierigkeiten hin (vgl. [Prei99] S. 219ff). Die Gestaltungsgrundsätze wurden eingehend erläutert. Die Entwurfstechniken und -prinzipen, die für ihre Umsetzung genutzt werden, können an dieser Stelle nicht erschöpfend behandelt werden. Aus diesem Grund wurden lediglich die wichtigsten genauer beschrieben. Welche der genannten im konkreten Projekt relevant sind, ergibt die genaue Analyse des Nutzungskontext. Die Evaluierung von Benutzungsschnittstellen stellt einen sehr wichtigen, aber auch komplexen Schritt im benutzerorientierten Gestaltungsprozess dar. Die dabei anzuwendende Methodik konnte im Rahmen dieser Arbeit nur angedeutet werden. Im Hinblick auf den praktischen Teil der Diplomarbeit wurde das Kapitel Prototyping ausführlich behandelt. Die zeitlichen Beschränkungen erlaubten jedoch keine vollständige Entwicklung des Prototypen. Dieser befindet sich noch in der ersten Iterationsphase und muss für eine Evaluierung weiter ausgebaut werden. Ein erster Eindruck über Aussehen und Verhalten, vor allem des Typeneditor, kann jedoch bereits vermittelt werden.

Im folgenden Abschnitt soll ein Überblick über zukünftige Entwicklungen und Trends gegeben werden. Die dazugehörigen Erläuterungen sollen auf den bestehenden Forschungsbedarf des entsprechenden Gebietes hinweisen.

Die heute vorherrschende Form von Benutzungsoberflächen sind grafische Fenstersysteme (vgl. [Prie99] S. 463). Daran wird sich auch in den nächsten Jahren wenig ändern. Gestützt

wird dieser Trend auch von den technischen Neuerungen wie Flachbildschirmen, schnelleren Prozessoren und immer leistungsfähigeren Grafikkarten. Damit wird die Qualität der visuellen Informationsdarstellung stetig verbessert.

Andererseits ist die Komplexität von Anwendungssystemen ist in den vergangenen Jahren enorm gestiegen. Somit wird es für den Benutzer immer schwieriger, die gesamte Funktionalität zu erkennen und zu nutzen. Die vorgestellten Techniken zum Entwurf von ergonomische Oberflächen begrenzen dieses Problem, können es jedoch nicht lösen. Ein Ansatz, der derzeit in der Wissenschaft verfolgt wird, ist die Entwicklung intelligenter Benutzungsschnittstellen. Unter **intelligent** versteht man die Fähigkeit einer Oberfläche, sich selbstständig an die Bedürfnisse des Benutzers anzupassen. Aus diesem Grund werden intelligente Benutzungsschnittstellen auch adaptive Systeme genannt. Zu ihrer Realisierung werden Methoden und Techniken der Künstlichen Intelligenz (KI) genutzt.

Bei heutigen Oberflächen sind umfangreiche Individualisierungsmöglichkeiten vorgesehen. Diese werden jedoch meist vom Benutzer aus initiiert. Benötigt beispielsweise ein Anwender häufig eine Option, die im Menübaum sehr tief angeordnet ist, so wird er dieser eine Beschleunigungstaste (shortcut key) zuweisen. Voraussetzung dafür ist, dass er diese Funktionalität kennt. Ist das System adaptiv, so erkennt es selbstständig, dass die Funktion oft genutzt wird und schlägt dem Benutzer den Anpassungsschritt vor. Eine weitere Möglichkeit besteht darin, einmal durchgeführte Eingaben als Voreinstellungen beim nächsten Dialogablauf zu verwenden.

Diese relativ einfachen Anpassungen sind mit herkömmlichen Programmiermethoden leicht zu realisieren. Voraussetzung dafür ist lediglich die Erfassung eines Dialogprotokolls. Diese erfasst sämtliche Interaktionsschritte und ermöglicht statistische Auswertungen. Erweiterte Systeme protokollieren nicht nur die einzelnen Schritte des Benutzers, sondern stellen die einzelnen Aktionen des Benutzers als Fakten dar. Diese können auch noch mit Wahrscheinlichkeiten gewichtet werden. Daraus können diese Systeme neues Wissen ableiten. Somit entsteht ein umfassendes Benutzermodell innerhalb des Anwendungssystems. Diese Methodik ist jedoch nur sinnvoll, wenn der Benutzer häufig mit dem Anwendungssystem arbeitet.

Ein weiterer Forschungsschwerpunkt liegt auf dem Gebiet der Agenten. Agenten erledigen für ihren Auftraggeber beziehungsweise Benutzer Aufgaben, für die ein spezielles Fachwissen notwendig ist, oder die aus vielen zeitaufwendigen Einzelschritten bestehen (vgl. [Bren+98] S. 22). Sie lassen sich in menschliche, Hardware- und Softwareagenten

kategorisieren. Mit Hilfe der hier betrachteten Softwareagenten soll die Benutzungsschnittstelle personifiziert werden. Ihre Entwicklung basiert auf der Metapher des persönlichen Assistenten. Beispiele für Agenten sind animierte Menschen oder comicartige Figuren. Sie sollen dem Benutzer Aufgaben abnehmen oder ihn anderweitig unterstützen. Ein Agent kann dabei durch seine Gestik und Mimik Informationen vermitteln. Der Benutzer hat die vollständige Kontrolle über den Agenten. So kann er Aufgaben delegieren, die dann selbstständig durch den Agenten durchgeführt werden. Nach Abschluss meldet sich der Agent automatisch zurück und berichtet über das Ergebnis seiner Arbeit. Im Idealfall kann der Agent die Informationen verdichten, um eine unnötige Belastung des Benutzers zu vermeiden. Der Agent orientiert dabei seine Kommunikationsart am Benutzer. Dies macht seinen Einsatz vor allem bei ungeübten Benutzer bzw. Anfänger sinnvoll. Wie der Anwender während der Benutzung des Systems dazu lernt, sollte das auch der Agent tun. Auch er kann sich ein Benutzermodell aufbauen und damit noch effizienter unterstützen. Agenten sind für schlecht strukturierte, große Informationsräume, wie beispielsweise das WWW nützlich. Die existierenden Suchmaschinen sind bereits einfache Agenten. Müssen in Anwendungen schnell schwerwiegende Entscheidungen getroffen, sind Agenten ungeeignet.

Zur Ausführung seiner Aufgaben benötigt der Agent einen gewissen Grad an Intelligenz. Damit werden auch in diesem Gebiet Ansätze der KI benötigt. Intelligente Agenten befinden sich erst am Anfang ihrer Entwicklung. Weltweit sind jedoch große Anstrengungen in Forschung und Praxis zu beobachten (vgl. [Bren+98] S. 20).

Ein weiterer Trend geht in Richtung der erkennungsbasierten Benutzungsschnittstellen (vgl. [Prei99] S. 483 ff). Diese ermöglichen es dem Benutzer über Sprache, Gestik und Schrift zu kommunizieren. Das Problem für die Schnittstelle liegt darin, dass diese auf zwischenmenschlicher Kommunikation basierende Interaktionsformen mehrdeutig sein können. Erschwerend kommt die Individualität jedes Menschen dazu. Aus diesem Grund sind heutige Systeme meist mit einer kurzen Trainingsphase verknüpft, in der das System die benutzerspezifischen Charakteristika erlernt. Diese werden bei der Interpretation der Benutzereingabe berücksichtigt. Um möglichst viele Mehrdeutigkeiten zu erkennen, wird während der Interpretation der aktuelle Kontext beachtet. Bei Spracherkennung werden entsprechende Grammatiken eingesetzt, um bei ähnlich klingenden Worten das richtige herauszufinden. Damit erreichen gute Spracherkennungsprogramme eine Erkennungsrate von 90 bis 98 Prozent. Diese sind jedoch benutzerabhängig. Vom Benutzer unabhängige Systeme erreichen maximal 70 Prozent, womit sie praktisch unbrauchbar sind. Eine Erhöhung der

Genauigkeit ist durch eine Verkleinerung des Sprachwortschatzes erreichbar. Damit eröffnen sich eine große Anzahl von Anwendungsgebieten. So kann beispielsweise das Telefon oder Radio im Auto durchaus mit Sprache gesteuert werden, da der benötigte Wortschatz klein ist. Dieser wird zusätzlich durch den Kontext eingeschränkt. Beispielsweise macht der Befehl *Auflegen* beim Telefon keinen Sinn, wenn keine Verbindung besteht. Der Benutzer wird in diesem Fall *Verbindung aufnehmen* gesagt haben.

Mit der zunehmenden Verbreitung der Handheld PCs nimmt die Bedeutung der Handschriftenerkennung wieder zu. Für diese sehr kleinen Computer mit ihren begrenzten Ressourcen ist eine Spracherkennung zu anspruchsvoll. Ein Tastatur ist ebenfalls ungünstig, da sie den Vorteil der geringen Größe gegenüber normalen Computern beseitigen würde. Die Erkennung der mit einem speziellen Stift geschriebenen Zeichen funktioniert sehr gut. Die Eingaben müssen allerdings mit einem speziellen Alphabet erfolgen. In diesem besteht jeder Buchstabe aus genau einem nicht abgesetzten Strich, was die Erkennung für das System erheblich vereinfacht. Das Alphabet ist jedoch leicht zu lernen und ermöglicht sogar eine schnellere Eingabe als die sonst übliche Schreibschrift.

Auch auf dem Gebiet der Gestenerkennung wurden in den letzten Jahren Fortschritte erzielt. Dabei können die Gesten durch Videokameras oder Datenhandschuhe übermittelt werden. Während die Erkennung mittels dem Datenhandschuh relativ einfach ist, gestaltet sich die Videoerkennung als schwieriger. Der Vorteil ist, dass nicht nur Gesten mit der Hand, sondern auch mit dem Kopf interpretiert werden können. Probleme in diesem Bereich treten bei wechselnden Beleuchtungen und Farben auf. Die bereits gut funktionierenden Datenhandschuhe werden vor allem in dem Bereich der Virtual Reality genutzt. Dabei kann der Benutzer durch eine 3D-Brille räumlich dargestellte Objekte direkt manipulieren (vgl. [Eckg95] S. 3). Genutzt wird diese Technik beispielsweise, um den Umgang mit teuren Maschinen wie Flugzeugen zu trainieren. Auch im chirurgischen Umfeld befinden sich Systeme in Erprobung. Damit können komplizierte Operationen simuliert werden. Ein weiteres Einsatzgebiet ist der Unterhaltungssektor, der mit einem Jahresumsatz von ca. 2 Mrd. DM einen interessanten Markt darstellt (vgl. [Verb00]). Gesten oder Spracheingabe ermöglichen beispielsweise auch für behinderte Menschen die Computernutzung.

Computer halten in allen Bereichen des Lebens Einzug. Sie werden vermehrt im Alltag anzutreffen sein. So können beispielsweise Waschmaschinen, Radios Fernseher usw. einen Computer beinhalten, der die Steuerung übernimmt. Für diese Anwendungsgebiete müssen entsprechende Benutzungsschnittstellen geschaffen werden, die sich von den heutigen

deutlich unterscheiden können. Das Ziel ist, einen Schnittstelle zu entwickeln, die vom Benutzer nicht bewusst wahrgenommen wird. Es ist fraglich, ob die heute anzutreffenden Fenstersysteme in diesen Bereichen die gebrauchstaugliche Benutzungsschnittstellen charakterisieren.

Abbildungsverzeichnis

1 Architektur der menschlichen Kognition (vgl. [Glas94] S. 10) 7
2 Die wichtigsten Gestaltgesetze 8
3 Beispiel für subjektive Konturen 9
4 Modell des Antwortverhalten von Computersystemen (vgl. [Shne98] S. 353) 35
5 Dialog einer einfachen Möglichkeit der Individualisierung im Prototyp 46
6 Mehrstufige Filterung der verschiedenen Style Guides 48
7 Darstellung des Layout der Button in einem Fenster 56
8 Baumdarstellung von verschiedenartigen Objekten 58
9 Dimensionen des Prototyping (vgl. [Niel93] S. 94) 64
10 Schichtenarchitektur von UI-Werkzeugen (vgl. [VoNe98] S.17) 68
11 Typeneditor 79

Abkürzungsverzeichnis

AWT	Abstract Window Toolkit
BS	Benutzungsschnittstelle
CAD	Computer-aided Design
DIN	Deutsche Industrie Norm
DV	Datenverarbeitung
EN	Euro-Norm
GME	Generischer Modelleditor
GUI	Graphical User Interface
HCI	Human-Computer-Interaction
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JDK	Java Development Kit
JFC	Java Foundation Class
JSP	Java Server Pages
JVM	Java Virtual Machine
KI	Künstliche Intelligenz
MCI	Mensch-Computer-Interaktion
MS	Microsoft
MB	Megabyte
TÜV	Technischer Überwachungs-Verein
UI	User Interface
UIMS	User Interface Management Systeme
WWW	World Wide Web
WYSIWYG	What you see ist what you get

Literaturverzeichnis

[Alle97] ALLEN, R. B.: Mental Models and User Models; in: HELANDER M. G.; LANDAUER T. K.; PRABHU P. V.: Handbook of Human-Computer Interaction, 1997, S. 49-63

[Bren+98] BRENNER W.; ZARNEKOW R.; WITTIG H.: Intelligente Softwareagenten, Grundlagen und Anwendungen, Springer-Verlag Berlin Heidelberg New York, 1998

[BrRu94] BRODBECK F. C.; RUPIETTA W.: Fehlermanagement und Hilfesysteme, in: EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994, S.: 197-234

[Burm+00] BURMESTER M.; HASSENZAHL M.; MACHATE J.: ISO13407: Empfehlungen für einen benutzerzentrierten Entwicklungsprozess – User im Mittelpunkt, in: Java Magazin 11/2000

[Buzb98] BUZBEE, A.: Virtual Product Development, http://www.uniforum.org.nz/conferences/1998/papers/vpdi/sld016.htm, Download: 16.12.2000, 1998

[Dzid94] DZIDA W.: Qualitätssicherung durch software-ergonomische Normen, in: EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994, S.: 373-406

[Eber94] EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994

[Eckg95] ECKGOLD F.: Virtual Reality, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1995

[FeSi98] FERSTL, O. K.; SINZ, E. J.: Grundlagen der Wirtschaftsinformatik Band 1, 3. Auflage, Oldenburg Verlag, München, 1998

[GEAE97] GESELLSCHAFT ARBEIT UND ERGONOMIE - ONLINE E.V.: Verordnung über Sicherheit und Gesundheitsschutz bei der Arbeit an Bildschirmgeräten, http://www.sozialnetz-hessen.de/ergo-online/Recht/BildschArbV/bildscharbv.htm, 22.12.97, Download: 12.03.2001

[GeHa98] GEIS T.; HARTWIG R.: Auf die Finger geschaut – Neue ISO-Norm für benutzergerechte interaktive Systeme, in: c´t magazin, 14/98, Heise-Verlag, S. 168-171

[Geis90] GEISER, G.: Mensch-Maschine-Kommunikation, R. Oldenburg Verlag GmbH, München 1990

[Glas94] GLASER W. R.: Menschliche Informationsverarbeitung, in: EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994, S.: 7-52

[GöMa00] GÖRNER C.; MACHATE J.: Vom Umgang mit Java Look and Feel Design Guidelines – Do´s and Don´ts, in: Java Magazin 11/2000

[Goul+97] GOULD J. D.; Boies S. J.; Ukelson J.: How To Design Usable Systems; in: HELANDER M. G.; LANDAUER T. K.; PRABHU P. V.: Handbook of Human-Computer Interaction, 1997, S. 231-253

[Grei00a] Greiffenberg, S.: Projekt GME 2001, http://wiseweb.wiwi.tu-dresden.de/gme, Download 27.02.2001

[Grei00b] Greiffenberg S.: Anforderungsspezifikation GME 2001, 7. Juni 2000

[GrEs01] Greiffenberg S.; Esswein W.: Stand der Entwicklung einer Methode zur Metamodellierung – Arbeitsbericht, 2001

[HaTh90] HARRISON M.; THIMBLEBY H.: Formal Methods in Human-Computer Interaction, Cambridge University Press, 1990

[Herc94] HERCZEG, M.: Software-Ergonomie: Grundlagen der Mensch-Computer-Kommunikation, Addison-Wesley, 1. Auflage, 1994

[HiHa93] HIX D.; HARTSON H. R.: Developing user interfaces: Ensuring Usability Through Product & Process, John Wiley & Sons, Inc. 1993

[HoHi97] HOUDE S.; HILL C.: What do Prototypes Prototype? in: HELANDER M. G.; LANDAUER T. K.; PRABHU P. V.: Handbook of Human-Computer Interaction, 1997, S. 367-382

[ISO10075] EN ISO 10075-1: Ergonomische Grundlagen bezüglich psychischer Arbeitsbelastung. Teil 1: Allgemeines und Begriffe, 2000

[ISO11581-1] DIN ISO/IEC 11581-1 (Norm-Entwurf): Informationstechnologie – Benutzersystemschnittstellen und Symbole – Darstellungen und Funktionen von Icons. Teil 1: Icons – Allgemeines, 1995

[ISO13407] EN ISO 13407: Benutzerorientierte Gestaltung interaktiver Systeme, 1999

[ISO14598-1] ISO/IEC 14598-1: Information technology – Software product evaluation. Teil 1: General Overview, 1999

[ISO9126-1] ISO/IEC FDIS 9126-1: Information technology – Software product quality. Teil 1: Quality model, 2000

[ISO9241-10] EN ISO 9241-10: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 10: Grundsätze der Dialoggestaltung, 1996

[ISO9241-11] EN ISO 9241-11: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 11: Anforderungen an die Gebrauchstauglichkeit – Leitsätze, 1998

[ISO9241-12] EN ISO 9241-12: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 12: Informationsdarstellung, 1998

[ISO9241-13] EN ISO 9241-11: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 13: Benutzerführung, 1998

[ISO9241-14] EN ISO 9241-10: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 14: Dialogführung mittels Menüs, 1999

[ISO9241-16] EN ISO 9241-10: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Teil 16: Dialogführung mittels direkter Manipulation, 1999

[Java01] SUN MICROSYSTEMS: Java™ Look and Feel Design Guidelines, http://java.sun.com/products/jlf/guidelines.html, Download 24.11.2000

[Joha93] JOHANNSEN G.: Mensch-Maschine-Systeme, Springer-Verlag Berlin Heidelberg New York, 1993

[Kens+95] Kensik A.; Prümper J.; Frese M.: Ergonomische Gestaltung von Software auf Grundlage handlungsorientierter Fehleranalysen, in: Böcker H. (Hrsg): Software-Ergonomie ´95, B. G. Teubner Stuttgart, 1995, S. 217-232

[Komm+98] KOMMERS, P. A. M.; FERREIRA, A. F.; KWAK, A. W.: Document Management for Hypermedia Design, Springer Verlag Berlin Heidelberg 1998

[Krus99] KRUSCHINSKI V.: Layoutgestaltung grafischer Benutzungsoberflächen: Generierung aus OOA-Modellen, Spektrum Akademischer Verlag Heidelberg Berlin, 1999

[NeCa97] NEALE D. C.; CARROLL J. M.: The Role of Metaphors in User Interface Design; in: HELANDER M. G.; LANDAUER T. K.; PRABHU P. V.: Handbook of Human-Computer Interaction, 1997, S. 441-462

[Niel93] NIELSEN, J.: Usability Engineering, Academic Press, Inc., 1993

[Ober94] OBERQUELLE H.: Formen der Mensch-Computer-Interaktion, in: EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994, S.: 95-143

[Olse98] OLSEN D. R.: Developing user interfaces, Morgan Kaufmann Publishers, Inc., 1998

[Oppe94] OPPERMANN R.: Individualisierung von Benutzungsschnittstellen, in: EBERLEH E.; OBERQUELLE H.; OPPERMANN R. (Hrsg.): Einführung in die Software-Ergonomie, Walter de Gryter Berlin New York, 1994, S.: 235-269

[Prei99] PREIM, B.: Entwicklung interaktiver Systeme, Grundlagen, Fallbeispiele und innovative Anwendungsfelder, Springer-Verlag Berlin Heidelberg New York 1999

[ReMo95] REDMOND-PYLE, D.; MORRE, A.: Graphial user interface design and evaluation (guide): a practical process, Prentice Hall International (UK) Limited, 1995

[Shne98] SHNEIDERMAN B.: Designing the User Interface: strateies for effective human-computer-interaction, Third Edition, Addison Wesley Longman, Inc., 1998

[Stau87] STAUFER M.J.: Piktogramme für Computer: kognitive Verarbeitung, Methoden zur Produktion u. Evaluation, Walter de Gruyter & Co. Berlin, 1987

[Treu94] TREU, S.: User Interface Design: A Structured Approach, Plenum Press New York, 1994

[Verb00] VERBAND DER UNTERHALTUNGSSOFTWARE DEUTSCHLAND E.V.: Der Markt der Unterhaltungssoftware 2000, http://www.vud.de/marktforschung/inhalt.php3, Download: 17.03.2001

[VoNe98] VOSS J.; NENTWIG D.: Entwicklung von graphischen Benutzungsschnittstellen: Modelle, Techniken und Werkzeuge der User-Interface-Gestaltung, Carl Hanser Verlag München Wien, 1998

[ZeZe92] ZEIDLER, A.; ZELLNER R.: Software-Ergonomie: Techniken der Dialoggestaltung, Oldenburg 1992

Anhang

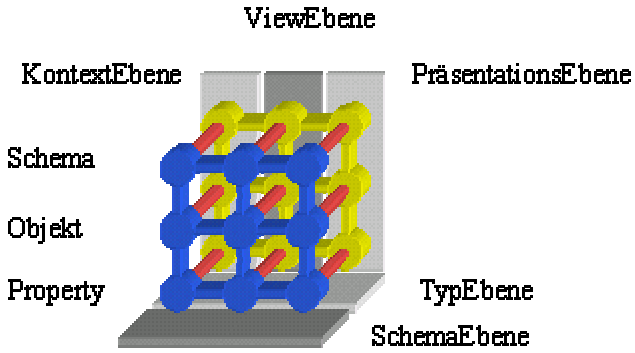
A Ergänzende Abbildungen

A.1 Vergleich der Entwicklungsumgebungen anhand einer Checkliste

Produkt		**Forte for Java, Community Edition 1.0**	**JBuilder 4.0 Foundation**	**VisualCafé 4.1 Standard Edition**	**VisualAge for Java, Entry Edition 3.0**
Hersteller		Sun	Borland/Inprise	WebGain	IBM
Internet		www.forte.com	www.borland.com/jbuilder	www.visualcafe.com	www.software.ibm.com/ad/vajava
OS	Solaris	ja	ja	nein	ja
	Windows	ja	ja	ja	ja
	Linux	ja	ja	nein	ja
Anforderung (Hersteller)	CPU (MHz)	Pentium 133	Pentium II 233	Pentium 133	-
	Speicher (MB)	64	128	128	64
	Festplatte (MB)	30	115	300	100
	Geschwindigkeit (1..10)	3	7	8	7
Funktionen	GUI Builder	integriert	integriert	integriert	integriert
	Swing Klassen	ja	ja	ja	ja
	Debugger	integriert	integriert	integriert	integriert
	distributed Debugger	-	Enterprise Edition	Enterprise Edition	ja
	Teamunterstützung	nein	Enterprise Edition	ab Expert Edition	Enterprise Edition
	Java 2	ja	ja	ja	ja
	Versionskontrolle	Enterprise Edition	Enterprise Edition	ab Expert Edition	ja
	JavaDoc	integriert	integriert	integriert	integriert
	JDK unabhängig	ja	ja	nein	nein
	Bedienung	6	8	9	8
Hilfesystem	Tutorial	ja	ja	ja	ja
	Wizards	nein, Templates	ja	nein, Templates	ja
	Hilfe	ja	ja	ja	ja
	Sprache	englisch	englisch	englisch	englisch
	localizition tools	-	ja	ab Expert E.	ja
	Bewertung der Hilfe (1..10)	8	4	7	7

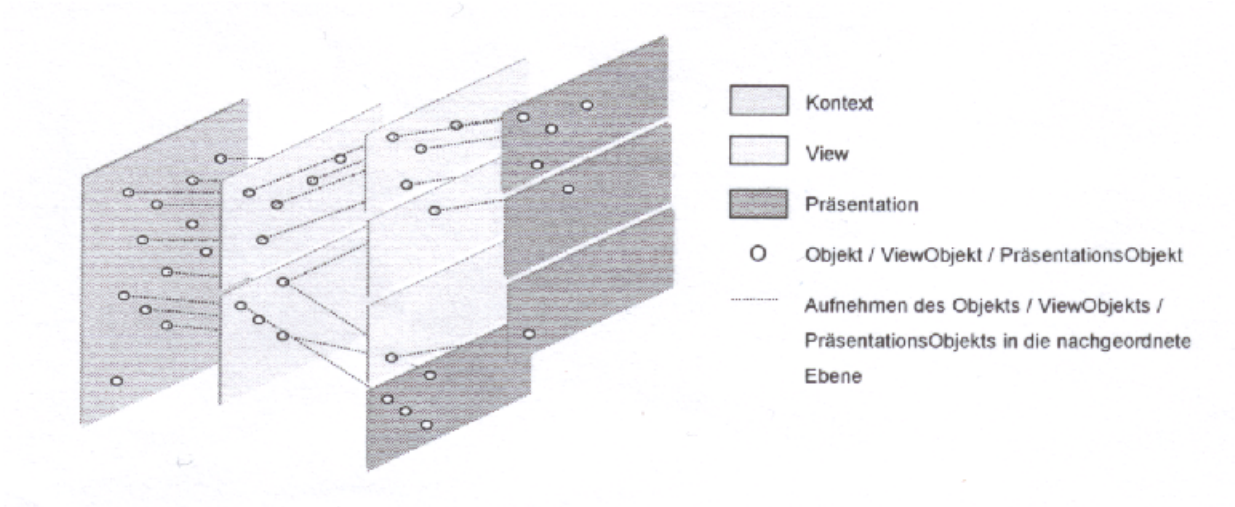
Testsystem: AMD Duron 700 MHz Prozessor, 98 MB Speicher und MS Windows NT 4.0

A.2 Das E^3-Modell



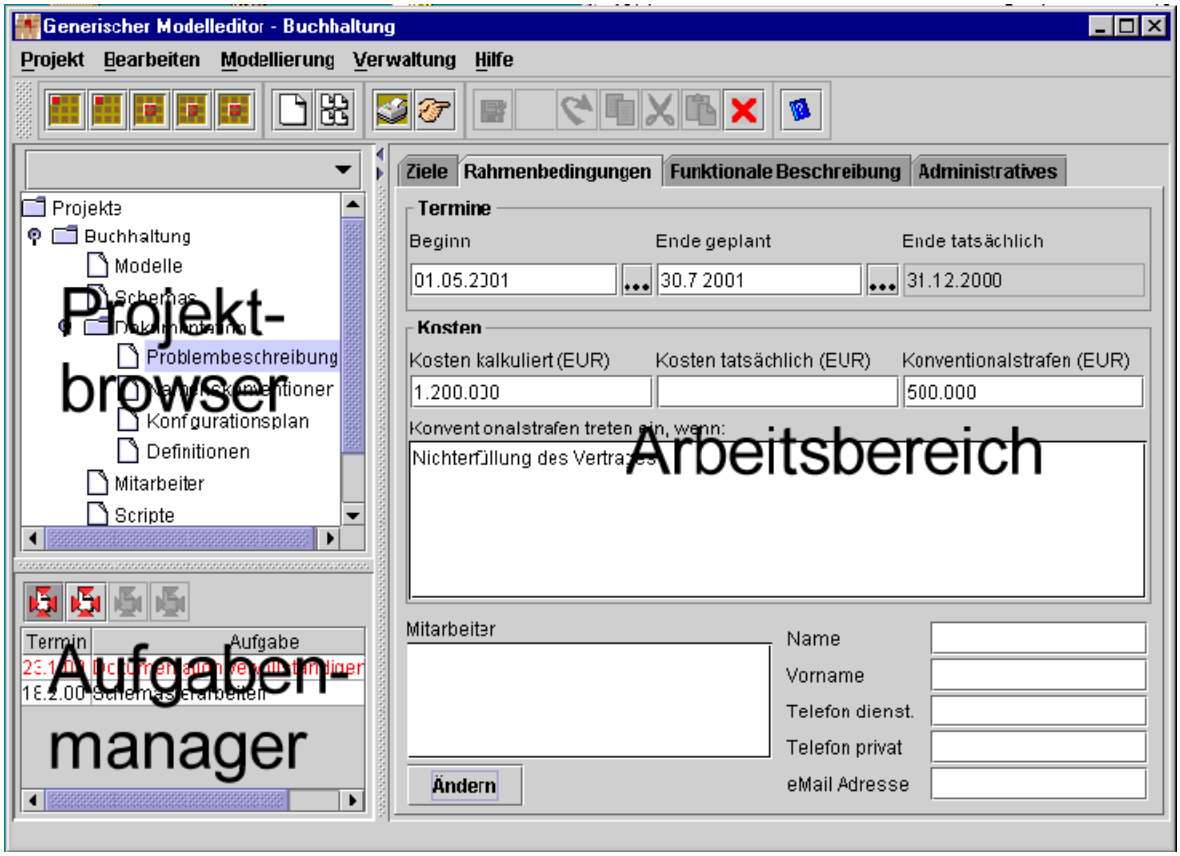
A.3 Mehrstufige Sichtenbildung im E^3-Modell

Umfangreiche Methoden machen eine mehrstufige Sichtenbildung notwendig (vgl. [GrEs01] S. 11f). Dabei wird eine weitere Sicht auf eine bereits vorhandene definiert. Die folgende Abbildung stellt solche teilweise nachgeordneten Views dar.



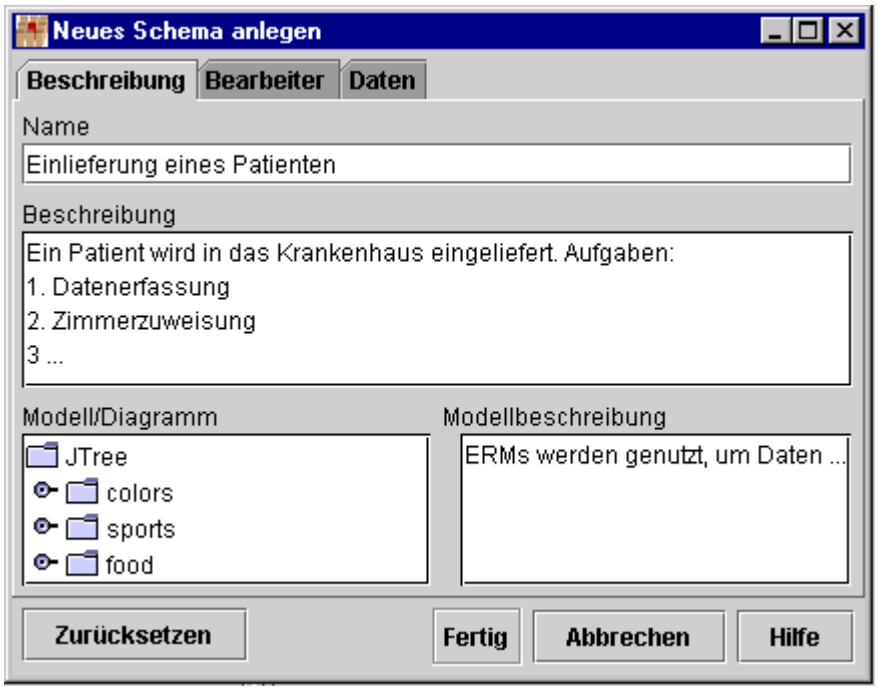
A.4 Abbildung des Workspace

Der Workspace ist in die drei Bereiche Projektbrowser, Arbeitsbereich und Aufgabenmanager gegliedert.

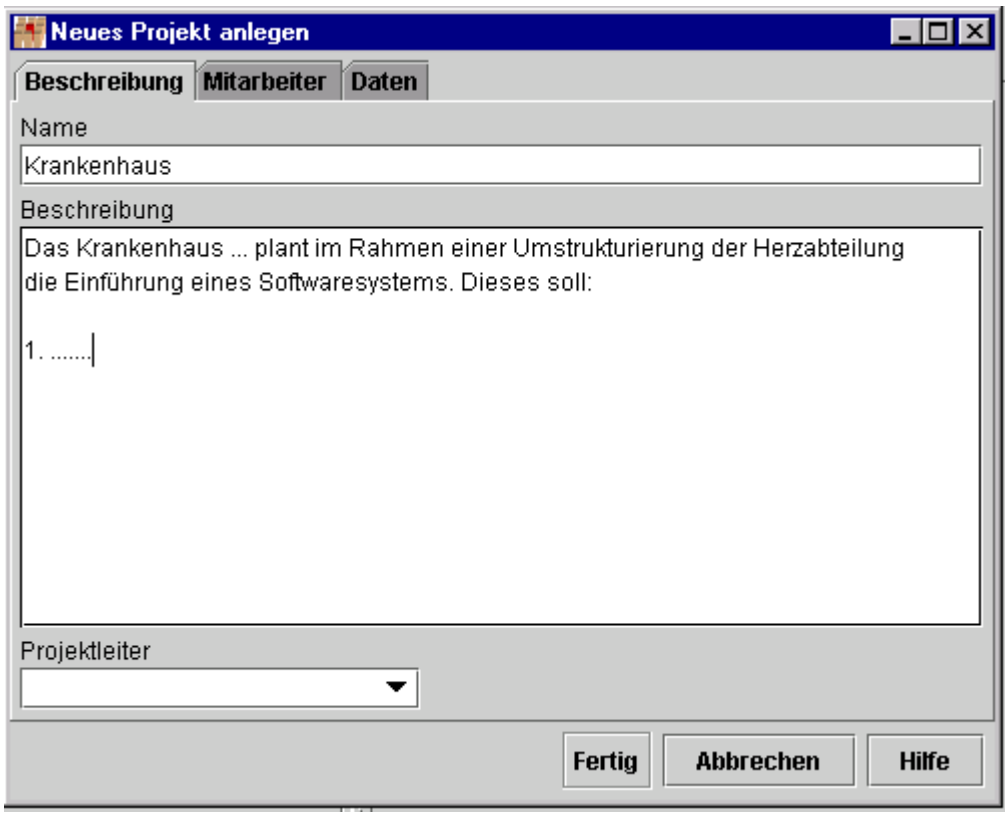


A.5 Wizards im Prototyp

A.5.1 Unterstützung beim Anlegen eines neuen Schemas



A.5.2 Unterstützung beim Anlegen eines neuen Projektes



A.6 Fehlersystem im Prototyp

Die folgende Abbildung zeigt eine Fehlermeldung. Der Benutzer kann weitere Hinweise über den Schalter *Hilfe* abfordern. Bei nicht kritischen Fehlern kann die Meldung ignoriert und der aktuelle Prozess fortgesetzt werden. Auf der Karte „Service-Kontakt“ sind sowohl die Adresse eines verantwortlichen Mitarbeiters, dem Systemadministrator, als auch die Adresse der Hotline.

A.7 Hilfesystem im Prototyp

Die folgende Abbildung zeigt ein Fenster des Hilfesystems. Dieses kann auch genutzt werden, um aufgetretene Fehler näher zu spezifizieren. Wie im Abschnitt 3.4.6 gefordert, soll die Hilfe in einem separaten Fenster dargestellt werden. Die Funktion des Hilfebrowsers sollte möglicht einfach sein. Aus diesem Grund wurden auch nur die Funktionen *Anfangsseite*, *Zurück*, *Vorwärts* und *Drucken* zur Verfügung. Es ist zu überlegen, ob die Funktionalität von *Lesezeichen* noch hinzugefügt werden soll. Damit wird dem Benutzer die Möglichkeit gegeben, eigene Kommentare anzufügen.

Generischer Modelleditor - Onlinehilfe
Fehler: Ausdruck nicht möglich
Dieser Fehler weißt darauf hin, daß es Probleme mit dem eingestellten Drucker gibt. Ist ihr Drucker
lokal oder im Netzwerk
angeschlossen?
Lokaler Drucker
Prüfen Sie:
Druckerkabel korrekt verbunden
Drucker angeschaltet
Drucker hat genügend Papier
Schließen

A.8 Design von Benutzungsschnittstellen

Die folgende Grafik stellt den von der ISO geforderten benutzerorientierten Gestaltungsansatz für interaktive Softwaresysteme dar (vgl. [ISO13407] S. 6).

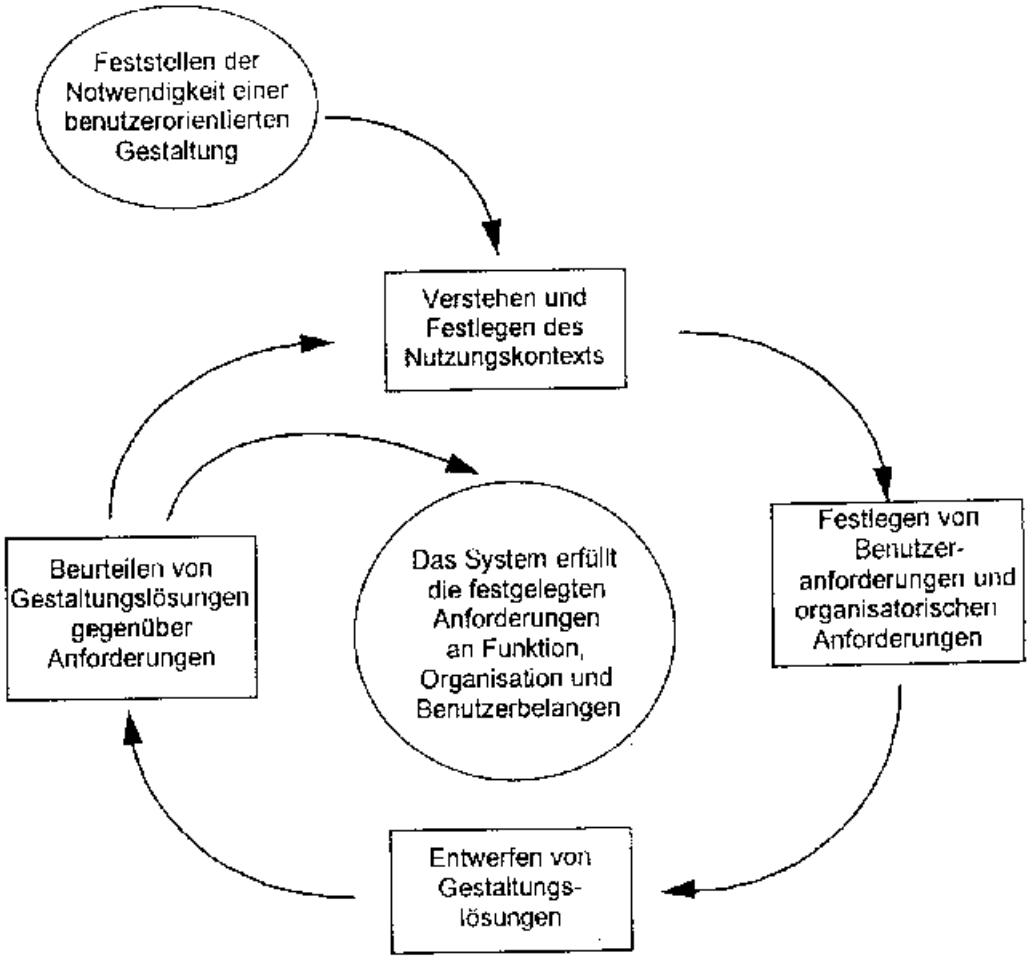
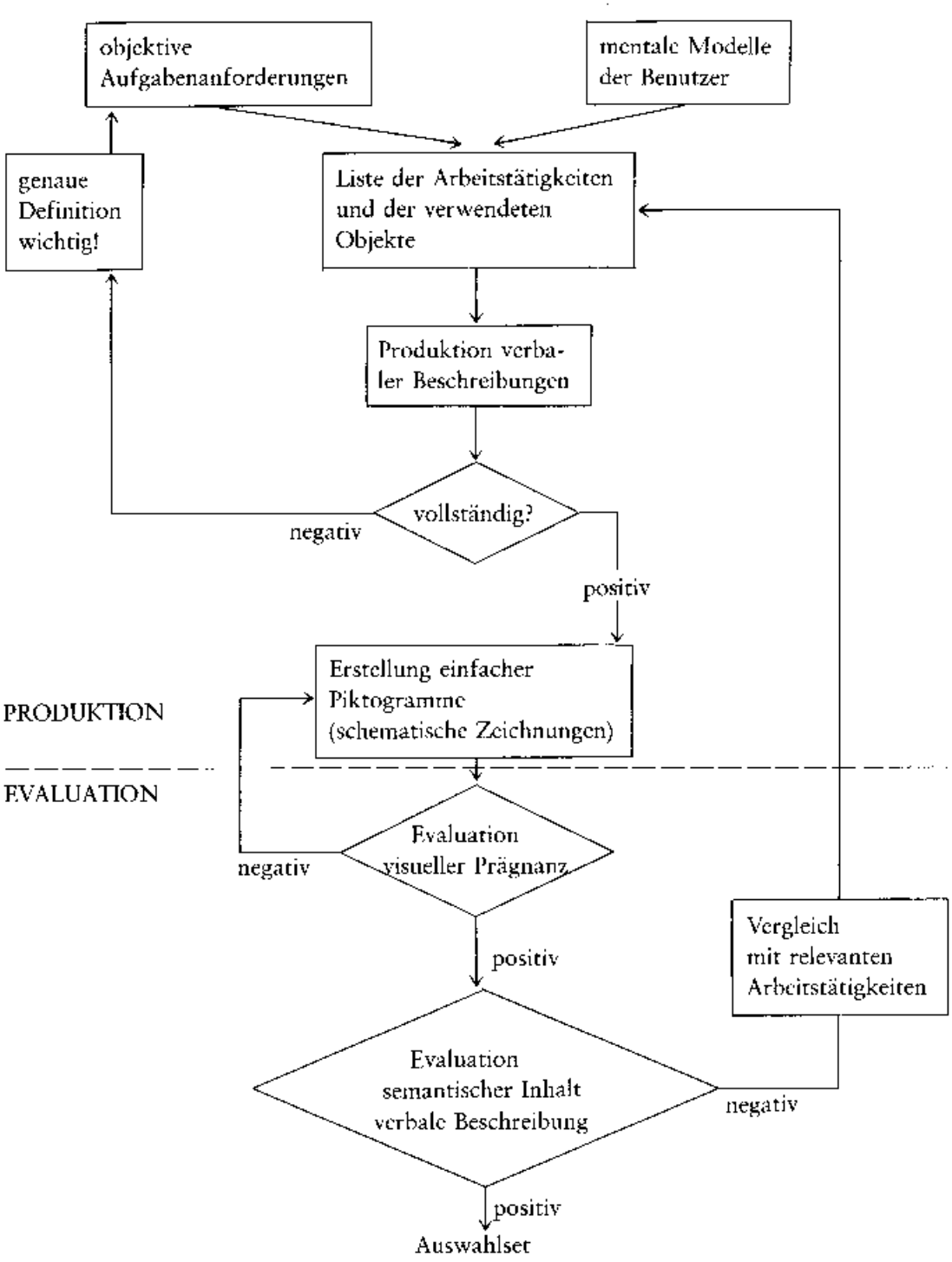


Bild 1: Wechselseitige Abhängigkeit benutzer-orientierter Gestaltungsaktivitäten

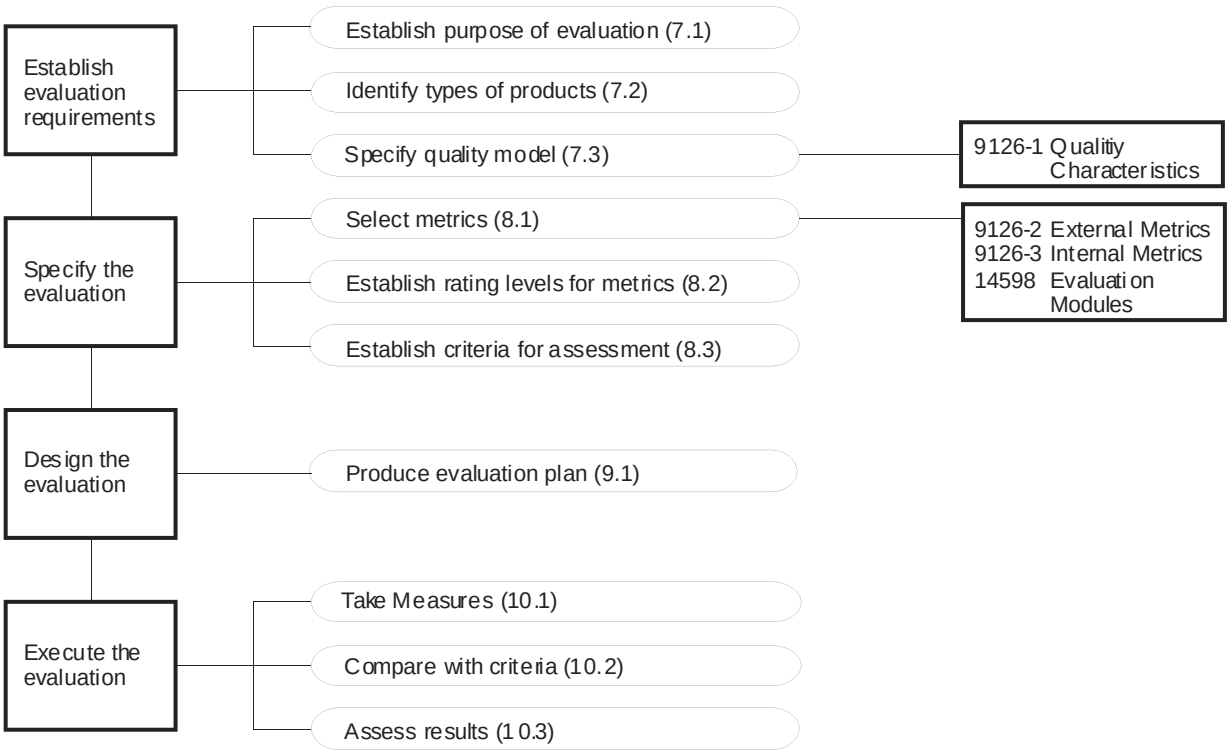
A.9 Gesamtprozeß der Piktogrammentwicklung

Die folgende Abbildung stellt den Prozess der Produktion und Evaluation von Piktogrammen dar (vgl. [Stau87] S. 133). Dabei wird besonders auf die Phase der Evaluation hingewiesen.



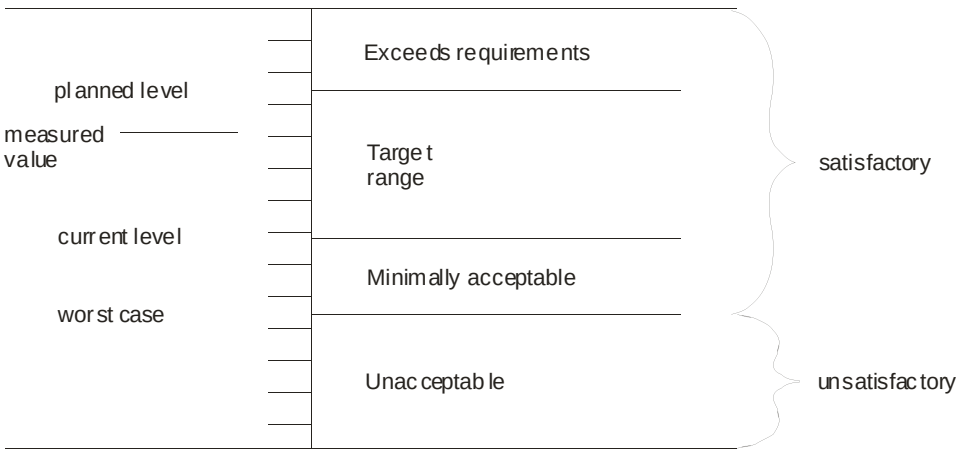
A.10 Evaluierungsprozeß von Software

Die folgende Grafik illustriert den Evaluierungsprozeß von Software (vgl. [ISO14598-1] S. 9).



A.11 Rating Level

Die Skala dient zur Einordnung von Kriterien bei der Evaluierung von Software (vgl. [ISO14598-1] S. 16).



A.12 Qualitätsmodell für Benutzungsschnittstellen (grob)

Das folgende Bild zeigt ein Qualitätsmodell das auf Benutzbarkeit ausgerichtet ist (vgl. [ISO9126-1] S. 12). Die vier einzelnen Charakteristika müssen noch durch messbare Attribute quantifizierbar gemacht werden.

Figure 5 - Quality model for quality in use

B Checklisten

B.1 Evaluierung von Fenstern

Für jedes Fenster müssen die folgenden Checklisten durchgearbeitet werden. Die Checklisten für Panele und Bildschirmelemente müssen jeweils für alle im Fenster befindlichen Elemente durchgearbeitet werden. Werden alle Punkte mit „ja“ beantwortet, sind die wichtigsten Aspekte des ergonomische Design beachtet worden. Jede Frage die mit „nein“ beantwortet wurde, muss für diesen speziellen Fall diskutiert werden. Auch wenn alle Fragen korrekt durchgearbeitet wurden, können ergonomische Schwächen vorliegen. Aus diesem Grund können die Checklisten nur unterstützend wirken. GÖRNER und MACHETE haben in ihrem Beitrag eine GUI-Checkliste entwickelt (vgl. [GöMa00] S. 48), die ebenfalls genutzt werden sollte.

B.1.1 Überprüfung von Struktur und Layout des GMEJFrame

Frage	Ja	Nein
Wurde ein Fenstertitel vergeben?		
Ist eine Standardgröße beim Öffnen des Fensters vorgegeben?		
Werden wichtige Funktionen als Icon angeboten?		
Werden shortcuts (Beschleunigungstasten) für wichtige Funktionen bereitgestellt?		
Besteht jederzeit die Möglichkeit des Abbruchs?		
Wurde ein Standardbutton festgelegt?		
Ist die Hilfefunktion verfügbar?		
Ist diese schnell und einfach erreichbar (shortcut) ?		
Wurde eine minimale Größe des Fensters festgelegt?		
Werden Bildschirmlaufleisten (scroll bar) beim Verkleinern sichtbar, um nicht sichtbare Bereiche anzeigen zu können?		
Sind die Abstände der Elemente zum Rand eingehalten?		
Werden die Texte in gemischter Schreibweise angezeigt? (Erhöhung der Lesegeschwindigkeit gegenüber reiner Großschreibung um 13 Prozent)		

B.1.2 Überprüfung von Panelen (inkl. JScrollPane und JTabbedPane)

Frage	Ja	Nein
Beträgt der Abstand zu anderen Panelen jeweils 6?		
Wurden minimale Größe angegeben?		
Sind bei Bedarf Bildschirmlaufleisten verfügbar?		
Wurden die Vorgaben aus Abschnitt 4.2.2 beachtet?		

B.1.3 Überprüfung sonstiger Bildschirmelemente

Frage	Ja	Nein
Wurden die definierten Standardfarben verwendet?		
Wurden die definierten Standardgrößen verwendet?		
Wurden für inhaltlich gleiche Elemente in ähnlichen Fenstern auch gleiche Positionen verwendet?		
Wurden sinnvolle Vorgabewerte angegeben?		
Wurde eindeutige Sprache (kein Englischdeutsch Mix) verwendet?		
Wurde das Element, falls nötig, eindeutig und klar verständlich durch einen Bezeichner (`JLabel`) beschrieben?		
Wurden bei Bedarf die Ränder entsprechend den Konventionen des Style Guide gesetzt.		
Wurden Bildschirmlaufleisten bereitgestellt, wenn sich das Element im eigenen Darstellungsbereich ändern kann (z.B. `JTree`, `JList`, `JTable`)		

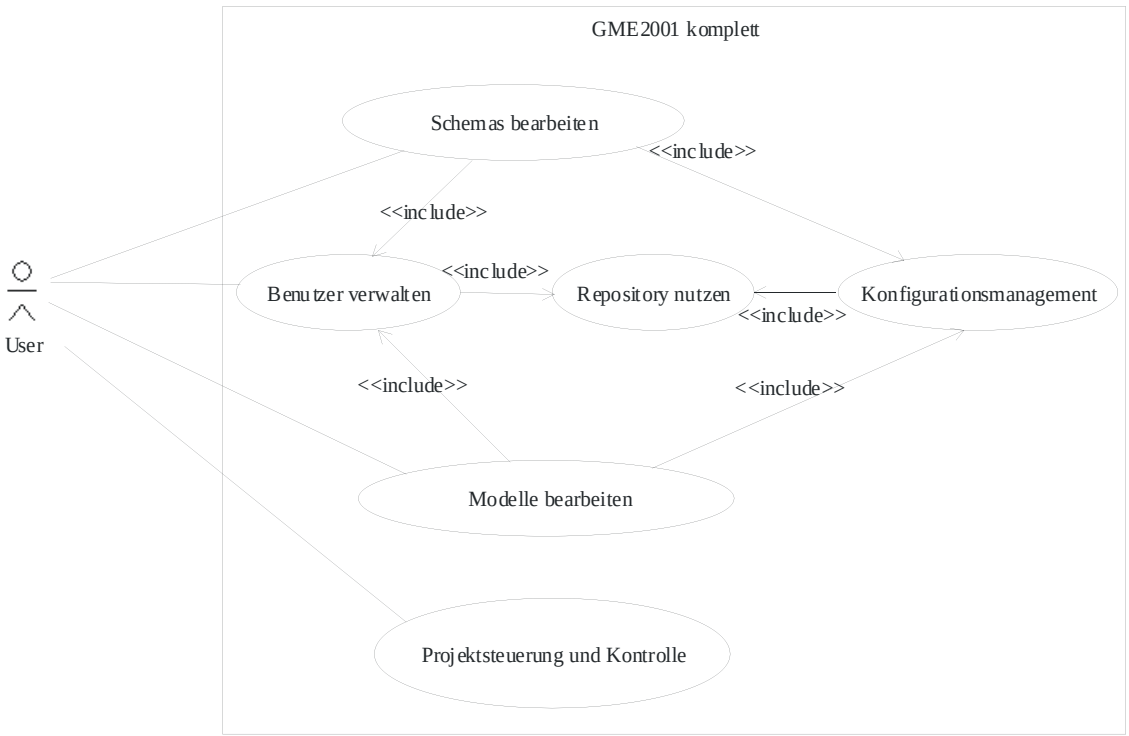
B.1.4 Verhalten von Bildschirmelementen

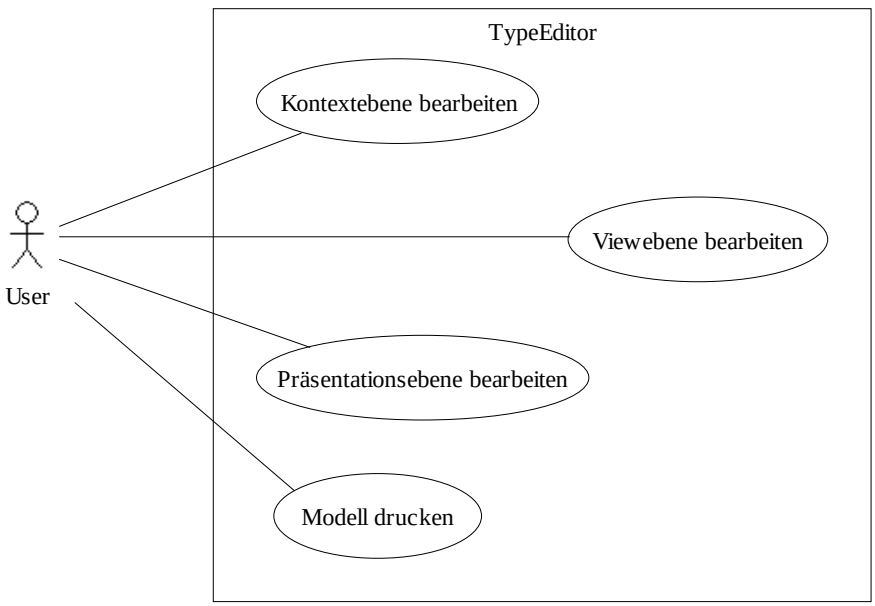
Frage	Ja	Nein
Entspricht das Verhalten des Elementes dem allgemeingültigen Interaktionsverhalten (gleiche Elemente agieren in allen Fenstern auch gleich)?		
Werden bei Listen, Verzeichnisbäumen u.ä. Kontextmenüs angeboten?		
Werden Fehlermeldungen bei Fehleingaben angezeigt?		
Sind Hilfetexte zu Fehlermeldungen abrufbar?		
Ist der Fokus (aktives Bildschirmelement) nach dem Öffnen eines Fensters sinnvoll gesetzt?		
Wird eine Fortschrittsanzeige bei Prozessen, die länger als 3 Sekunden dauern angezeigt (vgl. DIN 9241) ?		
Wird eine Rückmeldung bei erfolgreicher Ausführung einer Option angezeigt?		
Steht der Cursor bzw. die Selektion nach dem Einfügen von Objekten in Listen, Tabellen oder Bäumen auf diesen?		
Wurde das Fenster möglichst nicht-modal (der Benutzer kann in ein anderes Fenster wechseln) implementiert?		

C Spezifikation des Prototyp

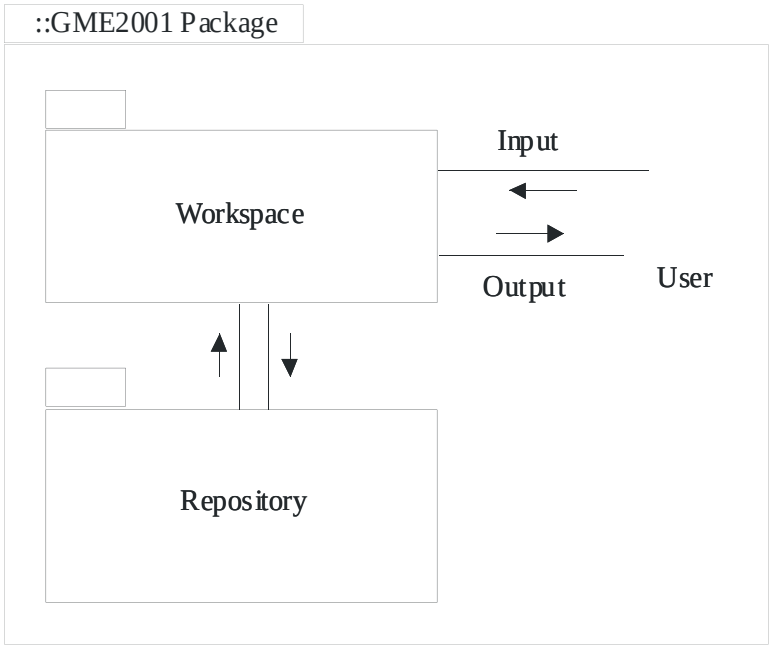
C.1 Statische Sicht

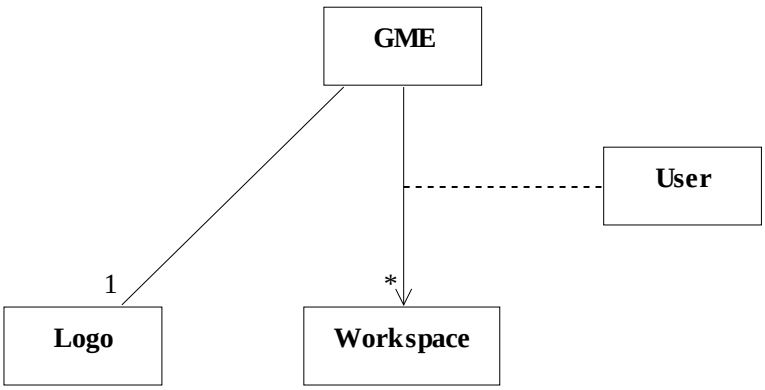
C.1.1 Use Case Diagramme des Prototyp GME 2001

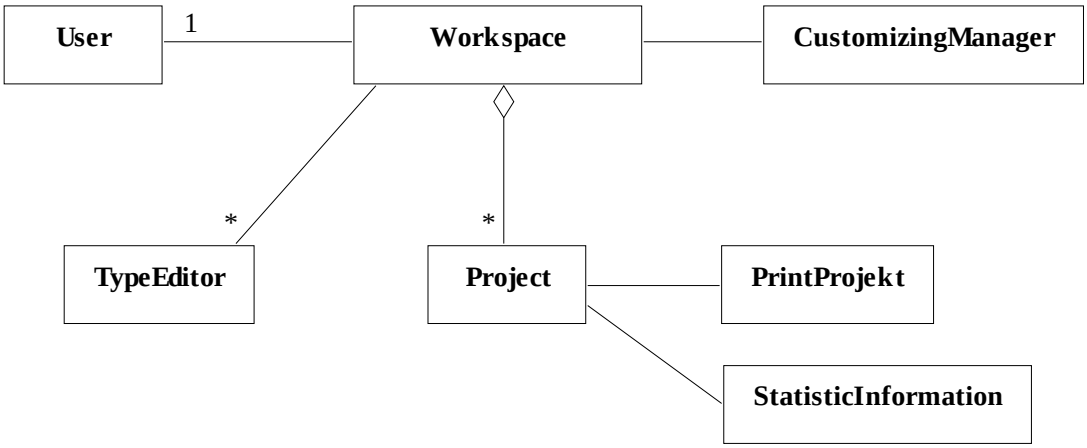


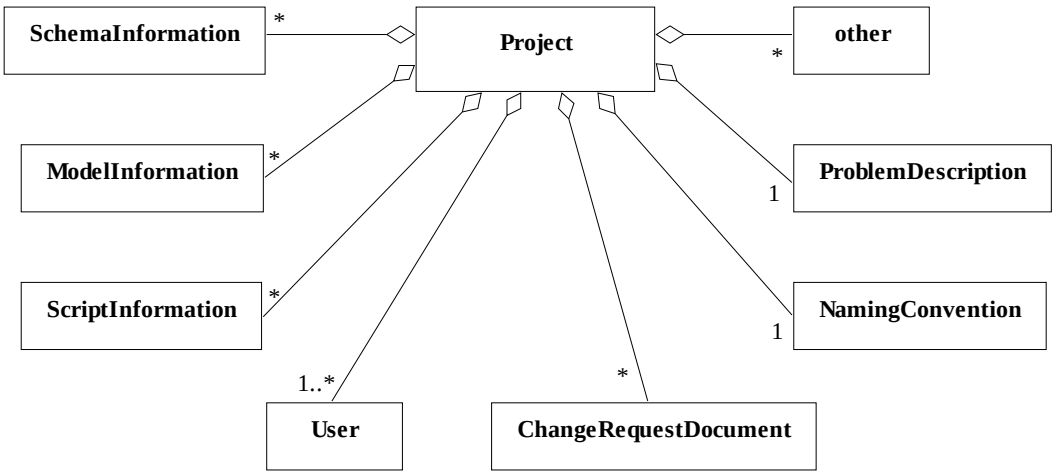


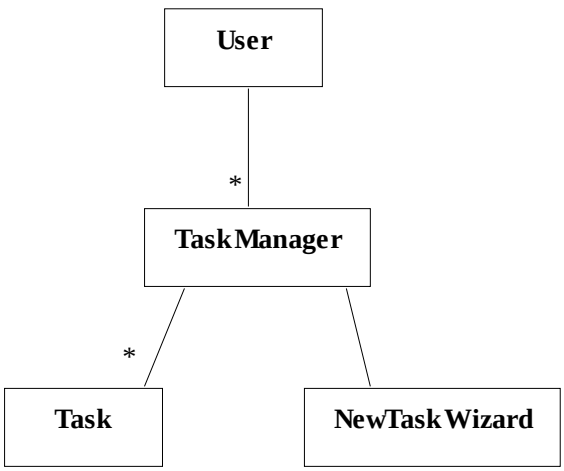
C.1.2 Klassendiagramme des GME Prototyp





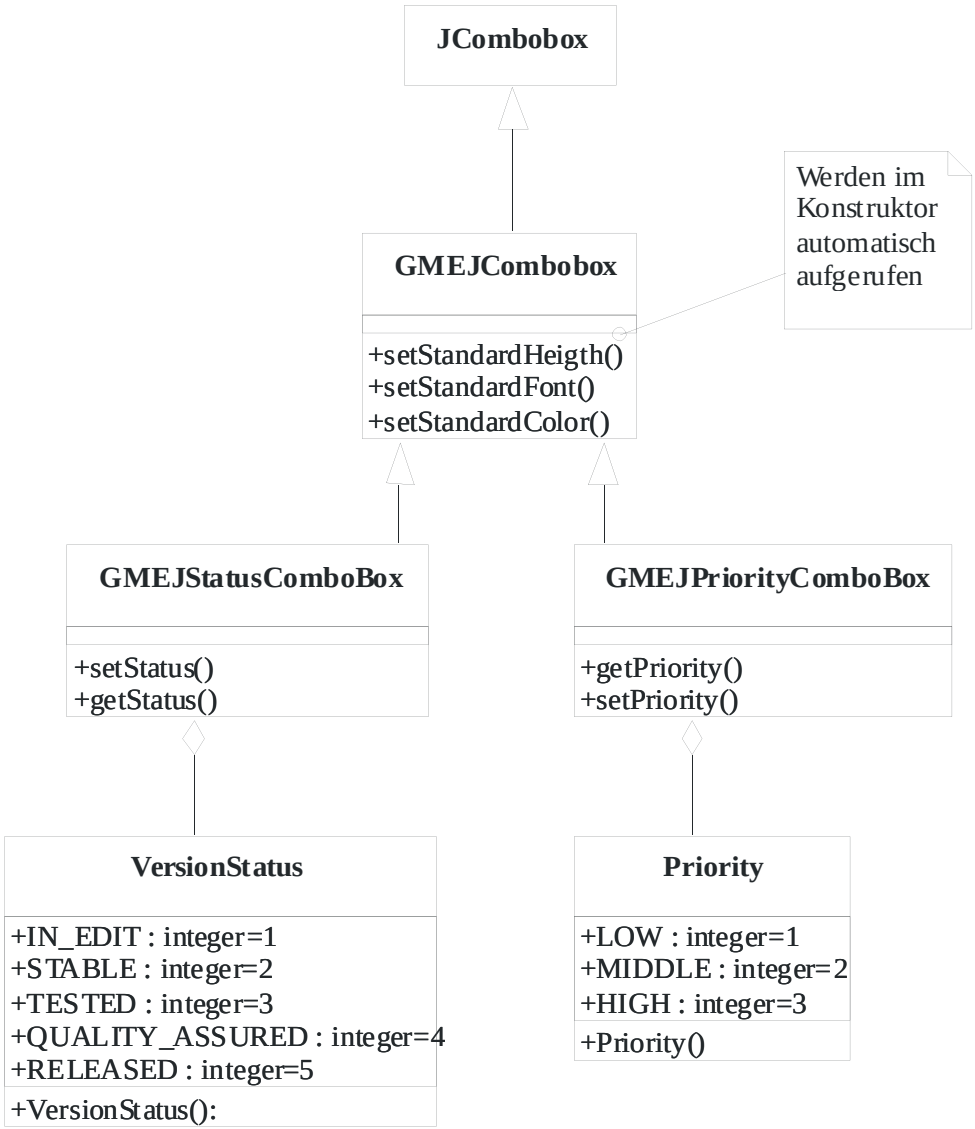

User
1
Workspace
CustomizingManager
*
*
TypeEditor
Project
PrintProjekt
StatisticInformation


SchemaInformation
*
Project
other
*
ModelInformation
*
ProblemDescription
1
ScriptInformation
*
NamingConvention
1
1..*
*
User
ChangeRequestDocument

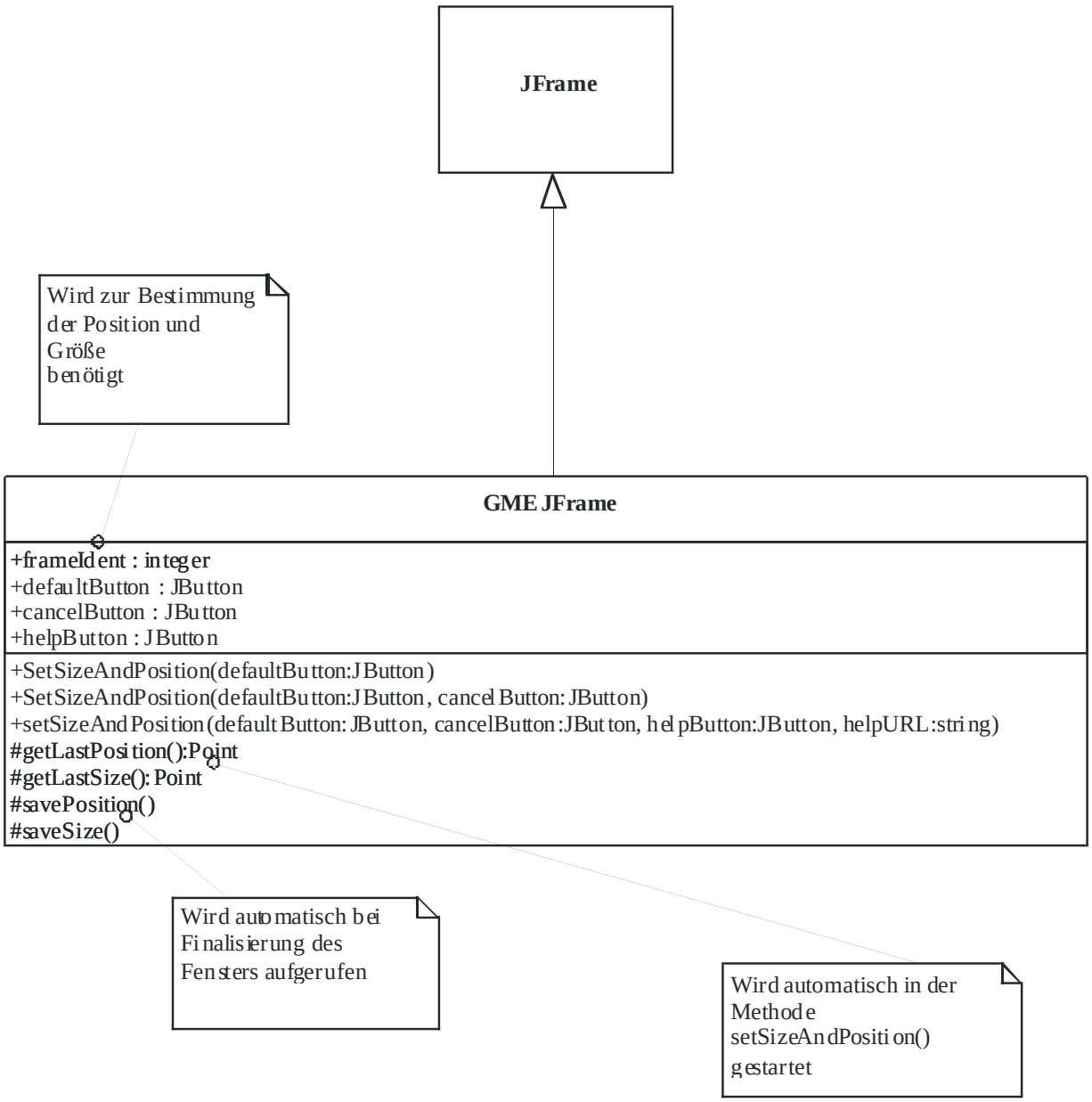

User
*
TaskManager
*
Task
NewTaskWizard

C.1.3 Klassendiagramm einzelner Interaktionselemente

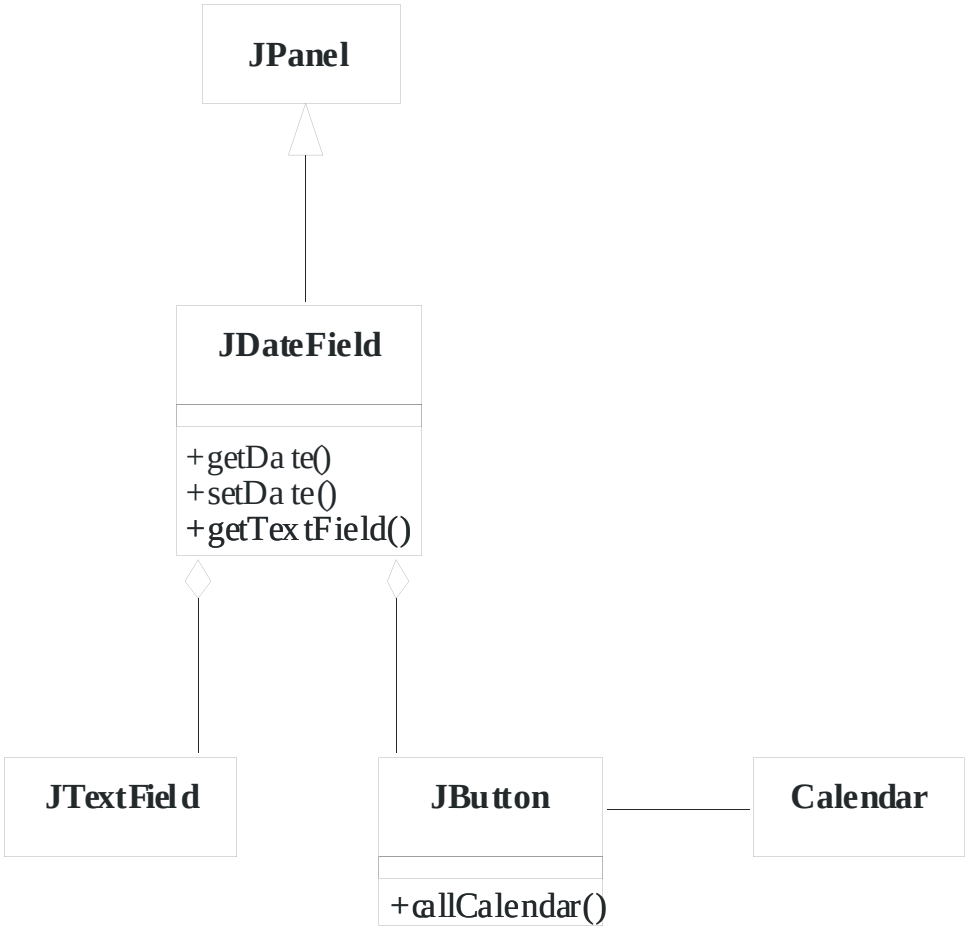
Klasse **GMEJCombobox**



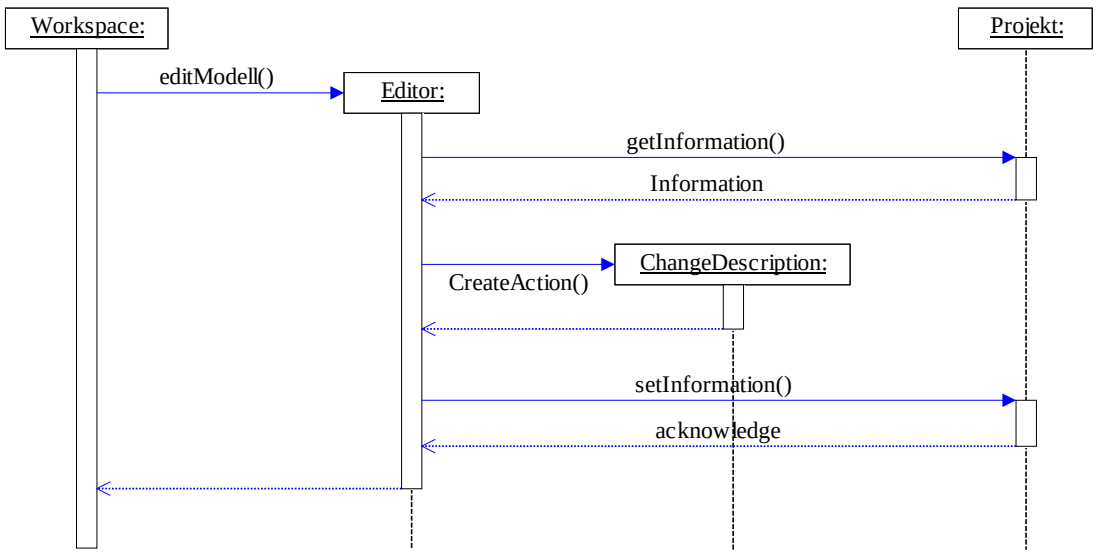
Klassendiagramm **`GMEJFrame`**



Klassendiagramm **`JDateField`**



C.2 Dynamische Sicht



C.3 Detaillierte Beschreibung ausgewählter Klassen

C.3.1 Spezifikation der Klasse `Project`

Attribut	Typ	Beschreibung
createdOn	Datum	Datum, an dem das Projekt erstellt wurde
startetOn	Datum	Datum, an dem die Bearbeitung beginnen soll
state	Integer	Der aktuelle Projektstatus: 1 = initial, 2 = ...,
projectLeader	User	Der Projektleiter.
description	Text	Eine kurze Beschreibung des Projektes. Die genauere Spezifikation wird im Rahmen der Dokumentation durchgeführt.
deadline	Datum	Der Termin des Projektes.
realDeadline	Datum	Der tatsächlich realisierte Termin.
password	Text	Das Passwort des Projektes.
customer	Adresse	Der Auftraggeber des Projektes.
internCustomer	Boolean	Ist der Auftraggeber ein interner oder externer Kunde?
documentation	Documentation	Diese Klasse beinhaltet die gesamte Projektdokumentation. Sie wurde im Rahmen der Diplomarbeit von Christian Kluge spezifiziert.
models	Liste: Model	Enthält die Liste der Modelle.
schemata	Liste: Schema	Enthält die Liste der Schemata.
collaborators	Liste: User	Enthält alle am Projekt beteiligten Mitarbeiter. Die Klasse User ist Teil des Benutzermanagements und des Rechteverwaltungssystems. Sie wird an dieser Stelle nicht weiter spezifiziert.
scripts	Liste: Script	Enthält alle Scripte, die programmiert wurden. Die Klasse Script wird an dieser Stelle nicht weiter spezifiziert.
changeRequests	Liste: ChangeRequest	Enthält alle Änderungsanträge. Die Klasse ChangeRequest wurde im Rahmen der Diplomarbeit von Christian Kluge spezifiziert.
other	Liste	Diese Liste enthält nur Verweise auf Dokumente. Diese sind u.U. vom Auftraggeber bereit gestellt worden.

Methoden	Beschreibung
login()	Ermöglicht unter Benutzung der Klasse UserLogin das Anmelden eines Benutzers
logout()	Meldet den zur Zeit angemeldeten Benutzer ab
createNewProject()	Erstellt mit Hilfe der Klasse NewProjectWizard ein neues, leeres Projekt.
openProject()	Lädt ein vorhandenes Projekt aus dem Repository.
checkIn()	Überführt Konfigurationseinheiten vom Workspace in das Repository.

checkOut()	Lädt Konfigurationseinheiten aus dem Repository in den Workspace.
print()	Ermöglicht das Drucken von Modellen, Schemata, Dokumentation, Aufgaben und Änderungsanträgen.
newModel()	Fügt ein neues Modell mit Hilfe des NewModelWizard in das Projekt ein.
newSchema()	Fügt ein neues Schema mit Hilfe der Klasse NewSchemaWizard in das Projekt ein.

C.3.2 Spezifikation der Klasse **`Task`**

Attribut	**Typ**	**Beschreibung**
createdOn	Datum	Datum, an dem die Aufgabe angelegt wurde.
startOn	Datum	Datum, an dem die Aufgabe beginnt.
deadline	Datum	Datum, an dem die Aufgabe erledigt sein soll.
realDeadline	Datum	Datum der tatsächlichen Fertigstellung der Aufgabe.
createdFor	Liste:User	Enthält die Mitarbeiter, für die diese Aufgabe erstellt wurde.
createdBy	User	Enthält den Mitarbeiter, der die Aufgabe angelegt hat.
state	Integer	Status der Aufgabe: 0 = neu, 1 = in Bearbeitung, 2 = erledigt.
priority	Integer	Priorität der Aufgabe: 0 = gering, 1 = mittel, 2 = hoch.
description	Text	Beschreibung der Aufgabe.
expenditure	Integer	Der tatsächliche Gesamtaufwand.
estimatedExpenditure	Integer	Vorher geschätzter Aufwand. Diese Aufwandsgrößen sind für die Projekt-Kalkulation von Bedeutung.

Methoden	**Beschreibung**
createNewTask()	Ermöglicht unter Benutzung der Klasse NewTaskWizard das Anlegen einer neuen Aufgabe.
print()	Druckt die Aufgabe aus.

C.3.3 Spezifikation der Klasse **`Model`**

Die Klasse **`Model`** ähnelt in ihrem Aufbau sehr stark der Klasse **`Schema`**. Aus diesem Grund wird lediglich sie genauer spezifiziert.

Attribut	**Typ**	**Beschreibung**
name	Text	Enthält den Namen des Modells.
description	Text	Beschreibung des Modells.
createdOn	Datum	Datum, an dem das Modell erstellt wurde.

createdBy	User	Enthält den Projektmitarbeiter, der das Modell erstellte.
deadline	Datum	Datum, an dem die Aufgabe erledigt sein soll.
realDeadline	Datum	Datum der tatsächlichen Fertigstellung der Aufgabe.
createdFor	Liste:User	Enthält die Mitarbeiter, die dieses Modell bearbeiten.
owner	User	Enthält den Mitarbeiter, der die Verantwortung für das Modell innehat Aufgabe angelegt hat.
stateObserver	Liste: (State, Datum, User)	Enthält die Liste der verschiedene Zustände, die das Modell bereits durchlaufen hat. Zu jede Zustandsänderung wird Datum und auslösender Mitarbeiter gespeichert. Status: 0 = angelegt, 1 = inEdit, 2 = stable, 3 = tested, 4 = qualityAssured, 5 = released
changeRequests	Liste: ChangeRequest	Enthält eine Liste aller zu diesem Modell gehörenden ChangeRequests
changeObserver	Liste: ChangeDescription	Enthält eine Liste aller Änderungen.
estimatedExpenditure	Integer	Geschätzte Gesamtbearbeitungszeit des Modells.
expenditure	Integer	Tatsächliche Bearbeitungszeit. Diese wird durch die im changeObserver gelisteten ChangeDescription berechnet.
data	ModelData	Hier werden die Daten des Modells abgespeichert. Dazu gehören alle benutzen E^3-Elemente und ihre Beziehungen zueinander.

Methoden	**Beschreibung**
createNewModelData()	Erstellt ein leeres Modell.
changeState()	Überführt das Modell in einen neuen Status.
print()	Druckt das Modell aus.
addChangeDescription()	Fügt nach einer Modifikation eine Beschreibung dieser Änderung ein.

C.3.4 Spezifikation der Klasse **`ChangeDescription`**

Attribut	**Typ**	**Beschreibung**
description	Text	Beschreibung der vorgenommenen Modifikationen.
createdOn	Datum	Datum, an dem die Veränderung vorgenommen wurde
createdBy	User	Enthält den Projektmitarbeiter, der die Modifikation vorgenommen hat.
reasons	Text	Beschreibung der Gründe für die Modifikation.
expenditure	Integer	Dauer der vorgenommenen Modifikation.

C.3.5 Alle bisher definierten Randtypen

Name	Typ	Rand oben	Rand links	Rand unten	Rand rechts	Bemerkung
OBEN_LINKS_UNTEN_RECHTS	0	3	3	3	3	Standardränder für alle Panele der untersten Ebene.
OBEN	1	3	0	0	0	
LINKS	2	0	3	0	0	
UNTEN	3	0	0	3	0	
RECHTS	4	0	0	0	3	
OBEN_LINKS	5	3	3	0	0	
LINKS_UNTEN	6	0	3	3	0	
UNTEN_RECHTS	7	0	0	3	3	
OBEN_UNTEN	8	3	0	3	0	
OBEN_RECHTS	9	3	0	0	3	
LINKS_RECHTS	10	0	3	0	3	
OBEN_LINKS_UNTEN	11	3	3	3	0	
OBEN_LINKS_RECHTS	12	3	3	0	3	
OBEN_UNTEN_RECHTS	13	3	0	3	3	
LINKS_UNTEN_RECHTS	14	0	3	3	3	
DEFAULT_BUTTON	15	4	4	4	4	Rand für den Standardbutton eines Fensters
TOOLBAR	16	0	0	0	0	Bei Verwendung mehrerer JToolBar innerhalb eines JToolBar diesen Rand zuweisen
DIVIDE_WINDOW	17	0	0	0	0	Abgrenzung des Haupt-Panels, wenn kein JTabbedPane bzw. JSplitPane verwendet wurde
EDITORPANE	18	0	0	0	0	Der Rand für JEditorPane, JTree usw. Er entspricht dem Standardrand, der bei allen Eingabeelementen genutzt wird.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Dresden, den 27. April 2001