Bericht Nr. 125

Interaktion in Systemen zur Bildfolgenauswertung basierend auf einem objektorientierten Ansatz

Volker Haarslev

FBI-HH-B-125/86

Dezember 1986

Fachbereich Informatik
Universität Hamburg
Schlüterstraße 70
D-2000 Hamburg 13

Ich hab oft arg mich geschunden,
Bis rechten Reim ich gefunden.
Es galt mancher Fluch
dem werdenden Buch –
Jetzt ist es gedruckt und gebunden.

Eugen Roth

Ich danke ganz herzlich allen Angehörigen, Diplomanden und Studienarbeitern unserer Forschungsgruppe für ihre Unterstützung bei der Durchführung dieser Arbeit.

Mein ganz besonderer Dank gilt meinem Betreuer Herrn H.-H. Nagel für seinen persönlichen Einsatz während der intensiven Betreuung dieser Arbeit sowie Herrn B. Neumann für seinen Beistand und die Begutachtung dieser Arbeit. Herrn B. Radig danke ich für sein Interesse und seine Unterstützung, die er in vielen Gesprächen meiner Arbeit entgegengebracht hat.

Mein aufrichtiger Dank gilt meinem Kollegen Herrn H. Faasch für die intensive Zusammenarbeit bei der Konzeption der Ada-Programmierumgebung sowie für kritische Anmerkungen zu einem Entwurf dieser Arbeit.

Ich danke meiner Kollegin Frau L.S. Dreschler-Fischer sowie Herrn W. Enkelmann aus Karlsruhe, die mit ihrer ständigen Bereitschaft zu zahlreichen Diskussionen wesentlich zum Entstehen dieser Arbeit beigetragen haben.

Frau U. Bauer danke ich für ihre Unterstützung und Entlastung von organisatorischen Tätigkeiten, Frau I. Heer-Mück und Herrn H. Krüger für ihren unermüdlichen Einsatz bei der Wartung der umfangreichen Laborapparatur.

Mein Dank gilt auch den Mitarbeitern unseres Rechenzentrums, insbesondere Herrn G. Friesland-Köpke für seine ständige Einsatzbereitschaft bei der Wartung des LaTeX-Systems.

Abschließend danke ich Sabine für ihr großes Verständnis und ihre Unterstützung beim Erstellen dieser Arbeit sowie meiner Katze Katharina für ihre zahlreichen Versuche, mich vom Schreiben abzulenken.

Kurzfassung

Die Interaktion zwischen Benutzern und Systemen zur Auswertung von Bildern und Bildfolgen genügt nur selten konsequent ergonomischen Prinzipien. Das Ziel dieser Arbeit besteht deshalb darin, die Interaktion von Benutzern mit derartigen Systemen zu verbessern. Dafür wird ein neuer Vorschlag zur Gestaltung der Benutzerschnittstelle von Bildfolgenauswertesystemen entwickelt. Dieser Entwicklungsvorgang stützt sich auf einer Modellierung der zukünftigen Benutzer einer solchen Schnittstelle und auf einer systematischen Untersuchung der Mensch-Maschine-Kommunikation für die Bildfolgenauswertung. Dieser Entwicklungsvorgang führt zu einer Benutzerschnittstelle für eine ganze Klasse von Bildfolgenauswertesystemen, die Benutzern eine objektorientierte Interaktion und eine direkte Manipulation graphischer Repräsentationen der Komponenten dieser Systeme gestattet. Um die Leistungsfähigkeit der vorgeschlagenen Benutzerschnittstelle aufzuzeigen, wird die Implementation eines Prototyps für ein entsprechendes Dialogsystem in der Programmiersprache Ada vorgestellt.

Abstract

The human-computer interface of systems for analysing image sequences is often neglected during the system design. The main concern of this thesis is the development of a human-computer interface better suited for these applications. Therefore a new approach to engineering the user interfaces of image sequence analysis systems is developed. This approach is based on a systematic analysis of the man-machine communication aspects of image sequence analysis systems. The result of this approach is the development of a user interface for a class of image sequence analysis systems. This user interface provides an object-oriented interaction capability and a direct manipulation of graphical representations of system components. The implementation of a prototype of a corresponding dialog system is described. This prototype is implemented using the programming language Ada.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Systeme für die Auswertung von Bildern und Bildfolgen sind komplexe Programmsysteme, die oft durch die Mitarbeit zahlreicher Autoren entstehen. Als Beispiel sei das MORIO-System [*Dreschler-Fischer et al. 83*] genannt. Auf diese Weise gewachsene Systeme berücksichtigen nur selten konsequent ergonomische Prinzipien für die Systemgestaltung und die Benutzerschnittstelle [*Maaß 84*]. Bei den Systemen zur Auswertung von digitisierten TV-Bildfolgen kommt erschwerend hinzu, daß es sich dabei um Experimentalsysteme handelt.

Aus dieser Beobachtung heraus erscheint es sinnvoll, dem Entwickler solcher Systeme deren Erstellung zu erleichtern. Ein in *Pfaff & Maderlechner 82* vorgestellter Ansatz zur automatischen Konfiguration und Erzeugung von Bildverarbeitungssystemen kann für den Bereich der Experimentalsysteme keine Anwendung finden, da bei diesen auch die Systemstruktur ein Gegenstand der Forschung ist. Eine generelle Erleichterung ist jedoch zu erwarten, wenn es gelingt, die Prinzipien der methodischen Programmentwicklung (software engineering) auch auf den Bereich der Bildverarbeitung zu übertragen [*Tanimoto 82*]. Diese Programmentwicklung sollte im Rahmen von Programmierumgebungen für die Bildverarbeitung erfolgen [*Brumfitt 84*]. Eine Programmierumgebung besteht üblicherweise aus einer Menge von Werkzeugen, die den Anwender bei der Erstellung seines Systems unterstützen.

Bei Systemen zur Auswertung von Bildern und Bildfolgen handelt es sich vorwiegend um interaktive Systeme, die dem Benutzer eine mehr oder weniger umfangreiche Überwachung und Beeinflussung des Verarbeitungsablaufs gestatten. Zwei wichtige Bestandteile bei der Entwicklung solcher Systeme sind somit die Interaktion und die Systemgestaltung, die deshalb als Kandidaten für entsprechende Werkzeuge einer Programmierumgebung in Frage kommen.

Neben der methodischen Programmentwicklung ist bei der Erstellung von Bildauswertesystemen die Wahl einer geeigneten Programmiersprache ein weiterer wichtiger Faktor. Für den Erfolg von Experimentalsystemen ist nicht nur der sorgfältige Entwurf, sondern auch die Unterstützung der Implementation durch geeignete Abstraktionen der Programmiersprache ausschlaggebend. Die meisten der bekannten Bildverarbeitungssysteme sind jedoch auch heute noch in FORTRAN implementiert. Ohne näher auf FORTRAN einzugehen, kann hier gesagt werden, daß diese Sprache nur geringe Möglichkeiten zur adäquaten Formulierung von Programm- und Datenabstraktionen bietet.

Eine Sprache, die dafür wesentlich besser geeignet ist, und die in ihren Möglichkeiten PASCAL weit übertrifft, ist die Programmiersprache Ada[1]. In die Sprache Ada wurden Konzepte aus der methodischen Programmentwicklung integriert. Weiterhin erlaubt sie die angemessene Formulierung von Programm- und Datenabstraktionen und bietet zur Beschreibung paralleler Abläufe ein Prozeßkonzept. Sie soll deshalb als Basis für den Entwurf einer objektorientierten Programmierumgebung zur Bildfolgenauswertung dienen [*Faasch & Haarslev 85*].

Als Bestandteil dieser Umgebung sind zwei Werkzeuge für die beiden oben beschriebenen wichtigen Bereiche von Bildauswertesystemen in der Entwicklung. Das eine Werkzeug beschäftigt sich mit der objektorientierten Systemgestaltung von Experimentalsystemen unter Verwendung komplexer Daten- und Programmabstraktionen [*Faasch 86*]. Das zweite Werkzeug behandelt die Interaktion der Benutzer mit dem Bildauswertesystem und wird im Rahmen dieser Arbeit dargestellt.

Die Interaktion zwischen dem Benutzer und experimentellen Bildauswertesystemen dient der Festlegung der Verarbeitungsdaten, des Ablaufs und der Steuerung des Systems. Der Benutzer hat dabei die Funktion eines Experten, der den Verarbeitungsablauf überwacht. Gleichzeitig versucht der Benutzer, ein adäquates System zur Lösung seiner Fragestellungen zu entwickeln.

Aufgrund der Erfahrung vieler Wissenschaftler finden das menschliche Denken und die damit verbundenen Problemlösungsprozesse überwiegend in visuellen Kategorien statt [*Benzon 85, Raeder 85*]. Die Strukturierung von Experimentalsystemen ist, wie oben erläutert, ein Bestandteil der Systementwicklung und unterliegt damit auch dem menschlichen Denken. Um

[1] Ada ist eine registrierte Handelsmarke der US-Regierung (Ada Joint Program Office)

den Entwickler bei diesen Vorgängen zu unterstützen, erscheint es sinnvoll, das Experimentalsystem und seine Struktur graphisch darzustellen. Gleichzeitig kann sich die Interaktion des Benutzers an dieser Systemdarstellung orientieren und ihm damit einen Bezug zwischen seinen Eingaben und den Systemreaktionen vermitteln.

Auch in dem Gebiet der Bild- und Bildfolgenauswertung existieren visuelle Vorstellungen, die konzeptuell die Struktur entsprechender Systeme geprägt haben. In dem MORIO-System [*Dreschler-Fischer et al. 83*] beispielsweise findet sich konzeptionell eine Unterteilung in Moduln, die in einer Fließbandverarbeitung organisiert sind, während in *Enkelmann 85* eine funktionelle Aufteilung des Programmsystems zur parallelen Ausführung auf einem homogenen Rechnernetz vorgenommen wurde. In *Stevens & Hunt 82* wird die Verwendung von Unix-Leitungen (pipes) für die Bildverarbeitung beschrieben.

Aus diesen Methoden zur Systemkonstruktion läßt sich als erster Ansatz eine Modellvorstellung dieser Systeme gewinnen, die als graphische Repräsentation in Form von miteinander verbundenen Komponenten existieren könnte. Es besteht damit eine Verwandtschaft zu CAD/CAM Systemen, wo ebenfalls Modelle über graphische Darstellungen beschrieben und konstruiert werden. Diese Systeme modellieren jedoch keine Programmsysteme und behandeln mehr die statischen Aspekte der zu modellierenden Gegenstände.

In dem Bereich der Programmentwicklung existieren bereits Ansätze, die den Vorgang der Programmerstellung durch graphische Darstellungen der sich in der Entwicklung befindenden Programme unterstützen. Ein Beispiel für ein derartiges System ist INCENSE [*Myers 83*], das dem Benutzer die graphische Darstellung von Datenstrukturen seines Programms im Rahmen eines Prüfsystems gestattet. In *Shneiderman 83* wird nicht nur die graphische Darstellung von Programmsystemen, sondern auch deren Konstruktion durch die direkte Manipulation von graphischen Darstellungen gefordert. Damit werden die Konzepte von CAD/CAM Systemen erweitert und für die Konstruktion von Programmsystemen angewendet.

Für den Benutzer sind aber nicht nur die statischen, sondern auch die dynamischen Aspekte eines Programmsystems von Interesse. Das Gebiet der graphischen Darstellung der Wirkungsweise von Algorithmen (algorithm animation) beschäftigt sich mit den dynamischen Aspekten eines Programmablaufs. Als ein Beispiel ist die Programmumgebung BALSA [*Brown & Sedgewick 84, Brown & Sedgewick 85*] zu nennen, die dem Be-

nutzer eine Verdeutlichung ihrer Algorithmen über dynamische Graphiken gestattet.

In *Lieberman 84* wird die Programmierumgebung TINKER vorgestellt, die für Lisp-Programme die Auswirkungen ihrer Operationen auf den Programmdaten graphisch anzeigt. Ein anderes System basiert auf der Smalltalk-Umgebung und verwendet deren Möglichkeiten zur graphischen Darstellung [*London & Duisberg 85*].

Im Vergleich zu den bisher vorgestellten Systemen gehen die graphischen Programmierumgebungen noch einen Schritt weiter. Sie bieten nicht nur eine einheitliche graphische Darstellung der statischen und dynamischen Aspekte von Programmsystemen, sondern gestatten auch die direkte Manipulation der Programme in Form von graphischen Symbolen. Ein derartiges System für das Anwendungsgebiet der Leistungsanalyse wird in *Melamed & Morris 85* beschrieben. Dieses System bietet dem Benutzer die Möglichkeit, Simulationen graphisch zu erstellen, zu beobachten und direkt zu manipulieren. Einen weitergehenden Ansatz verfolgt die PegaSys-Umgebung [*Moriconi & Hare 85*], die den Programmentwurf der Benutzer durch eine Hierarchie voneinander abhängiger graphischer Darstellungen beschreibt. Mit ThinkPad [*Rubin et al. 85*] wird ein System vorgestellt, das eine Programmierung durch Demonstration unterstützt. Die Beziehungen zwischen den Systemkomponenten werden mithilfe von Einschränkungen oder Zusicherungen beschrieben. In PICT [*Glinert & Tanimoto 84*] wird fast vollständig auf die Verwendung textueller Darstellungen verzichtet. Der Benutzer konstruiert dort seine Programme mithilfe von Bildsymbolen, die in Form von Piktogrammen graphisch dargestellt werden.

Zusammenfassend läßt sich sagen, daß bei der Entwicklung der graphischen Programmierumgebungen zur Zeit der Schwerpunkt in der Erprobung von adäquaten Techniken zur graphischen Darstellung und Interaktion liegt. Dabei wird die Programmkonstruktion auf der Ebene der Anweisungen und Prozeduren unterstützt. Der graphischen Darstellung von Programmen auf der Ebene von Moduln und der Entwicklung einer benutzergerechten Interaktion wird in diesen Systemen eine geringere Bedeutung beigemessen. Zudem wurden diese Umgebungen immer nur für spezielle Sprachen mit einem stark eingeschränkten Sprachumfang entwickelt, so daß diese Systeme eine geringe Portabilität besitzen.

Die Erfahrungen mit den graphischen Programmierumgebungen sind für den Bereich der Bildfolgenauswertung im Rahmen dieser Arbeit nicht direkt verwendbar, da hier die Interaktion mit Bildfolgenauswertesystemen

untersucht wird. Diese Erfahrungen unterstützen jedoch die Aufstellung der Hypothese, daß bei Verwendung der drei nachfolgenden Konzepte die Entwicklung einer benutzergerechten Interaktion mit Bildfolgenauswertesystemen wesentlich unterstützt wird:

1. Graphische Darstellung der Systemstruktur von Bildfolgenauswertesystemen;
2. Strukturierung und Veränderung durch die direkte Manipulation von graphischen Repräsentationen;
3. Graphische Darstellung der Wirkungsweise von Bildfolgenauswertesystemen (und ihrer Algorithmen).

Die Modellierung der Interaktion des Benutzers mit einem Bildfolgenauswertesystem erfolgt durch die Gestaltung einer *Benutzerschnittstelle.* Diese Benutzerschnittstelle definiert die Kommunikation zwischen dem Benutzer und einem Bildfolgenauswertesystem (siehe Abbildung 1.1).

Abbildung 1.1: Benutzerschnittstelle

Die obigen Konzepte beziehen sich auf die visuellen Aspekte der Interaktion mit Bildfolgenauswertesystemen und bilden eine Grundlage für die graphische Gestaltung der Schnittstelle. Das nachfolgende Konzept betrifft die Architektur interaktiver Systeme und damit auch Bildfolgenauswertesysteme:

4. Strukturelle Aufteilung von Bildfolgenauswertesystemen in Dialogsystem und Anwendungssystem (siehe Abbildung 1.2).

Dieses bei der Architektur interaktiver Systeme allgemein anerkannte Prinzip [*Coutaz 84a, Coutaz 84b, Draper & Norman 85, Hayes 84, Olsen et al. 84, Shaw et al. 83*] bietet mehrere Vorteile. Die Systemstruktur gestattet eine klare Trennung zwischen der Interaktion und der Anwendung. Das Anwendungssystem realisiert die durchzuführenden Aufgaben

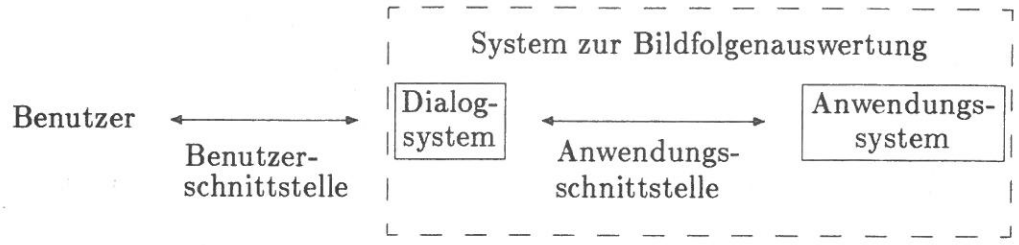

Abbildung 1.2: Struktur von Bildfolgenauswertesystemen

des Benutzers, während das Dialogsystem die Behandlung der Ein- und Ausgabe sowie die Form der Interaktion mit dem Benutzer erfaßt. Diese Trennung ermöglicht eine weitere Modularisierung von Bildauswertesystemen und damit eine Reduzierung der Komplexität dieser Systeme. Weiterhin wird eine unabhängige Entwicklung beider Systemteile ermöglicht. Die Kommunikation zwischen dem Dialog- und dem Anwendungssystem findet über eine *Anwendungsschnittstelle* statt, die für den Benutzer verdeckt ist. Der Entwickler eines derartigen Dialogsystems ist damit in der Lage, die Kommunikation zwischen dem Benutzer und dem Dialogsystem beliebig zu modifizieren, sofern davon nicht die Anwendungsschnittstelle berührt wird.

Durch diese Aufgabenteilung wird das Anwendungssystem von der Dialogführung befreit, die beispielsweise beim MORIO-System einen Anteil von etwa 20 Prozent (gemessen in Quellcode-Zeilen) ausmacht [*Faasch 86*]. Neben dieser Aufgabenentlastung besteht die Möglichkeit, die Portabilität der Anwendungssysteme zu erhöhen, sofern es gelingt, eine einheitliche Anwendungsschnittstelle zu schaffen. Derartige Überlegungen existieren bereits in der Bildverarbeitung für die Behandlung von Bildspeichern [*Stevens & Alexander 83*], für die Einbettung in Betriebssysteme [*Krusemark & Haralick 82*] und in dem Bereich der graphischen Ein- und Ausgabe durch das Graphische Kernsystem (GKS) [*Enderle et al. 84*].

Bei der Verwendung eines universellen Dialogsystems, das eine derartige Anwendungsschnittstelle realisiert, hat der Benutzer den Vorteil, daß die interaktiven Bildauswertesysteme eine einheitliche Benutzerschnittstelle besitzen. Er muß sich dann nicht mehr an die unterschiedlichen Konventionen zur Interaktion anpassen, die bei verschiedenen Systemen meistens vorhanden sind.

Bei der Entwicklung einer Benutzerschnittstelle sind aber nicht nur die

visuellen Aspekte der Schnittstelle wichtig, sondern auch die konsequente Konzeption der Schnittstelle ausgehend von dem zukünftigen Benutzer. Diese Entwicklung wird erleichtert, wenn Konzepte der methodischen Programmentwicklung auf den Entwurf von Benutzerschnittstellen angewendet werden [*Draper & Norman 85*].

In dieser Arbeit wird deshalb die Entwicklung einer Benutzerschnittstelle für Bildfolgenauswertesysteme in zwei Schritten durchgeführt. Zuerst wird ein Modell entworfen, das den zukünftigen Benutzer und seine Erwartungen beschreibt, und dann aus diesem die Konzeption einer graphischen Schnittstelle abgeleitet. Dieser Vorgang gründet sich auf dem folgenden Konzept:

5. Entwurf einer interaktiven Benutzerschnittstelle für Bildfolgenauswertesysteme ausgehend von einem vorher entwickelten Benutzermodell.

Das Benutzermodell beschreibt auch die Interaktion mit Bildfolgenauswertesystemen im Rahmen der Benutzerschnittstelle. Die Art und Form der Interaktion ist aber in vielen interaktiven Systemen — und speziell auch in Systemen zur Verarbeitung und Auswertung von Bildfolgen — an bestimmte Zustände (modes) der Benutzerschnittstelle gebunden. Diese Zustände werden im weiteren als *Modi*[2] bezeichnet. Interaktive Systeme verwenden Modi, um sämtliche Benutzereingaben in einer vom aktuellen Modus des Systems abhängigen einheitlichen Form zu interpretieren.

Das Problem bei solchen Systemen besteht nun darin, daß häufig bei der Realisierung der Benutzerschnittstelle ungenügende Vorstellungen über die adäquate Struktur der Interaktion und eine den Benutzeraufgaben angemessene Abstraktionsebene der angebotenen Operationen vorlagen. Dies hat zur Folge, daß der Benutzer Operationen benötigt, die nur in verschiedenen Modi verfügbar sind. Der Benutzer wird somit gezwungen, zur Lösung seines Problems in verschiedenen Modi zu arbeiten. Die meisten interaktiven Systeme gestatten aber nur einen Modus zur Zeit. Deshalb muß der Benutzer einen anderen Modus wählen. Der zur Zeit des Moduswechsels vorhandene Interaktionskontext wird aber üblicherweise nicht gespeichert, so daß dieser Moduswechsel den Verlust des vorherigen Kontextes zur Folge hat.

Zur Lösung dieser Probleme werden in dieser Arbeit deshalb Konzepte aus dem Bereich der objektorientierten Programmierung [*Stefik & Bobrow*

[2]Eine genaue Definition dieses Begriffs folgt in Kapitel 3.3.6

86] in analoger Weise zur Strukturierung der Benutzerschnittstelle angewendet:

6. Modellierung der Benutzerschnittstelle auf der Basis von Objekten.

Diese *objektorientierte Interaktion* beruht darauf, daß der Benutzer nicht mehr mit dem gesamten System, sondern nur mit einzelnen Objekten kommuniziert. Als *Objekte* werden eigenständige Systemkomponenten verstanden, mit denen der Benutzer eine Interaktion durchführen kann. Die Art und Form der Interaktion wird ausschließlich durch die *Interaktionsschnittstelle* der Objekte bestimmt. Die Interaktionsschnittstelle eines Objektes legt die von ihm akzeptierten Benutzereingaben und damit das *Kommunikationsprotokoll* fest.

Das Problem der Modi wird durch die objektorientierte Interaktion gelöst, da die Objekte einen Satz auf sie anwendbarer Operationen zur Interaktion definieren. Der Benutzer kann während der Interaktion zwischen den Objekten wechseln. Jedes Objekt kann ihm zu jedem Zeitpunkt Auskunft über die anwendbaren Operationen geben. Der Interaktionskontext ist an die einzelnen Objekte gebunden, ein Kontextwechsel oder -verlust im obigen Sinne findet nicht mehr statt.

Das Problem der ungenügenden Abstraktionsebene der angebotenen Operationen läßt sich vereinfachen, indem man dem Benutzer eine Interaktion auf unterschiedlichem Abstraktionsniveau gestattet. Die von Bildfolgenauswertesystemen zur Zeit angebotenen Operationen zur Interaktion sind teilweise noch ungenügend, deshalb muß eine geeignete Ebene der Abstraktion gefunden werden. Dieser Vorgang läßt sich erleichtern, wenn man dem Benutzer das Bilden von höheren Abstraktionen zur Interaktion gestattet. Dieser Weg soll hier im Rahmen der objektorientierten Interaktion beschritten werden.

Anhand der sechs aufgeführten Konzepte wird in dieser Arbeit die Interaktion mit Bildfolgenauswertesystemen analysiert und ein neuer Vorschlag zur objektorientierten Gestaltung der Interaktion vorgestellt. Dabei wird darauf Wert gelegt, daß die Konzeption einer entsprechenden Benutzerschnittstelle für Bildfolgenauswertesysteme von einer speziellen Programmiersprache völlig unabhängig ist. Die Darstellung dieses Entwicklungsvorganges gliedert sich wie folgt:

In *Kapitel 2* werden fünf interaktive Programmsysteme beschrieben, die für bestimmte Bereiche als innovativ angesehen werden können. Diese

Systeme sollen als Ausgangspunkt für die Ableitung wünschenswerter Interaktionskonzepte dienen.

Ausgehend von diesen Konzepten folgt in *Kapitel 3* der Entwurf eines Benutzermodells und die Aufstellung von Konzepten zur Gestaltung einer Benutzerschnittstelle. Diese Konzepte bilden die Basis für die weitere Entwicklung der Benutzerschnittstelle.

In *Kapitel 4* findet eine Darstellung der interaktiven Schnittstellen von Systemen zur Verarbeitung und Auswertung von Bildern und Bildfolgen statt. Die in diesen Systemen verwendeten Interaktionskonzepte und ihre Form der Darstellung während der Interaktion werden abgeleitet. Anschließend folgt ein Vergleich dieser Systeme anhand der in Kapitel 3 aufgestellten Konzepte.

Ausgehend von dem entwickelten Benutzermodell und den Erkenntnissen über interaktive Bildauswertesysteme wird in *Kapitel 5* die Konzeption einer Benutzerschnittstelle vorgestellt. Diese Konzeption basiert auf einem objektorientierten Ansatz und bietet dem Benutzer eine direkte Manipulation von graphisch dargestellten Objekten.

Kapitel 6 stellt einen neuartigen Systementwurf für ein Dialogsystem vor, das die konzipierte Benutzerschnittstelle realisiert, und beschreibt die Implementation eines Prototyps mit der Programmiersprache Ada.

In *Kapitel 7* werden die Ergebnisse dieser Arbeit noch einmal zusammengefaßt und ein Ausblick auf weitere Entwicklungen gegeben.

Kapitel 2

Interaktion am Beispiel von innovativen Systemen

In diesem Kapitel werden fünf interaktive, integrierte Programmsysteme oder Programmierumgebungen dargestellt. Sie sollen als innovativ bezeichnet werden, da sie auf neuen Konzepten basieren, die sich als ausbaufähig und erfolgreich erwiesen haben.

Ein wesentliches Merkmal dieser Systeme ist ihre interaktive Handhabung durch den Benutzer. Diese Interaktionsfähigkeit mit den zugrundeliegenden Konzepten hat entscheidend zu ihrer "Innovativität" beigetragen.

Das Ziel dieses Kapitels liegt darin, die dargestellten Systeme als Ausgangspunkt für die Abstraktion wünschenswerter Interaktionskonzepte zu verwenden. Die dabei gewonnenen Erkenntnisse werden dann in analoger Anwendung auf den Bereich der Bildfolgenauswertung übertragen.

Das Unix[1]-System [*Ritchie & Thompson 74*] bietet eine Kommandosprache an, die dem Benutzer eine flexible und kompakte Notation zur einfachen Verknüpfung von Programmen ermöglicht. Das Interlisp-System [*Teitelman & Masinter 81*] ist eine der ersten interaktiven Programmierumgebungen gewesen, die alle zur Programmentwicklung notwendigen Werkzeuge enthielt. Das Smalltalk-System [*Goldberg & Robson 83*] hat sich durch einen konsequent angewendeten objektorientierten Ansatz hervorgetan, der zu einer Programmierumgebung geführt hat, die bis auf die unterste Ebene aus Objekten besteht. Das Cedar-System [*Teitelman 85*] ist als eine Weiterentwicklung von Interlisp und Smalltalk anzusehen und basiert auf einer zu PASCAL ähnlichen Sprache. Das Xerox Star System [*Lipkie et al. 82*] wurde für eine spezielle Anwendung, die Textverarbeitung, entworfen und entstand aus den Erfahrungen mit Smalltalk. Durch den eingeschränkten Funktionsumfang unterscheidet es sich auch von den vier

[1] Unix ist eine Handelsmarke von AT & Bell Laboratories

anderen Systemen.

Alle diese Systeme werden nachfolgend genauer dargestellt. Im Anschluß daran werden die von diesen Systemen verwendeten Konzepte zur Interaktion auf ihre Verwendbarkeit für den Bereich der Bildfolgenauswertung bewertet. Auf Grund dieser Beurteilung werden die für die Bildfolgenauswertung geeigneten Konzepte ausgewählt.

2.1 Das Unix-System

Das Unix-System [*Ritchie & Thompson 74*] ist ein sehr weit verbreitetes Betriebssystem und hat viele Entwicklungen beeinflußt. Es ist aus den Erfahrungen mit dem MULTICS-System entstanden und hat die besten Konzepte von diesem übernommen und weiterentwickelt.

Unix hat mehrere bemerkenswerte Eigenschaften, die seine schnelle Verbreitung unterstützt haben [*Kernighan & Mashey 81*]. Das Dateisystem ist hierarchisch strukturiert. Dateien werden als Folge von Bytes angesehen; die Struktur einer Datei wird durch Anwendungsprogramme induziert und obliegt deren Interpretation. Sowohl Inhaltsverzeichnisse als auch E/A-Geräte werden als Dateien angesehen. Durch die hierarchische Struktur des Dateisystems und die Einbettung der verfügbaren Geräte in Form von speziellen Dateien ist eine einheitliche Behandlung der Ein- und Ausgabe möglich.

Unix verfügt über ein Mehrprozeßkonzept. Ein Prozeß kann weitere Prozesse als seine Unterprozesse erzeugen. Die Prozesse können im Prinzip parallel zueinander ausgeführt werden. Die Kommunikation zwischen Prozessen wird über einen *Kanal* oder auch eine *Datenleitung* (*pipe*) abgewickelt. Der Datenaustausch findet über die normalen E/A-Anweisungen des Dateisystems statt.

Die Benutzerschnittstelle wird durch ein spezielles Programm (*shell*) realisiert. Dieses interpretiert zeilenweise die Eingaben des Benutzers und führt dessen Befehle aus. Dieser Interpretierer definiert eine Kommandosprache, die durch ihre vielfältigen Fähigkeiten mit einer Programmiersprache vergleichbar ist und als beispielhaft für eine kompakte und elegante Kommandosprache gilt.

Der Interpretierer öffnet vor der Aktivierung eines Programms zwei Dateien, die als Standardeingabe bzw. -ausgabe dieses Programms verfügbar sind. Alle Programme, die ihre Eingabe von der Standardeingabe der

Kommando-Interpretation lesen und ihre Ausgabe entsprechend auf die Standardausgabe schreiben, heißen in der Unix-Terminologie *Filter*. Die Voreinstellung für die Standardeingabe und -ausgabe ist das Benutzerterminal.

Die Unix-Konventionen für die Eingaben einer einfachen Kommandozeile sind ($\supset$ bedeutet Zeilenende)

$$\texttt{Kommando Parameter}_1\ \texttt{Parameter}_2 \cdots \texttt{Parameter}_n \supset,$$

wobei der Kommandoname und die Parameter durch Leerzeichen getrennt werden. Durch die Symbole '<' und '>' kann der Benutzer die Voreinstellung für die Standard-E/A verändern. Die vollständige Form eines Kommandos lautet dann

$$\texttt{Kommando Parameter}_1 \cdots \texttt{Parameter}_n < \texttt{Eingabe} > \texttt{Ausgabe} \supset.$$

Eingabe und *Ausgabe* bezeichnen jeweils die Quelle bzw. Senke für die Standard-E/A.

Durch die Verwendung des Symbols '|' kann eine Folge von Kommandos getrennt werden. Der Interpretierer führt diese Kommandos dann parallel aus und steuert die Standard-E/A derart, daß eine Fließbandverarbeitung (pipeline) aufgebaut wird. Die Syntax für den Aufbau einer Fließbandverarbeitung mithilfe von Filtern und Datenleitungen ist sehr einfach:

$$\texttt{Kommando}_1 < \texttt{Eingabe} \mid \texttt{Kommando}_2 \mid \cdots \mid \texttt{Kommando}_n > \texttt{Ausgabe} \supset.$$

Kommando$_1$ wird auf die *Eingabe* der Leitung angewendet, dessen Ausgabe wird an *Kommando*$_2$ gesendet, usw., bis abschließend die Ausgabe von *Kommando*$_n$ als *Ausgabe* der Leitung gespeichert wird.

Der Interpretierer erlaubt weiterhin die Angabe von mehreren hintereinander auszuführenden Kommandos auf einer Zeile (getrennt durch ein ';') und die Ausführung einer Kommandofolge im Hintergrund (abgeschlossen durch ein '&').

Im Rahmen der Unix-Programmierumgebung hat sich deshalb ein Programmstil entwickelt, der statt der Konstruktion von großen Programmen die Konstruktion von vielen kleinen Programmen für spezielle Aufgaben bevorzugt. Diese können nach Bedarf mit den oben beschriebenen Möglichkeiten der Kommandosprache kombiniert und verknüpft werden, um umfangreiche Aufgaben durchzuführen. Eine Untersuchung hat gezeigt,

daß die Dateien der Benutzer von Unix-Systemen teilweise mehr Kommandoprozeduren als ausführbare Programme enthalten [*Kernighan & Mashey 81*]. Die Unix-Kommandosprache besitzt derart viele Möglichkeiten, daß sie durchaus als Programmiersprache bezeichnet werden kann. Deshalb werden oft von geplanten Anwendungen Prototypen mithilfe der Kommandosprache erstellt.

Trotz aller Vorzüge ist Unix keineswegs vorbildlich, wenn die interaktive Schnittstelle von Unix nach ergonomischen Kriterien bewertet wird. Unix kann mehrere einfache Forderungen, die üblicherweise an interaktive Systeme gestellt werden, nicht erfüllen [*Norman 81*]:

- *Konsistenz:* Die Kommandonamen und die von der Programmiersprache C verwendeten Namen für dieselben Funktionen sind inkonsistent. Das Kopier-Kommando (`copy`) und die äquivalente Funktion in C (`strcpy`) haben unterschiedliche Namen und eine vertauschte Reihenfolge ihrer Parameter. Unix-Kommandos (`move`, `copy`) zur Dateibehandlung können vorhandene Dateien ohne Warnung zerstören. Viele Programme besitzen spezielle optionale Parameter (argument flags), aber die Art und Weise ihrer Spezifikation ist inkonsistent und variiert bei unterschiedlichen Programmen. Viele Kommandonamen sind als Abkürzung aus ihrer Funktion entstanden. Diese Abkürzungen wurden jedoch völlig inkonsistent gewählt:
  `c compiler` ⇔ `cc`, `change working directory` ⇔ `chdir`, `change password` ⇔ `passwd`, `concatenate` ⇔ `cat`, `copy` ⇔ `cp`, `search file for pattern` ⇔ `grep`.

- *Funktionalität:* Viele Kommandonamen scheinen keinen Bezug zu ihrer Funktion zu besitzen. Das Programm zum Ausdrucken einer Datei auf dem Benutzerterminal heißt beispielsweise nicht `list` oder `type` sondern `cat`. Erklärungen zu Kommandos erhält der Benutzer nicht über `help` sondern durch `apropos`.

- *Zustandsinformation:* Unix ist bei seinen Operationen mit Meldungen an den Benutzer äußerst sparsam; soweit es geht, werden sie vermieden. Dieser Mangel an Rückmeldungen macht es dem Benutzer schwer, den momentanen Modus des Systems herauszufinden. Das Fehlen von mnemonischen Kommandostrukturen stellt an das Gedächtnis hohe Anforderungen und kann leicht fehlerhafte Benutzereingaben zur Folge haben.

Es gibt jedoch Ansätze, die Benutzerschnittstelle von Unix anders und damit benutzerfreundlicher zu gestalten. In *Gittins et al. 84* wird die graphische Schnittstelle UNICON (unix icon) vorgestellt, die dem Benutzer die Interaktion mithilfe von farbig dargestellten Piktogrammen gestattet. Dabei wurde vorerst ein Kern von Operationen zur Dateiverwaltung realisiert.

2.2 Das Interlisp-System

Interlisp ist eine der ersten integrierten Programmierumgebungen gewesen, die dem Benutzer eine hochgradig interaktive Erstellung von Programmen erlaubt hat [*Teitelman & Masinter 81, Sandewall 78*]. Lisp basiert auf dem Konzept der Liste. Sowohl Daten als auch Programme existieren in Lisp als Listenstrukturen. Die äquivalente Darstellung von Daten und Programmen sowie die interaktive Verfügbarkeit von Lisp durch einen entsprechenden Interpretierer haben das frühe Entstehen einer Programmierumgebung begünstigt.

Lisp-Umgebungen werden vor allem unter folgenden Aspekten benutzt:

- Programmieren als Experiment;
- der Akt der Programmerstellung ist Gegenstand der Forschung;
- schnelle Erstellung von Prototypen, um Prinzipien zu verifizieren;
- "Bottom-up" Entwicklung von Programmen durch "strukturiertes Wachsen";
- sofortiges Testen von Programmen durch deren Interpretation.

Interlisp ist ein integrierter Ansatz, bei dem alle verfügbaren Komponenten ein einheitliches System bilden. Die verschiedenen Programme können jederzeit unterbrochen und andere aufgerufen werden, ohne daß der Kontext des unterbrochenen Programms verloren geht. Da alle Komponenten in Lisp implementiert sind, kann das System durch die Benutzer leicht erweitert und an die Bedürfnisse der Benutzer angepaßt werden.

Die Interlisp-Umgebung enthält mehrere Hilfsmittel, die sich aufgrund ihres Komforts als wegweisend für andere interaktive Systeme erwiesen haben. Es existiert beispielsweise ein Programm (*masterscope*), welches eine Analyse von existierenden Programmen durchführt und den Benutzer bei einer Änderung durch die Erstellung von Kreuz-Referenzlisten unterstützt. Eine weitere bekannte Komponente *DWIM* (*Do-What-I-Mean*) wird von

dem Basissystem beim Auftreten eines Fehlers aktiviert. DWIM versucht, die Intention des Benutzers bei der fehlerhaften Operation zu ergründen und diese Operation entsprechend zu korrigieren. Eine einfache Anwendung ergibt sich bei der Korrektur von Schreibfehlern. Der *"Programmer's Assistant"* ist eine interaktive, bildschirmorientierte Schnittstelle, die den Benutzer bei der Programmerstellung unterstützt. Interlisp führt eine *Historie* über den Interaktionsablauf. Der Benutzer kann eine Kopie dieser Historie inspizieren, die dort enthaltenen Eingaben editieren und erneut ausführen. Weiterhin existiert die Möglichkeit, die Wirkung von bereits ausgeführten Operationen wieder rückgängig zu machen.

Lisp-Systeme haben sich besonders in dem Bereich der Künstlichen Intelligenz (KI) bewährt. Durch die Äquivalenz von Programmen und Daten konnten in Lisp besonders leicht Konzepte für den KI-Bereich entwickelt und untersucht werden.

Eines dieser Konzepte ist die *datengesteuerte Programmierung.* Sie ähnelt den indirekten Sprüngen aus Maschinensprachen und kann wie folgt beschrieben werden. Eine Prozedur kann in konventionellen höheren Programmiersprachen nur eine andere aufrufen, wenn sie deren Namen explizit als Prozeduraufruf in ihrem eigenen Rumpf aufführt. Ein datengesteuerter Aufruf hingegen bewirkt, daß ausgehend von den Eingabedaten in einer Datenbasis nach einer Prozedur gesucht wird, die mit diesen Daten in einer gewissen Weise assoziiert ist. Diese Prozedur wird dann aufgerufen.

Ein anderes Konzept sind die *Programmgeneratoren.* Dabei wird zur Problemlösung ein Programm entwickelt, das wieder andere Programme erzeugt. Diese Methode wird oft eingesetzt, wenn ähnliche Probleme zusammengefaßt und durch einen einheitlichen Algorithmus beschrieben werden können. Dieser Algorithmus wird dann in einem Programmgenerator formuliert, der für eine spezielle Anwendung ein entsprechendes Programm erzeugt. Dieses kann nach seiner Erzeugung sofort durch seine Interpretation ausgeführt werden.

Im Zusammenhang mit der Programmerzeugung steht die *beratende Programmierung.* Sie erlaubt dem Benutzer Änderungen an Programmen ohne Wissen über deren Struktur. In Interlisp hat der Benutzer dadurch die Möglichkeit, den Kontrollfluß einer zu ändernden Prozedur umzuleiten und damit zusätzliche Seiteneffekte zu erzielen. Analog zu den Maschinensprachen wird diese Form auch als "high-level patching" bezeichnet.

2.3 Das Smalltalk-System

Smalltalk-80 [*Goldberg & Robson 83*] ist eine Programmierumgebung, die auf dem Modell der kommunizierenden Objekte basiert. Weiterhin ist Smalltalk eine graphische, interaktive Programmierumgebung. Jede Komponente des Systems, die für den Benutzer zugreifbar ist, kann sinnvoll zur Beobachtung und Manipulation dargestellt werden. Die Benutzerschnittstelle kann als ein Versuch angesehen werden, eine visuelle Sprache für jedes Objekt zu erzeugen.

Die Sprache Smalltalk basiert auf wenigen grundlegenden Konzepten. Diese sollen im folgenden kurz dargestellt werden. Anschließend folgt eine Beschreibung der interaktiven Benutzerschnittstelle.

Ein *Objekt* stellt eine Komponente des Smalltalk-80 Programmsystems dar. Ein Objekt kann u.a. verstanden werden als Zahl, Zeichenkette, Inhaltsverzeichnis, Programm, Übersetzer usw. Es besteht aus privaten Datenstrukturen und einer Menge darauf anwendbarer Operationen. Die Objekte unterstützen die Modularität des Systems, da ihre Funktionsweise von keinen internen Details anderer Objekte abhängt.

Eine *Nachricht* ist eine Aufforderung für ein Objekt, eine seiner Operationen auszuführen. Ein Objekt, welches Empfänger einer Nachricht ist, bestimmt die Art und Weise, wie eine Operation ausgeführt wird. Eine Nachricht kennzeichnet nur die auszuführende Operation, nicht die Form der Ausführung.

Die *Schnittstelle* bezeichnet die Menge aller Nachrichten, auf die ein Objekt antworten kann. Ein Objekt kann nur über seine Schnittstelle angesprochen werden. Die privaten Daten des Objektes können nur von dessen Operationen manipuliert werden.

Eine *Klasse* beschreibt die Implementation einer Menge von Objekten, die alle dieselbe Art von Systemkomponenten darstellen. Das Klassenkonzept, welches sich an dem Klassenkonzept von SIMULA orientiert, dient der Zusammenfassung ähnlicher Systemkomponenten. Die durch eine Klasse beschriebenen individuellen Objekte heißen *Instanzen* dieser Klasse. Eine Klasse beschreibt ihre Objekte durch den Umfang und die Struktur der privaten Daten. Diese privaten Daten bestehen aus einer Menge von *Instanzvariablen*. Weiterhin beschreibt die Klasse eine Menge von *Methoden*, die die Ausführung der Operationen ihrer Objekte definieren.

Durch das Bilden von Teil- oder Unterklassen (ähnlich wie in SIMULA) gestattet Smalltalk eine hierarchische Definition von Klassen. Aus derar-

tigen Klassenstrukturen werden gemeinsame Merkmale durch Vererbungshierarchien abgeleitet.

Programmieren in Smalltalk besteht aus der Erzeugung von Klassen, von Instanzen dieser Klassen und aus der Spezifikation von Nachrichtenfolgen, die zwischen den Objekten ausgetauscht werden. Die Sprache Smalltalk ist bis auf die unterste Ebene objektorientiert, da auch arithmetische Ausdrücke aus Objekten und Nachrichten bestehen. Die Programmierung erfolgt als interaktiver Vorgang im Rahmen des Smalltalk-80 Systems.

Die Interaktion zwischen dem Benutzer und der Smalltalk-80 Umgebung erfolgt durch einen *hochauflösenden Rastergraphik-Bildschirm*, eine *Tastatur* und ein *Zeigeinstrument.* Der Bildschirm dient der Darstellung von graphischer und textueller Information. Die Tastatur steht dem Benutzer zur Eingabe textueller Information zur Verfügung. Das Zeigeinstrument wird zur Auswahl von auf dem Bildschirm dargestellter Information verwendet. Es handelt sich dabei um eine *Maus* mit drei Knöpfen, die dem Benutzer als zusätzliche Signalgeber für die graphische Interaktion zur Verfügung stehen. Die durch die Maus aktuell angesprochene Position auf dem Bildschirm wird durch eine Marke angezeigt.

Der Bildschirm enthält einen oder mehrere rechteckige Bereiche, die *Fenster* (*views*) genannt werden. Ihre Darstellung erfolgt auf einem grauen Hintergrund. Die Fenster können sich überlappen. Jedes Fenster besitzt einen Titel zur Identifikation. Zur Durchführung einer Aktion muß der Benutzer die graphisch dargestellte Information markieren, die von der gewünschten Aktion betroffen sein soll. Dieses wird als *Auswahl* bezeichnet und erfolgt durch das Positionieren der Marke mit der Maus und der Betätigung einer ihrer Knöpfe. Das System gibt für eine Auswahl eine graphische Rückmeldung.

Ein selektiertes Fenster wird immer vollständig dargestellt. In einem Fenster kann der Benutzer jederzeit textuelle Information erstellen und modifizieren. Dies geschieht durch einen integrierten Texteditor. Diese vom Benutzer allgemein verwendbaren Fenster werden im folgenden zur Unterscheidung von speziell genutzten Fenstern auch als Hauptfenster bezeichnet.

Für jedes Hauptfenster sind zwei Menüs definiert, die sich der Benutzer darstellen lassen kann. Für deren Aktivierung sind zwei der drei Knöpfe der Maus fest reserviert. Ein aktives Menü stellt seine Information in einem eigenen Menüfenster dar. Das erste Menü enthält eine Liste von ausführbaren Befehlen. Die Art der verfügbaren Befehle hängt von der in dem Haupt-

fenster dargestellten Information ab. Handelt es sich beispielsweise um textuelle Information, so werden Kommandos zur Textverarbeitung angeboten. Das zweite Menü bietet Kommandos an, die an diesen Typ von Hauptfenster gebunden und unabhängig von der dort aktuell dargestellten Information sind.

Da meistens der Umfang der Menüfenster zur vollständigen Darstellung des Menüs nicht ausreicht, wird nur ein Ausschnitt dargestellt. Der Benutzer kann den im Menüfenster anzuzeigenden Ausschnitt von der Liste der verfügbaren Optionen durch das Verschieben des Fensterinhalts wählen und somit auf alle Bereiche des Menüs zugreifen.

Während der Interaktion mit dem Smalltalk-80 System werden standardmäßig zwei Fenster auf dem Bildschirm dargestellt. Der *Arbeitsbereich* (work space) dient zur Erstellung von Benutzereingaben und der Protokollierung der Systemausgaben. Das zweite Fenster (*system browser*) macht dem Benutzer die Objekte des Systems verfügbar. Es können die Definitionen von Klassen angesehen und verändert werden. Der Benutzer kann in dem Arbeitsbereich Nachrichten erstellen, diese an Objekte senden und deren Rückmeldung empfangen. Zum Testen und Erproben gibt es Objekte, die Fehlermeldungen protokollieren sowie eine Inspektion und Modifikation von Objekten erlauben.

Die Implementation des Smalltalk-80 Systems basiert auf der Definition einer speziellen virtuellen Maschine [*Ingalls 81*]. Ein Betriebssystem existiert in einem Smalltalk-80 System nicht. Die üblichen Komponenten eines Betriebssystems wurden in die Sprache integriert. Die Speicherverwaltung erfolgt automatisch. Das Dateisystem besteht aus Objekten wie "Datei" und "Inhaltsverzeichnis" mit Nachrichtenprotokollen zum Zugriff. Die Bildschirmhandhabung erfolgt durch eine Instanz der Klasse "Form", die immer sichtbar ist und Nachrichten zur Manipulation ihrer graphischen Darstellung erhält.

Während des Entwurfs von Smalltalk wurde frühzeitig das Problem der Modi beim Umgang mit interaktiven Systemen erkannt und vermieden [*Tesler 81*]. Die modusvermeidende Interaktion wird dadurch unterstützt, daß Smalltalk eine Kommunikation des Benutzers mit mehreren Teilprozessen im Zeitmultiplex von einem einzigen Terminal aus gestattet. Weiterhin wird das Prinzip der überlappenden Fenster zur Interaktion und Informationsdarstellung angewendet. Die Fenster werden dabei als Hilfsmittel zur Abwicklung und graphischen Verdeutlichung des Interaktionsvorganges benutzt.

2.4 Das Cedar-System

Die Cedar-Programmierumgebung [*Teitelman 85*] ist eine interaktive graphische Umgebung. Sie verwendet zur Interaktion ein hochauflösendes Rastergraphik-System und eine Maus als Zeigeinstrument. Sie bietet einen komfortablen Editor, ein System zur Erzeugung von Dokumenten und eine Vielzahl von Hilfsmitteln, die den Benutzer bei der Konstruktion und Erprobung seiner Programme unterstützen. Diese experimentelle Programmierumgebung basiert auf der Cedar-Programmiersprache, die der PASCAL-Familie angehört. Die Cedar-Sprache enthält ein Typenkonzept mit strenger Typenprüfung und ist übersetzerorientiert, d.h. sie wurde als Sprache konzipiert, die übersetzt werden muß.

Die Cedar-Programmiersprache ist aus der Sprache Mesa entstanden. Sie unterstützt den Programmierstil von Lisp, Mesa und Smalltalk. Die Programmierumgebung basiert auf den Ideen und Techniken von Interlisp und Smalltalk.

Die Interaktion mit dem Benutzer wird über den Bildschirm abgewikkelt. Dazu kann man eine Maus verwenden, die drei Knöpfe besitzt. Die Komponenten des Systems werden durch kompakte graphische Symbole repräsentiert, auch *Piktogramme* (icons) genannt. Zur Kommunikation mit dem Benutzer und zur Darstellung von Information werden *Fenster* (viewers) verwendet. Der Benutzer kann ein Piktogramm aktivieren, was zur Folge hat, daß die dem Piktogramm zugehörige Information in einem Fenster dargestellt wird. Bei den Fenstern können drei Typen unterschieden werden. Der eine Typ (*tool*) ermöglicht einen Zugriff auf Hilfsmittel des Systems, der zweite (*button*) dient allein der Aktivierung einer einzigen Systemfunktion und der dritte (*whiteboard*) kann wieder weitere Fenster und Text enthalten.

Eine Komponente des Systems (*Tioga*) erlaubt das Editieren jeglicher textueller Information und das Erstellen von Textdokumenten. Alle Benutzereingaben werden auf Schreibfehler kontrolliert und in Anlehnung an DWIM (siehe Kapitel 2.2) korrigiert.

Für eine Teilmenge der Programmiersprache steht zusätzlich zu dem Übersetzer ein Interpretierer zur Verfügung. Die Cedar-Sprache enthält eine automatische Speicherverwaltung und die Unterstützung für "lisp-ähnliche" Listen und Atome. Das Prüfsystem ist quellsprachen-orientiert und gestattet das Setzen von Haltepunkten und die Inspektion des Programmzustandes.

Die Interaktion des Benutzers mit den betroffenen Systemkomponenten ist an die verwendeten Fenster gebunden. Diese Fenster können gleichzeitig dargestellt werden. Das System gestattet die parallele Durchführung von Operationen. Der Benutzer findet durch die graphische Darstellung entsprechender Fenster eine Unterstützung bei der Überwachung der parallelen Operationen. Die Qualität der Interaktion wird durch den hohen Grad der Integration der Systemkomponenten und die einheitliche Benutzerschnittstelle wesentlich gefördert.

2.5 Das Xerox Star System

Das Xerox Star 8010 System [*Lipkie et al. 82*] stellt einen Arbeitsplatz für die Textverarbeitung dar. Es verfügt über einen integrierten Text- und Graphikeditor. Das Gerät selber besteht aus einem Prozessor, einem 10–40 Mbyte Plattenspeicher, einem hochauflösenden Rastergraphik-Bildschirm, einer Tastatur mit drei Funktionsgruppen und einer Maus mit zwei Knöpfen. Das System kann an ein lokales Netzwerk (Ethernet) angeschlossen werden, über das beispielsweise die erstellten Dokumente zum Drucken gesandt werden können.

Auffallend an dem Xerox Star System ist u.a. die Tatsache, daß etwa 30 Mannjahre nur zum Entwurf der Benutzerschnittstelle aufgewendet wurden. Dies zeigt, welche Bedeutung man damals bereits einer Gestaltung der Benutzerschnittstelle beimaß. Die Konzeption der Benutzerschnittstelle wurde vor der ersten Realisierung des Systems durchgeführt. Dafür wurde ein Modell des Benutzers entworfen, welches beschreibt, wie sich der Benutzer gegenüber dem System verhält.

Bei dem Entwurf der Benutzerschnittstelle wurden nur wenige Prinzipien verwendet [*Smith et al. 82*]:

- Das System bietet dem Benutzer ein vertrautes konzeptuelles Modell;
- Sehen und Zeigen gegenüber Erinnern und Schreiben;
- Was man sieht, kann bearbeitet werden;
- Verwendung der Metapher "reale Bürowelt";
- Benutzerfreundlichkeit;
- Universelle Kommandos;
- Modusvermeidende Interaktion;

- Konsistenz;
- Einfachheit.

Das System stellt dem Benutzer ein graphisches Schema der Bürowelt und der Schreibtischoberfläche dar. Die dort repräsentierten Komponenten können mithilfe der Maus manipuliert werden.

Die Komponenten des Systems werden in Form von Piktogrammen graphisch dargestellt. Durch Selektion dieser Piktogramme mit der Maus können diese aktiviert werden. Dazu kann auf dem Bildschirm ein Fenster eröffnet werden, das zur genauen Darstellung der Systemkomponente und zur weiteren Interaktion mit dem Benutzer dient. Nach Auswahl einer Komponente kann diese mithilfe der universellen Kommandos bearbeitet werden. Die genaue Bedeutung der universellen Kommandos wird durch die Komponenten definiert, auf die sie angewendet werden.

Xerox Star verwendet für die Operationen des Benutzers das Konzept "Editieren". Neben dem Editieren von Text, Graphik und Formeln kann die Verwaltung von Dateien durch Editieren von Dateifenstern erfolgen. Die Anordnung der Arbeitsumgebung vollzieht sich durch das Editieren der Schreibtischoberfläche. Eigenschaften von Komponenten können durch Editieren von Merkmalsfenstern verändert werden.

Das Konzept der "Informationserfassung" wird durch Datenbasen realisiert. Ordner werden als Dokumentendatenbasis, Eingangskörbe für Post als Nachrichtendatenbasis dargestellt.

Bei der Erstellung von Dokumenten wird das Konzept "Erzeugen durch Kopieren" verwendet. Der Benutzer kann aus vorgegebenen graphischen Elementen seine Dokumente erstellen, indem er diese Elemente in sein Dokument kopiert.

Die modusvermeidende Interaktion wird dadurch unterstützt, daß Kommandos die Struktur *Objekt — Aktion* haben. Der Benutzer selektiert erst das zu bearbeitende Objekt und wählt dann die darauf anzuwendende Operation aus. Der Editor unterscheidet nicht zwischen verschiedenen Modi. Ein Überladen von Tasten mit mehreren modusabhängigen Bedeutungen wurde weitgehend vermieden.

Die konsistente Gestaltung des Systems und seiner Benutzerschnittstelle war eines der Grundprinzipien bei dem Entwurf. Diese Konsistenz garantiert dem Benutzer, daß er Operationen unabhängig von ihrem Auftreten gleichartig verwenden kann. An einem Beispiel, dem Ausdrucken eines Dokumentes, kann gezeigt werden, daß dieses Ziel nicht leicht erreichbar ist.

Der Benutzer kann ein Dokument ausdrucken, indem er dessen Piktogramm als Quelle auswählt, das universelle Kommando "MOVE" oder "COPY" betätigt und das Drucker-Piktogramm als Senke bezeichnet. Der Drucker akzeptiert das Dokument und trägt es in seine Warteschlange ein. Wählt der Benutzer die Operation "COPY", so wird eine Kopie des Dokumentes dem Drucker übergeben. Damit bleibt das Originaldokument von dem Druckvorgang und seinen möglichen Konsequenzen unberührt. Dieses wird dem Benutzer auch graphisch verdeutlicht, indem das Dokument-Piktogramm an seinem Platz verbleibt und eine Kopie dieses Piktogramms in der Druckerwarteschlange erscheint.

Verwendet der Benutzer das Kommando "MOVE", so wird das eigentliche Dokument und nicht eine Kopie in die Warteschlange des Druckers übertragen. Dem Benutzer wird dieser Vorgang ebenfalls graphisch dargestellt, indem das entsprechende Piktogramm von seiner vorherigen Position verschwindet und in der Druckerwarteschlange auftaucht. Das Konsistenzproblem tritt bei der Frage auf, was mit dem Dokument und damit mit dem Piktogramm nach Beendigung des Druckvorganges geschehen soll. Es ergeben sich zwei Alternativen:

1. Das System löscht das Piktogramm.
2. Das System löscht das Piktogramm nicht, daraus ergibt sich:
   a) Das System bringt das Piktogramm an die Stelle zurück, wo es vor dem Drucken angeordnet war.
   b) Das System alloziert das Piktogramm an einer beliebigen Stelle der Schreibtischoberfläche.
   c) Das System beläßt das Piktogramm im Drucker. Der Benutzer muß es explizit zurückholen.

Das Argument zur Auswahl der ersten Alternative lautet, daß das Kommando "MOVE" mit derselben Bedeutung wie bei anderen Tätigkeiten angewendet wird. Als Beispiele seien das Versenden von Nachrichten (mail) und das Eintragen eines Dokumentes in einen Ordner (directory) genannt. Beim Versenden einer Nachricht wird das entsprechende Piktogramm in einem speziellen Ausgangskorb für "Post" plaziert. Das System übermittelt die Nachricht und löscht sie dann aus dem Arbeitsbereich des Benutzers. Analog wird das Eintragen in einen Ordner behandelt. Deshalb sollte bei der Übertragung eines Piktogramms durch "MOVE" in das Drucker-Piktogramm das Dokument gedruckt und anschließend aus dem Arbeits-

bereich gelöscht werden. Dadurch wird eine konsistente Anwendung von "MOVE" erreicht.

Das Argument zur Auswahl der zweiten Alternative wird durch die zugrunde liegende Metapher der Bürowelt geliefert. Der Benutzer hat die Vorstellung, daß er die Piktogramme ähnlich wie ihre realen Ebenbilder aus dem Büro behandeln kann. Deshalb ist es vernünftig, wenn Nachrichten nach dem Versenden gelöscht werden, da ein mit der Post versandter Brief ebenfalls nicht mehr zur Verfügung steht. Als Ebenbild des Druckers ist jedoch der Bürokopierer anzusehen, und der vernichtet üblicherweise nicht die zu kopierenden Dokumente. Deshalb ist es sinnvoll, ein Piktogramm nach dem Drucken nicht zu löschen.

Beide Argumente erscheinen plausibel, und jedes betont einen Bereich der Konsistenz von Xerox Star. Zur Lösung dieses Dilemmas wurde eine Gewichtung der Konsistenzkriterien vorgenommen. Dabei wurde der Metapher von der Bürowelt und der Analogie mit den Ebenbildern aus dem Büro Vorrang gegeben. Die erste Alternative befürwortet ein implizites Modell, das der Benutzer erst lernen muß, während die zweite das explizite Modell verwendet, das der Benutzer bereits aus seinen Erfahrungen aus dem Bürobereich besitzt. Die Wahl der zweiten Alternative wird auch durch pragmatische Erwägungen unterstützt. Etwas zu löschen ist generell gefährlich, wenn ein Benutzer dieses nicht erwartet. Weiterhin ist bei den oben angeführten Beispielen (Versenden einer Nachricht und Eintragen in einen Ordner) das Dokument nicht endgültig verschwunden, während das Löschen des Dokumentes nach dem Drucken endgültig wäre. Es wurde Alternative 2a gewählt, da diese für den Benutzer den geringsten Arbeitsaufwand bedeutet.

An diesem Beispiel wird deutlich, daß es bei der Realisierung eines interaktiven Systems keine einheitlichen und absoluten Konsistenzkriterien geben kann, sondern daß häufig verschiedene Kriterien miteinander konkurrieren und ihre Einhaltung im Einzelfall sorgfältig abgewogen werden muß.

Ein einfaches System ist vorteilhaft, da Benutzer dieses leichter verstehen können. Ein System wird u.a. einfach, wenn die Redundanz minimal bleibt. Deshalb verwendet Xerox Star universelle Kommandos. Diese allgemeinen Mechanismen werden oft aber dann unhandlich, wenn sie häufig anzuwenden sind. Ein Beispiel dafür ist die Veränderung des Aussehens eines Zeichens (Fettdruck, Kursiv, Index). Diese erfolgt durch im System allgemein verwendete Merkmalsfenster. Bei der Eingabe eines Dokumentes

würde dieser Vorgang aber den Benutzer in seiner Tätigkeit zu stark unterbrechen. Deshalb wurden Kurzformen durch spezielle Tasten geschaffen.

Die Art der Benutzerschnittstelle von Xerox Star und das darin enthaltene Benutzermodell haben dazu geführt, daß bei der Programmentwicklung von Anfang an ein objektorientierter Ansatz gewählt wurde.

Anfangs wurde ein Stil verwendet, der sich an den Konzepten der Programmiersprachen SIMULA und Smalltalk orientierte. Dabei ist die Bildung von Unterklassen eine der wichtigsten Methoden, um komplexere Objekte durch die Verwendung von einfacheren Objekten zu strukturieren. Diese Spezialisierungsrelation ist in SIMULA und Smalltalk baumartig geordnet. Für Xerox Star wurde jedoch dieser Ansatz verallgemeinert, so daß die Spezialisierungsrelation in Form eines azyklischen gerichteten Graphen dargestellt werden kann [*Lipkie et al. 82*].

Neben dem Xerox Star System gibt es noch zwei andere Systeme, die auf ähnlichen Prinzipien basieren. Das Apple Lisa [*Williams 83*] und das Apple MacIntosh-System [*Williams 84*] verwenden ebenfalls die Metapher der Bürowelt für die Textverarbeitung. Diese Systeme haben viele Konzepte von Xerox Star übernommen und eine Reihe von Weiterentwicklungen durchgeführt. Sie unterscheiden sich im Vergleich zu Xerox Star u.a. dadurch, daß die verwendeten graphischen Symbole detaillierter sind und dreidimensional wirken [*Marcus 84*].

2.6 Folgerungen für die Auswertung von Bildfolgen

Den in dieser Arbeit vertretenen Ansatz haben zwei der oben dargestellten Systeme entscheidend geprägt. Das Xerox Star System kann als beispielhaft für den Entwurf einer benutzergerechten Schnittstelle angesehen werden. Das Smalltalk-System demonstriert die Vorteile bei der Realisierung eines Programmsystems, die aus der konsequenten Verwendung eines objektorientierten Ansatzes resultieren.

Die Konzeption einer Interaktionsumgebung für die Bildfolgenauswertung erscheint erfolgreicher, wenn es gelingt, die Entwurfsprinzipien von Xerox Star auf den Bereich der Bildfolgenauswertung zu übertragen. Der Erfolg der Benutzerschnittstelle von Xerox Star liegt in der sorgfältigen Planung begründet, die vor der Realisierung erfolgte.

Für den Anwendungsbereich der Bildfolgenauswertung bedeutet dies vor allem, daß für die Konzeption der Benutzerschnittstelle ein Benutzermodell formuliert wird. Dieses Benutzermodell ist dann bei dem Entwurf der Schnittstelle zu berücksichtigen. Die von Xerox Star und Apple MacIntosh verwendeten Interaktionskonzepte wie die Manipulation von graphisch dargestellten Objekten und die an diesen Objekten orientierte Interaktionsgestaltung erscheinen für die Bildfolgenauswertung mit einem ähnlichen Erfolg anwendbar.

Eine Benutzerschnittstelle, die auf der Interaktion mit Objekten basiert, kann nur dann überzeugend wirken, wenn sich die Benutzerschnittstelle auf ein Bildfolgenauswertesystem stützt, das ebenfalls objektorientiert gestaltet ist. Das Smalltalk System hat die Durchführbarkeit eines auf Objekten basierenden Ansatzes bewiesen. Es unterscheidet sich jedoch von der hier beabsichtigten Zielsetzung dadurch, daß es eine eigene Sprache definiert hat und als allgemeine Programmierumgebung konzipiert wurde.

Das Unix-System hat gezeigt, wie leicht flexible, modulare Programmsysteme mithilfe von Filtern und Datenleitungen konstruiert und interaktiv genutzt werden können. Dieses Konzept läßt sich prinzipiell auf Objekte (in Unix als Filter) zurückführen, die über Nachrichten (in Unix eingeschränkter als Datenleitungen) miteinander kommunizieren. Damit können diese Konzepte von Unix als eine spezielle Anwendung des von Smalltalk vertretenen Ansatzes angesehen werden. Für bestimmte Teilbereiche der Bildauswertung wurde schon gezeigt, daß die Konzepte von Unix durchaus vielversprechend angewendet werden können [*Stevens & Hunt 82, Petters 85*]. Deshalb soll in dieser Arbeit versucht werden, das Konzept der Datenleitungen und Filter in die geplante Interaktionsumgebung zu integrieren.

Das Interlisp System hat als eines der ersten Systeme dem Benutzer eine Historie über den Interaktionsablauf angeboten. Weiterhin wird die Interaktion durch eine automatische Korrektur von Schreibfehlern in den Benutzereingaben erleichtert. Die leichte Erweiterbarkeit des Systems und die Möglichkeit zur flexiblen Reaktion auf Fehlersituationen durch den Benutzer haben sich ebenfalls als hilfreich erwiesen. Eine Fortführung dieser Konzepte unter Verwendung von Fenstertechniken findet sich in dem Cedar-System. Für die Bildfolgenauswertung bleibt zu prüfen, inwieweit sich diese Konzepte sinnvoll übertragen lassen.

Die hier vorgestellten Systeme werden als Motivation angesehen, eine Interaktionsumgebung für die Bildfolgenauswertung ausgehend von den hier dargelegten Konzepten zu entwerfen.

Im nachfolgenden Kapitel wird deshalb die Interaktion mit einem Bildfolgenauswertesystem auf der Basis eines Benutzermodells konzipiert.

Kapitel 3

Interaktion ausgehend von einem Benutzermodell

Der erfolgreiche Entwurf einer Benutzerschnittstelle für ein System setzt voraus, daß klare Vorstellungen von Eigenschaften und Intentionen des zukünftigen Benutzers vorliegen, d.h. daß der zukünftige Benutzer dieses Systems modelliert wird. Dieses Benutzermodell wird dann als Grundlage verwendet, um die Interaktion zwischen dem Benutzer und dem geplanten System zu gestalten. Ein wichtiger Schritt bei der Modellierung des Benutzers ist die Formulierung eines konzeptuellen Modells, das der zukünftige Benutzer von einem System hat. Dieses *konzeptuelle Modell* wird folgendermaßen definiert [*Smith et al. 82*].

> Es besteht aus einer Menge von Konzepten, die eine Person bildet, um das Verhalten eines Systems zu erklären. Diese Konzeptbildung ist davon unabhängig, ob es sich bei diesem System um einen Rechner, ein physikalisches oder ein hypothetisches System handelt. Dieses vom Benutzer entwickelte Modell erlaubt ihm, die Reaktion eines Systems zu antizipieren und mit diesem System zu kommunizieren.

Dieses konzeptuelle Modell von einem System wird hauptsächlich durch dessen Interaktionsnormen geprägt. Es kann deshalb auch als *Systemmodell* bezeichnet werden [*Maaß 84*]. Unter einem Systemmodell wird im Rahmen dieser Arbeit somit die Vorstellung verstanden, die der Benutzer vom System hat.

Als *Benutzermodell* werden hier die Vorstellungen des Systementwicklers bezeichnet, die dieser vom Benutzer und dessen Systemmodell hat.

Aus dem vom Entwickler formulierten Systemmodell werden Konzepte zur Gestaltung einer Benutzerschnittstelle entwickelt. Diese Gestaltungs-

konzepte explizieren das vom Systementwickler angestrebte Benutzermodell. Ihre Verwendung beim Entwurf eines Systems soll bewirken, daß im Benutzer bei der Interaktion mit dem System das angestrebte Systemmodell entsteht. Der Systementwickler setzt somit seine Erwartungen bezüglich der Art der Benutzereingaben und Systemausgaben in die Dialoggestaltung des Systems um.

Diese vom Entwickler im System niedergelegten Erwartungen werden als Benutzerbild des Systems [*Maaß 84*] angesehen. Das *Benutzerbild* bezeichnet im Rahmen dieser Arbeit die Vorstellungen über den Benutzer, die der Systementwickler im System verankert hat. Das Benutzerbild ist damit das Benutzermodell des Systementwicklers, das sich im System wiederfindet und vom Benutzer nur indirekt abgeleitet werden kann. Das Benutzerbild eines Systems äußert sich in den Interaktionsnormen, die der Systementwickler setzt [*Maaß 84*].

Die Gestaltung einer Benutzerschnittstelle ist erfolgreicher, wenn es gelingt, die Vorstellungswelt des zukünftigen Benutzers als Benutzerbild in der Schnittstelle zu manifestieren. Das sich bildende Systemmodell des Benutzers läßt sich dann leicht in seine bisherige Vorstellungswelt integrieren und erleichtert ihm den Umgang mit dem System.

Durch eine *Metapher* kann dieser Prozeß zur Bildung eines Systemmodells erheblich unterstützt werden [*Carroll & Thomas 82*]. Diese Theorie sagt aus, daß Benutzer neue kognitive Strukturen lernen, indem sie schon vorhandene metaphorisch erweitern. Bei der interaktiven Nutzung von Rechensystemen vollzieht sich dieses Lernen vorwiegend durch deren Benutzung.

Liegt einem System eine gut gewählte Metapher zugrunde, so wird es dem Benutzer erleichtert, sich ein Systemmodell zu bilden. Diese Metapher sollte sich an Strukturen orientieren, die den Benutzern von ihren bisherigen Erfahrungen schon bekannt sind. Die Metapher kann dann dem Benutzer das Bilden einer gedanklichen Verbindung zwischen der Benutzerschnittstelle und der realen Benutzerumgebung erleichtern. Diese metaphorische Verbindung kann bewirken, daß das die Arbeitsumgebung modellierende System für den Benutzer zur Realität wird und damit seine vorherige Arbeitsumgebung ersetzt.

Beispiele für Systeme, denen eine derartige Metapher bei der Systemgestaltung zugrunde gelegt wurde, sind Xerox Star, Apple Lisa und MacIntosh. Diese Systeme für Textverarbeitung sind für "unerfahrene" Benutzer von Textsystemen konzipiert. Die dort verwendete Metapher wurde

deshalb aus dem Erfahrungsbereich der Bürobenutzer gewählt. Die Benutzerschnittstelle liefert das Bild einer Schreibtischoberfläche und bietet Operationen zur Manipulation an, die sich an den vorhandenen Tätigkeiten aus der bisherigen Arbeitsumgebung (Schreibtisch) orientieren.

Im folgenden wird hier der Versuch unternommen, eine Metapher zu formulieren, die dem Benutzer und Entwickler von Bildfolgenauswertesystemen den Umgang mit einer entsprechend konzipierten Benutzerschnittstelle erleichtern soll.

Wie im Bürobereich spielen auch bei der Bildfolgenauswertung die dort verwendeten Geräte eine wichtige Rolle. Sie bilden einen Vorrat an Werkzeugen, aus denen ein Benutzer die zur Durchführung seines Experimentes notwendigen Geräte auswählt. Diese Geräte sind dann als Betriebsmittel ein Bestandteil seines Experimentalsystems. Dieses besteht weiterhin aus einem Programmsystem, welches unter Verwendung der gewählten Betriebsmittel und unter der Kontrolle des Benutzers die Bildfolgenanalyse durchführt.

Das Spektrum der Bildfolgenauswertung erstreckt sich vom Bereich der Bildfolgenerfassung über die Bildvorverarbeitung bis hin zur 3D-Modellierung von Objekten und der Interpretation von Änderungen. Ein dieses Aufgabengebiet umfassendes System wird deshalb in natürlicher Weise aus Komponenten bestehen, die jeweils eine Teilaufgabe des Gesamtsystems realisieren. Die Interaktion des Benutzers mit dem System findet auf der Basis der vorhandenen Komponenten statt. Sie sollen deshalb auch als *Interaktionskomponenten* bezeichnet werden.

Für den Bereich der Bildfolgenauswertesysteme sollen in diesem Abschnitt die Erkenntnisse aus Kapitel 2 für den Entwurf einer Benutzerschnittstelle berücksichtigt werden.

Zuerst wird deshalb eine Aufgabenanalyse durchgeführt. Dabei sind die zukünftigen Benutzer zu klassifizieren, ihre Ziele zu beschreiben, die von ihnen benutzten und erzeugten Informationen zu identifizieren und die von ihnen verwendeten Methoden zu erfassen. Diese Aufgabenbeschreibung umfaßt die verwendeten Informationsstrukturen und Methoden. Sie dient als Ausgangspunkt für eine neu zu schaffende Aufgabenumgebung des Benutzers, in der er wie vorher seine Aufgaben erledigen kann, die aber neue Objekte und Methoden anbietet.

Im Rahmen dieser Aufgabenanalyse sollen zuerst mögliche Interaktionskomponenten von Bildfolgenauswertesystemen untersucht werden. Danach folgt eine Beschreibung von Benutzerzielen während der Interaktion. Ab-

schließend wird ein Benutzerbild entwickelt, das sich in der Aufstellung von Konzepten für die Gestaltung einer zukünftigen Benutzerschnittstelle äußert.

3.1 Interaktionskomponenten in der Bildfolgenanalyse

Die in Bildfolgenauswertesystemen denkbaren Komponenten können grob in zwei Klassen unterteilt werden. Die eine Klasse enthält alle möglichen Geräte, die speziell der Bildverarbeitung dienen. In der zweiten finden sich alle Programmkomponenten wieder. Diese zeichnen sich in der Regel durch ein abgegrenztes Aufgabengebiet aus, das sie in einem bestimmten Rahmen bearbeiten.

Bei den Programmkomponenten braucht die Notwendigkeit zur Interaktion mit dem Benutzer nicht näher ausgeführt werden. Die Klasse der Geräte gehört ebenfalls in den Bereich der Interaktionskomponenten, da deren Nutzung in der Bildverarbeitung oft experimenteller Natur ist. Sie besitzen Parameter, die während eines Verarbeitungsvorganges vom Benutzer interaktiv veränderbar sein müssen.

3.1.1 Periphere Geräte

Die peripheren Geräte spielen für die Bildfolgenanalyse eine zentrale Rolle. Die meisten der verwendeten Geräte zeichnen sich dadurch aus, daß sie von dem Benutzer bzw. von dem Bildfolgenauswertesystem programm- oder ferngesteuert werden können. Diese Fernsteuerung wird üblicherweise von dem Programmsystem durchgeführt und von dem Benutzer interaktiv kontrolliert. Nun kann es aber auch periphere Geräte geben, bei denen eine Fernsteuerung kaum oder gar nicht möglich ist. Das Programmsystem ist in diesem Fall nicht in der Lage, eine Steuerung derartiger Geräte im Auftrag des Benutzers durchzuführen.

Die diesen Geräten zugeordneten Systemkomponenten werden sich deshalb an den Benutzer wenden, damit er die Bedienung der Geräte manuell ausführt. Das System sollte den Benutzer dabei unterstützen, ihm den gewünschten Zustand mitteilen und eine Hilfestellung für den Umgang mit derartigen Geräten geben.

Im folgenden werden einige repräsentative Beispiele für periphere Geräte vorgestellt und ihre Rolle im Systemmodell des Benutzers beschrieben.

Bildsignalquellen und -senken

Sehr viele Geräte werden auf Grund ihrer Fähigkeit, Bildsignale zu verarbeiten, von dem Benutzer als Quelle oder Senke eines derartigen Signals betrachtet. Aus der Eigenart dieser Geräte ergibt sich die Möglichkeit der schaltnetzartigen Verbindung von Quellen und Senken. Um die Geräte vielfältig nutzen zu können, ist eine möglichst flexible Verknüpfungskomponente wünschenswert. Gute Beispiele für Geräte als Signalquellen bzw. -senken sind:

- Kamera
  Eine S/W- oder Farbkamera dient als reine Signalquelle. Die ferngesteuerte Bewegung der Kamera sowie die Entfernungs- und Brennweiteneinstellung sind Operationen, die häufig noch eine Interaktion mit dem Benutzer notwendig machen.

- Bildplatte, Bildspeicher
  Die Bildplatte (analog oder digital) dient als Hintergrundspeicher für Bildfolgen und ist als schnelles Speichermedium für die Aufzeichnung von Bildfolgen unentbehrlich. Durch die Speicherfähigkeit fungieren diese Geräte als Quellen und Senken. Die Kapazität von Bildspeichern ist meistens gering gegenüber Bildplatten. Sie werden zur Darstellung und Manipulation von Bildern und Graphiken benötigt.

  Je nach Art und Komplexität dieser Geräte sind unterschiedliche Interaktionen mit dem Benutzer denkbar. Diese reichen von Aufträgen wie Aufzeichnen, Wiedergeben, Vorwärts- und Rückwärtslauf, Standbild, Einzelbildbetrieb bis zu Such- und Verwaltungsoperationen, wie sie von herkömmlichen Dateisystemen bekannt sind. Eine konsequente Fortentwicklung äußert sich in speziellen Bilddatenbanksystemen, die dem Benutzer statt eines einfachen Bildspeichers eine problemorientierte Speicherung seiner Daten gestatten (siehe auch Kapitel 3.1.2). Die Verwendbarkeit und Kontrolle dieser Geräte durch den Benutzer wird durch einen interaktiven Zugang erheblich gesteigert.

- Monitor
  Der S/W- oder Farbmonitor ist die einfachste Form einer Signalsenke. Er wird zur Darstellung verwendet. Sie spielen eine wichtige Rolle, da

sie meistens als Medium für graphische Interaktionen benutzt werden. Bei derartigen Geräten könnte der Benutzer interaktiv Darstellungsparameter wie Helligkeit, Kontrast, Farbsättigung usw. setzen.

- Verknüpfung von Signalquellen und -senken
  In einer experimentellen Laborumgebung mit einer vielfältigen Peripherie für die Bildverarbeitung ist es sinnvoll, eine möglichst flexible Verknüpfung von Bildsignalen zu erlauben. Ein Beispiel dafür ist der programmgesteuerte Kreuzschienenverteiler für Video-Signale im Labor des Arbeitsbereichs "Kognitive Systeme" (KOGS). Die Interaktion des Benutzers mit derartigen Geräten wird zum Schalten oder Abfragen von Signalwegen notwendig.

Graphische Ein- und Ausgabegeräte

Unter graphischen Ein- und Ausgabegeräten sollen nicht nur Monitore, sondern eine Synthese von Bildverarbeitungssystemen oder Bildspeichern in Verbindung mit Zeigeinstrumenten wie Rollkugel, Lichtgriffel oder Maus und einem Monitor zur Darstellung verstanden werden.

Die Komplexität derartiger Systeme reicht von einfachen Bildspeichern über Systeme mit Graphiküberlagerung und Funktionstabellen bis hin zu komplexen Bildverarbeitungssystemen mit speziellen Verarbeitungsprozessoren.

Die Möglichkeiten der Interaktion mit derartigen Geräten hängen stark von deren Komplexität ab. Denkbar sind Operationen wie die Inspektion und Änderung des Gerätezustandes, der Funktionstabellen und die Darstellungsschaltung von Bild- und Graphikspeichern.

Spezielle Prozessoren

Diese für besondere Aufgaben dedizierten Systeme können völlig unabhängig zur Verfügung stehen oder in einem Bildverarbeitungssystem integriert sein. Die mit diesen Systemen nötigen Interaktionsformen hängen von deren Struktur ab.

Ein Beispiel für derartige Prozessoren, die in einem Bildverarbeitungssystem integriert sind, ist das Modulare Bildverarbeitungssystem der Fa. VTE. Dieses enthält u.a. arithmetisch-logische Einheiten, die unter Verwendung von Funktionstabellen Operationen wie Schwellwertbildung und Addition von Vollbildern in Echtzeit ermöglichen.

Eine Studie hat gezeigt, daß gerade bei komplexen Geräten wie das Modulare Bildverarbeitungssystem eine graphische Zustandsdarstellung das experimentelle Kennenlernen wesentlich vereinfacht [*Mansfeld 84*].

3.1.2 Datenverwaltung

Die Datenverwaltung spielt eine zentrale Rolle bei der Bildfolgenauswertung. Die dabei anfallenden Datenmengen sind sehr umfangreich und können oft wegen des begrenzten Hintergrundspeichers nur teilweise gehalten werden. Bei den für die Datenverwaltung zur Verfügung stehenden Geräten handelt es sich um konventionelle Magnetplattenspeicher und spezielle Bildspeicher.

Ein interaktiver Zugang im Rahmen eines Bildfolgenanalysesystems ist notwendig, da der Benutzer während eines Experimentes die Möglichkeit haben muß, den Datenbestand zu inspizieren. Es sind dabei die von Dateisystemen bekannten Verwaltungs- und Suchoperationen denkbar. Der Umfang und die Funktionalität der zur Verfügung stehenden Operationen hängt von der Leistungsfähigkeit der Datenverwaltung ab. Im Falle einer relationalen Bildfolgenbank [*Benn & Radig 84*] ist es denkbar, daß dem Benutzer tupel- und inhaltsbezogene Operationen zum interaktiven Zugriff auf die Daten der Bildfolgenbank angeboten werden.

3.1.3 Programmkomponenten

Unter den Programmkomponenten eines Systems für die Bildfolgenauswertung werden in diesem Zusammenhang Einheiten verstanden, die durch ein abgegrenztes Aufgabengebiet in dem Gesamtsystem charakterisiert sind. Als Beispiele seien Komponenten aus dem MORIO-System [*Dreschler-Fischer et al. 83*] wie Objektmasken, Punktfinder usw. genannt.

Diese Komponenten bilden oft ihre Eingangsdatenstrukturen in entsprechende Ausgangsstrukturen ab. Die auf den Daten durchgeführten Operationen stellen meistens einen signifikanten Verarbeitungsschritt in dem Gesamtsystem dar.

Eine Interaktion mit dem Benutzer ist oft notwendig, um die Parameter des Verarbeitungsvorganges festzulegen. Weiterhin sind Protokollausgaben über den Fortgang der Verarbeitung üblich.

3.2 Interaktionsziele

Im folgenden sollen Ziele des Benutzers bei der Interaktion mit dem Bildfolgenauswertesystem dargestellt werden. Dabei lassen sich zwei Ziele bei der Interaktion unterscheiden. Das erste Ziel der Benutzerinteraktion ist das *Gewinnen von Information.* Diese teilt sich auf in Information über den *Systemzustand* und über die *möglichen* Veränderungen des Systemzustandes (Hilfestellung).

Das zweite Ziel der Interaktion ist die *Steuerung* des Systems. Diese Steuerung umfaßt den *Ablauf*, den *Datenfluß* und die *Verarbeitungsparameter.*

3.2.1 Systemzustand

Eine wichtige Aufgabe ist das Gewinnen von Information über den Systemzustand. Dieser Systemzustand setzt sich aus den Zustandsinformationen der einzelnen Komponenten und der Gesamtverwaltung zusammen.

Die zur Verfügung stehenden Daten für die Zustandsdarstellung einer Systemkomponente sind von der jeweiligen Anwendung abhängig und daher kaum vorhersehbar.

Es wird davon ausgegangen, daß Bildfolgenauswertesysteme aus einer Vielzahl von Komponenten bestehen. Die zur Verfügung stehenden Daten für den Systemzustand sind vermutlich sehr umfangreich. Eine permanente, vollständige Darstellung des gesamten Systemzustandes ist deshalb unzweckmäßig. Der Benutzer muß die Möglichkeit haben, bestimmte Komponenten zur ausführlichen Zustandsdarstellung auszuwählen, während bei anderen nur eine Kurzbeschreibung oder keine Informationsdarstellung erforderlich sind.

Bei der Beobachtung des Systemzustandes durch den Benutzer sind zwei wichtige Ziele erkennbar. Zum einen will der Benutzer als Experte den Fortlauf eines Experimentes überwachen und durch die Inspektion seines Systems die zu klärenden Hypothesen verifizieren. Bei dieser Form der Beobachtung finden keine grundlegenden Eingriffe wie beispielsweise die Umgestaltung des Systems statt. Die Interaktion mit den Komponenten erfolgt, um Ergebnisse zu dokumentieren oder Verarbeitungsparameter zu ändern.

Ein weiteres wichtiges Ziel ergibt sich beim Fehlverhalten des Systems, welches nur mit einer Fehlersuche behoben werden kann. Dafür muß die

Benutzerschnittstelle Hilfsmittel bereitstellen, die den Benutzer bei der Fehlersuche unterstützen. Um eine Fehlerursache zu lokalisieren und das Fehlverhalten zu erklären, benötigt der Benutzer wesentlich ausführlichere Zustandsinformationen als sonst üblich. Die Zustandsdarstellungen enthalten u.a. Zwischenergebnisse, Daten- und Ablaufprotokolle. Diese Form der Fehlersuche findet auf einer problemorientierten Abstraktionsebene statt und sollte in der Regel nicht durch die Implementationssprache des Bildfolgenauswertesystems beeinflußt werden.

Kann ein Fehler beispielsweise in einer Systemkomponente lokalisiert, aber die eigentliche Fehlerursache noch nicht festgestellt werden, so wird spätestens zu diesem Zeitpunkt ein Prüfsystem benötigt. Dieses sollte dem Benutzer die Möglichkeit geben, auf der Ebene der Implementationssprache diese Systemkomponente zu inspizieren und zu überprüfen. Diese Form der Fehlersuche findet auf einer implementationsnahen Abstraktionsebene statt und orientiert sich mehr an den zur Realisierung des Bildfolgenauswertesystems verwendeten Konzepten der Implementationssprache.

3.2.2 Hilfestellung

Eine interaktiv verfügbare Hilfestellung für den Benutzer sollte ein integraler Bestandteil eines Systems sein. Die über die Hilfestellung verfügbare Information kann in zwei Kategorien unterschieden werden:

1. Information über den Umgang mit der Benutzerschnittstelle.
2. Information über den Umgang mit Systemkomponenten, die sich dieser Schnittstelle bedienen.

Die erste Form der Hilfestellung muß dem Benutzer über die verfügbaren Operationen und Methoden der Benutzerschnittstelle Auskunft geben. Darin sind auch Konventionen enthalten, die der Benutzerschnittstelle zugrunde liegen und das Benutzerbild definieren. Diese Konventionen dienen einer einheitlichen Benutzeroberfläche und sind deshalb von den einzelnen Anwendungen unabhängig.

Die zweite Form der Hilfestellung ist anwendungsbezogen. Sie soll die von einer Systemkomponente definierten Operationen und die von ihr erzeugten Informationen erläutern. Deshalb muß diese Hilfestellung durch die befragte Komponente geleistet werden. Die Komponente hat dabei Basisfunktionen der Benutzerschnittstelle zu verwenden. Diese dem Entwerfer

der Verarbeitungskomponente angebotenen Basisoperationen garantieren eine systemkonsistente Interaktion.

3.2.3 Manipulation des Systems

Neben der reinen Abfragetätigkeit ist ein weiteres wichtiges Ziel der Benutzerinteraktion die Einflußnahme auf den Systemablauf. Diese Einflußnahme ist als eine Veränderung des Systemzustandes mit einer bestimmten Zielsetzung des Benutzers anzusehen. Traditionell lassen sich die folgenden Aktionen unterscheiden.

- Ablaufsteuerung
  Der Benutzer will die Reihenfolge und Art der Bearbeitung seiner Daten bestimmen. Dabei ergibt sich eine *globale* Ablaufsteuerung, die das Zusammenspiel der einzelnen Komponenten regelt, und eine *lokale* Ablaufsteuerung, die die Verarbeitung innerhalb einer Komponente bestimmt.

- Datenflußsteuerung
  Der Benutzer wählt die zu bearbeitenden Daten aus und legt den Datenfluß zwischen den einzelnen Komponenten fest. Dies ist ebenfalls eine Form der globalen Ablaufsteuerung, bei der durch das Wählen von Datenkanälen bestimmte Komponenten implizit in den Verarbeitungsvorgang einbezogen werden.

- Parameterkontrolle
  Die Zustandsdarstellung von Komponenten zeigt auch veränderbare Größen dieser Komponenten. Diese können vom Benutzer in Abhängigkeit von ihrer Struktur verändert werden. Sie stellen einen Satz von Parametern dar, der eine detaillierte Beeinflussung einer Komponente ermöglichen soll.

3.3 Konzepte zur Gestaltung der Schnittstelle

Nach der Beschreibung des konzeptuellen Modells des Benutzers und seiner Ziele bei der Interaktion werden im folgenden grundlegende Konzepte dargestellt, die bei der Gestaltung einer Benutzerschnittstelle oder bei der Beurteilung vorhandener Schnittstellen anzuwenden sind:

- Visuelle Darstellung des Systems
- Komponentendarstellung
- Objektorientierte Interaktion
- Objektgebundener Handlungskontext
- Direkte Manipulation
- Modusvermeidende Interaktion
- Reaktionsbereitschaft
- Konsistenz
- Einfachheit

Diese Konzepte finden sich zum Teil bereits in den in Kapitel 2 beschriebenen Systemen, wobei besonders die für den Bereich der Textverarbeitung entwickelten Systeme Xerox Star (siehe auch Kapitel 2.5) und MacIntosh [*Williams 84*] zur Ableitung der nachfolgenden Konzepte gedient haben.

3.3.1 Visuelle Darstellung des Systems

Die Interaktion des Benutzers mit dem System soll derart gestaltet werden, daß er sich leicht ein Systemmodell bilden kann. Dieses Systemmodell umfaßt u.a. die Struktur des Systems sowie die Möglichkeiten der Interaktion und ihre Auswirkungen auf das System.

Um den Vorgang der Bildung eines Systemmodells zu unterstützen, ist eine visuelle Darstellung des Systems erforderlich. Diese Darstellung muß nicht notwendigerweise in einer rechnergenerierten graphischen Form vorliegen, sie kann beispielsweise auch als Zeichnung zur Verfügung stehen.

Diese visuelle Darstellung soll aber nicht nur eine statische Systemstruktur zeigen. Sie hat als Rahmen für die Interaktion mit dem Benutzer zu dienen. Dazu gehört das Aufzeigen des Handlungskontextes sowie eine Andeutung der möglichen Auswirkungen einer Interaktion auf das System. Deshalb ist eine rechnergenerierte graphische Darstellung unter Verwendung eines hochauflösenden Graphik-Bildschirms vorzuziehen.

3.3.2 Komponentendarstellung

Die Zusammensetzung eines Bildfolgenauswertesystems aus einer Menge von Interaktionskomponenten muß sich in der graphischen Darstellung wi-

derspiegeln. Dieses wird erreicht, wenn die einzelnen Komponenten als unterscheidbare Einheiten in der Darstellung auftreten. Diese Visualisierung der Komponenten umfaßt gleichzeitig eine Darstellung ihres Zustandes.

Die Zahl der Komponenten ist vermutlich umfangreich. Der zur Darstellung der Komponenten verfügbare Platz auf dem Bildschirm ist jedoch beschränkt und wird eine vollständige Darstellung aller Komponenten selten ermöglichen. Es liegt nahe, eine abgestufte Darstellung zu fordern. Dabei kann vom Benutzer entschieden werden, mit welcher Genauigkeit der Zustand einer Komponente angezeigt wird.

Weiterhin ist die einer hierarchischen Strukturierung angemessene Systemdarstellung zu unterstützen. Darunter soll eine Zusammenfassung von Komponenten zu *virtuellen Komponenten* verstanden werden. Durch diese Abstraktion kann die Komplexität der Systemdarstellung reduziert und der Umfang der angezeigten Information über das System gesteuert werden. Dem Benutzer wird es weiterhin durch eine geeignete visuelle Strukturierung der Komponenten möglich, die Schwerpunkte seiner aktuellen Tätigkeit deutlich zu machen. Er kann diejenigen virtuellen Komponenten in der Darstellung verfeinern, auf die er seine Aufmerksamkeit konzentrieren will, während er andere durch Vergröberung zusammenfassen kann.

Diese Form der interaktiven Strukturierung erlaubt die für eine spezielle Aufgabe eines Benutzers angemessene Darstellung des Bildfolgenauswertesystems.

3.3.3 Objektorientierte Interaktion

Bei der Auswertung und Deutung von Bildfolgen arbeitet der Benutzer mit einer Reihe von abstrakten Modellen. Diese Modellierung ist Teil des Prozesses der Problemlösung durch den Benutzer. Die Auswirkung zeigt sich in der Strukturierung des Auswertesystems und in den dabei verwendeten Datenstrukturen. Als Beispiele seien Datenstrukturen wie Bilder (binär, grauwert, intrinsisch), Masken jeder Art, markante Punkte, Punkteketten, Verschiebungsmessungen, 3D-Modelle usw. genannt, die im Rahmen des MORIO-Systems [*Dreschler & Nagel 82*] verwendet werden.

Während der Interaktion des Benutzers mit dem System soll deutlich werden, mit welcher Komponente er kommuniziert oder welche Datenstrukturen er manipuliert. Diese Form der Interaktion soll sich an dem Objektbegriff orientieren. *Objekte* können sowohl Interaktionskomponenten als auch zu manipulierende Datenstrukturen sein. Die Interaktion des Benutzers

mit dem System findet immer auf der Basis der im System vorhandenen Objekte statt.

Diese *objektorientierte Interaktion* stellt während des Kommunikationsvorganges die beteiligten Objekte in den Vordergrund. Diese Objekte sollten auch in der graphischen Darstellung hervorgehoben werden. Die Art der Kommunikation und die Struktur der verwendeten Daten werden ausschließlich durch die betroffenen Objekte bestimmt. Andere nicht beteiligte Objekte können diesen Vorgang nicht beeinflussen und werden dadurch auch nicht direkt betroffen. Es existieren weder eine globale Kontrolle noch ein Dialograhmen, die eine zentrale Steuerung der Interaktion bewirken.

Die Bindung der Interaktionsmöglichkeiten an Objekte erlaubt eine den Problemen adäquate Dialogführung. Sie verhindert unnötige Abhängigkeiten der Objekte untereinander und gestattet eine flexible Anpassung des Systems durch eine Änderung seiner Objekte.

Diese objektorientierte Interaktion sollte gleichzeitig visuell unterstützt werden, indem der Benutzer eine graphische Darstellung dieser Objekte erhält. Es empfiehlt sich die Verwendung von Fenstertechniken, um die Dialogführung mit Objekten zu erleichtern und optisch hervorzuheben.

3.3.4 Objektgebundener Handlungskontext

Der Handlungskontext der Interaktion wird an die betroffenen Objekte gebunden und nicht an den globalen Ablauf des Systems. Der Benutzer kann den Dialog mit einem Objekt jederzeit unterbrechen, um einen Dialog mit einem anderen Objekt aufzunehmen oder fortzuführen. Der Kontext bleibt bei einer Unterbrechung solange erhalten, bis der einmal begonnene Interaktionsvorgang durch den Benutzer abgeschlossen wird.

Dem Benutzer sollte aber auch die Möglichkeit gegeben werden, einen einmal begonnenen Interaktionsvorgang endgültig abzubrechen oder einen Teil seiner Aktionen rückgängig zu machen. Dabei ist jedoch durch die Benutzerschnittstelle sicherzustellen, daß auch bei einem Abbruch der Interaktion durch den Benutzer das davon betroffene Objekt in einem konsistenten Zustand verbleibt.

Der Benutzer kann seine Dialogführung nach der Aufgabenstellung und den anfallenden Ereignissen richten. Er ist nicht an eine vom System vorgegebene Reihenfolge gebunden.

3.3.5 Direkte Manipulation

Das Konzept der objektorientierten Interaktion sollte gleichzeitig durch die Fähigkeit zur direkten Manipulation von Objekten [*Shneiderman 83*] unterstützt werden.

Die Möglichkeit der Steuerung eines Systems durch den Benutzer ist wohl in allen interaktiven Systemen gegeben. Dabei liegt die Initiative für die Systemsteuerung und die Dialoggestaltung beim System und nicht beim Benutzer. Dies gilt besonders für alle bekannten Bildverarbeitungssysteme (siehe Kapitel 4).

Der Benutzer kann häufig nur in einer von dem System vorgegebenen Reihenfolge mit diesem kommunizieren und so in den Verarbeitungsablauf eingreifen. Die meisten Bildverarbeitungssysteme bieten eine Schnittstelle, die an einer Kommandosprache orientiert ist. Diese erlaubt dem Benutzer durchaus die Initiative beim Eingreifen, die Kommandoschnittstelle ist aber meistens nur in einem speziellen Modus des Systems verfügbar.

Eine direktmanipulative Schnittstelle sollte dem Benutzer eine andere Art der Einflußnahme anbieten. Die Interaktion findet nicht mittels vorgegebener Frage-Antwort Schemata oder durch eine an einer komplexen Syntax orientierten Kommandoschnittstelle statt, sondern bietet dem Benutzer ein Eingreifen durch physische Aktionen. Diese äußern sich in der Auswahl und Bewegung von graphisch dargestellten Objekten oder durch Drücken speziell markierter Tasten. Als Zeigeinstrumente können eine Maus, Rollkugel, Lichtgriffel o.ä. verwendet werden.

Die Manipulation durch den Benutzer ist als direkt anzusehen, da er jederzeit die Initiative ergreifen kann, um das System durch eine Veränderung der dargestellten Objekte zu beeinflussen. Die Auswirkungen seiner Aktionen werden ihm sofort angezeigt.

Die Benutzerinteraktion findet verstärkt nach der Devise statt:

"Mehr agieren, weniger reagieren."

3.3.6 Modusvermeidende Interaktion

Bei den meisten interaktiven Systemen ist die Art und Form der Interaktion an bestimmte Modi gebunden. Ein solcher Modus (mode) kann folgendermaßen definiert werden [*Smith et al. 82*]:

> Ein *Modus* eines interaktiven Systems äußert sich gegenüber dem Benutzer als ein spezieller Zustand der Benutzerschnitt-

> stelle, der für einen bestimmten Zeitraum anhält und den der Benutzer mit keinem graphisch dargestellten Objekt der Benutzerschnittstelle verbinden kann.

Ein Modus wird im Rahmen eines interaktiven Systems dazu verwendet, sämtliche Benutzereingaben in einer einheitlichen Form zu interpretieren. Konventionelle interaktive Systeme können immer nur einen Modus zur Zeit annehmen. Bei mehreren verfügbaren Modi ist die Definition von mindestens einem speziellen Eingabesymbol (*Fluchtsymbol*) vorteilhaft, damit der Benutzer mithilfe des Fluchtsymbols einen sofortigen Moduswechsel durchführen kann.

Fast alle interaktiven Systeme verwenden Modi zur unterschiedlichen Interpretation der Benutzereingabe. Gut bekannt sind derartige Modi von Texteditoren. Bei zeilenorientierten Editoren wird oft zwischen einem "äußeren" und "inneren" Zeilenmodus, bei bildschirmorientierten Editoren zwischen einem Zeilen- und Bildschirmmodus unterschieden.

Eine der häufigsten Fehlerursachen beim Umgang des Benutzers mit interaktiven Systemen liegt an dessen Unkenntnis über den aktuellen Modus der Systeme. Der Benutzer befindet sich oft in Situationen, in denen ihm der momentane Modus der interaktiven Schnittstelle unbekannt ist oder von ihm falsch interpretiert wird. Die Folgen eines solchen Irrtums äußern sich in einer von dem Benutzer nicht erwarteten Reaktion des Systems auf seine Eingaben.

Bei Bildverarbeitungssystemen ist im Gegensatz zu Texteditoren das Erreichen von bestimmten Modi kaum oder gar nicht vom Benutzer steuerbar. Ein Grund dafür ist in der zur Zeit noch wesentlich längeren Bearbeitungsdauer eines Bildes gegenüber einem Zeichen zu sehen. Weiterhin bieten die meisten Bildverarbeitungssysteme keine Fluchtsymbole an, die dem Benutzer das (asynchrone) Auslösen eines Moduswechsels erlauben würden. Die Interaktion mit einem Bildverarbeitungssystem kann meistens nur dann erfolgen, wenn sich dessen Kommandoschnittstelle als eingabebereit meldet.

Aus diesen Gründen soll eine Benutzerschnittstelle gefordert werden, die auf dem Prinzip der modusvermeidenden (modeless) Interaktion basiert. Kann die Verwendung von Modi nicht vermieden werden, so soll die Schnittstelle die Dauer eines derartigen Modus dem Benutzer graphisch signalisieren und die in diesem Modus eingeschränkten Operationen klar anbieten.

3.3.7 Reaktionsbereitschaft

Zur Unterstützung der modusvermeidenden Interaktion muß die Benutzerschnittstelle ständig bereit zur Kommunikation mit dem Benutzer sein. Die Interaktion mit dem System ist dann jederzeit möglich und nicht mehr an bestimmte Modi gebunden.

Auf Benutzereingaben sollte die Schnittstelle immer sofort reagieren. Diese Reaktion dient als Bestätigung der Eingaben des Benutzers. Die Rückmeldung sollte in Form einer graphischen Darstellung erfolgen, die von dem Benutzer eindeutig interpretiert werden kann.

Als Grundgebot für die Interaktion hat zu gelten:

> "Niemals eine Benutzereingabe ohne eine
> (graphische) Rückmeldung des Systems."

3.3.8 Konsistenz

Erfahrungsgemäß entstehen komplexe Programmsysteme durch die Mitarbeit zahlreicher Autoren. Dieses gilt auch für Systeme zur Auswertung von Bildfolgen. Dadurch entstehen oft Systeme, deren Benutzerschnittstellen nicht konsistent sind.

Für den raschen Aufbau eines zutreffenden Systemmodells im Verständnis des Benutzers ist aber die Konsistenz der interaktiven Schnittstelle ein wichtiges Kriterium. Diese Konsistenz garantiert dem Benutzer, daß er ihm bekannte Interaktionsmechanismen unabhängig von der aktuellen Situation immer gleichartig anwenden kann. Die Forderung der Konsistenz gehört zu denen, die am schwersten zu realisieren sind.

Als ein Beispiel für ein System, welches eine konsistente Schnittstelle für die Benutzer propagiert, diese Konsistenz aber dann doch nicht einhält, kann das MacWrite-Programm auf dem MacIntosh-System von Apple dienen. Das MacWrite-Programm erlaubt dem Benutzer das bildschirmorientierte Erstellen von Textdokumenten. Um ein erstelltes Dokument drukken zu können, muß ein Bearbeitungsmenü aktiviert werden. Aus diesem kann die Option "Drucken" selektiert werden. Daraufhin wird ein Fenster eröffnet, in dem der Benutzer seinen Wunsch und den Namen des zu druckenden Dokumentes bestätigt.

Diese Art, ein Dokument auszudrucken, widerspricht dem Gebot der Konsistenz, da der Benutzer nicht wie gewohnt das Dokument direkt handhaben kann. Eine systemkonsistente Form des Druckens wird in dem Xerox

Star System angeboten. Um ein Dokument auszudrucken, selektiert der Benutzer dieses Dokument und überträgt oder kopiert es in ein vom System definiertes Drucker-Piktogramm. Bevor das System ein in diesem Piktogramm abgelegtes Dokument ausdruckt, fragt es nach Parametern (z.B. Auswahl der zu druckenden Seiten; Anzahl der Ausdrucke usw.).

Die Konsistenz hat aber nicht nur für die Form der Operationen, sondern auch für die Ein- bzw. Ausgabe während der Interaktion zu gelten. Für den Bereich der Bildfolgenauswertesysteme ist eine allgemeine Schnittstelle zu fordern, die Standardoperationen und -strukturen zur konsistenten Interaktionsführung anbietet.

3.3.9 Einfachheit

Ein System ist benutzerfreundlicher, d.h. leichter handhabbar und kontrollierbar, wenn seine Schnittstelle einfach benutzbar ist. Bei einer einfachen Schnittstelle hat es der Benutzer leichter, sich ein Systemmodell zu bilden. Diese Eigenschaft der einfachen Handhabung darf aber nicht auf Kosten der Funktionalität der Benutzerschnittstelle erreicht werden.

Dabei hat die folgende Devise zu gelten:

> "Einfache Aufgaben sollten leicht durchführbar sein,
> komplexe Aufgaben müssen möglich sein."

Eine konsistente Schnittstelle ist eine Grundvoraussetzung für eine einfache Schnittstelle. Ein Mittel zur Gestaltung einer einfachen Schnittstelle ist die Definition einer minimalen, aber ausreichenden Menge von Operationen.

Die Redundanz von Operationen führt oft nur zu einer unnötigen Komplexität und trägt zur Verwirrung der Benutzer bei. Die Redundanz ist allerdings zulässig, wenn durch sie eine signifikante Erleichterung des Benutzers im Umgang mit dem System erreicht wird. Als Beispiel sei die Möglichkeit genannt, vom Benutzer häufig verwendete Operationsfolgen als Kurzform (d.h. ein Tastendruck) anzubieten.

Zur Vereinfachung der Schnittstelle trägt auch eine Abstraktion der Funktionalität von Operationen bei. Diese Abstraktion kann durch die Definition von *universellen Operationen* erfolgen, die jeweils eine Aufgabenklasse realisieren und universell auf Objekte anwendbar sein sollen. Diese Operationen haben eine allgemein gültige Semantik, die bei der Anwendung

der Operatoren auf bestimmte Objekte erweitert wird. Ihre exakte Semantik wird durch die ausgewählten Objekte bestimmt. Sie werden deshalb auch als *generische Operationen* bezeichnet.

3.4 Zusammenfassung

Die in diesem Kapitel vorgestellte These besagt, daß der Entwurf einer ergonomischen Benutzerschnittstelle für Bildfolgenauswertesysteme mehr Erfolg verspricht, wenn dieser Entwurf durch die Entwicklung eines *Benutzermodells* unterstützt wird. Dieses vom Konstrukteur der Schnittstelle entwickelte Benutzermodell sollte dann als Basis für den Entwurf der Benutzerschnittstelle dienen.

Die Modellierung des Benutzers durch den Konstrukteur der Schnittstelle und damit die Herleitung seines Benutzermodells erfolgt durch die Herleitung eines *Systemmodells* für Bildfolgenauswertesysteme. Dabei entwickelt der Konstrukteur des Systems stellvertretend für die zukünftigen Benutzer deren Systemmodell.

Auf der Grundlage dieses Systemmodells werden *Konzepte zur Gestaltung der Schnittstelle* entwickelt. Diese Kriterien zur Gestaltung der Benutzerschnittstelle dienen als Grundlage für den Systementwurf und prägen das im System verankerte *Benutzerbild.* Das Ziel dieses Entwurfs liegt darin, das Benutzerbild der Schnittstelle mit dem von den Benutzern gebildeten Systemmodell in eine möglichst große Übereinstimmung zu bringen.

Als Systemmodell des Benutzers wird die Zerlegung eines Bildfolgenauswertesystems in eine Menge von *Interaktionskomponenten* zugrunde gelegt. Die Interaktionskomponenten teilen sich auf in *periphere Geräte*, *Datenverwaltung* und *Programmkomponenten.* Das Systemmodell beschreibt weiterhin die Ziele des Benutzers während der Interaktion. Dabei lassen sich die die Ermittlung des aktuellen *Systemzustandes*, die *Hilfestellung* und die *Manipulation des Systems* unterscheiden.

Es wurden Konzepte entwickelt, die als Grundlage für die Gestaltung einer Benutzerschnittstelle dienen sollen. Die *graphische Darstellung* des Bildfolgenauswertesystems und seiner Komponenten soll beim Benutzer die Bildung eines Systemmodells unterstützen. Zur Interaktion mit dem System und zur graphischen Darstellung werden fortgeschrittene Ein- und Ausgabegeräte wie eine *Maus* und ein *Farbrastergraphik-Gerät* gefordert. Die Kommunikation des Benutzers mit dem System soll auf der Basis der

dargestellten Zerlegung des Systems stattfinden. Durch die *objektorientierte Interaktion* und den *objektgebundenen Handlungskontext* wird dieser Vorgang für den Benutzer erleichtert. Das Konzept der *direkten Manipulation* von graphisch dargestellten Objekten wird durch eine weitgehend *modusvermeidende Interaktion* und eine ständige *Reaktionsbereitschaft* der Benutzerschnittstelle unterstützt. Die Forderung nach einer *konsistenten* und *einfachen* Schnittstelle soll deren Handhabung erleichtern.

Das in diesem Kapitel entwickelte *Benutzermodell* wird im folgenden Kapitel durch die Untersuchung der Interaktion anhand von ausgewählten Systemen zur Verarbeitung und Auswertung von Bildern und Bildfolgen untermauert und verfeinert. Die dort beschriebenen Systeme lassen einen Rückschluß auf bei Benutzern schon vorhandene Modelle von Bildauswertesystemen zu und gestatten aus den dabei gewonnenen Erkenntnissen eine detaillierte Formulierung des Benutzermodells.

Kapitel 4

Interaktion in Bildverarbeitungssystemen

In diesem Abschnitt sollen einige Bildverarbeitungssysteme dargestellt werden. Der Schwerpunkt dieser Darstellung liegt auf der interaktiven Schnittstelle dieser Systeme. Das Spektrum der betrachteten Systeme umfaßt Unterprogrammbibliotheken mit einer einfachen Kommandoschnittstelle als interaktiven Zugang (VICAR [*Castleman 79*]), komfortable Systeme, die eine umfangreichere Kommandostruktur anbieten (DIBIAS [*Triendl et al. 82*]), Systeme mit einem integrierten Texteditor (CELLO [*Bengtsson et al. 81*]), hochsprachliche Systeme mit speziellen Geräten und eigener Verarbeitungssprache (PICAP [*Gudmundsson 82*]), Systeme mit einer Kommandosprache, die auf Unix basiert (HIPS [*Landy et al. 84*]), und Systeme mit einem Mehrfachfenster-Konzept (BROWSE [*McKeown & Denlinger 82*]).

Die am Beispiel dieser Systeme gewonnenen Erkenntnisse über Interaktionsstrukturen in der Bildverarbeitung werden dargestellt. Die grundlegenden Konzepte werden extrahiert und auf ihre Verwendbarkeit für die Auswertung von Bildfolgen überprüft. Abschließend erfolgt eine Bewertung der interaktiven Schnittstellen der in diesem Abschnitt dargestellten Systeme.

4.1 Beispielsysteme

Die Auswahl der hier vorgestellten Systeme erhebt keinen Anspruch der Vollständigkeit. Es wurden aus den vielen bekannten Systemen diejenigen ausgewählt, deren interaktive Schnittstelle als repräsentativ für einen bestimmten der nachfolgend erläuterten Klassifikationsaspekte angesehen werden kann oder unter den in dieser Arbeit behandelten Gesichtspunkten

als bemerkenswert erscheint.

Vergleiche von existierenden Sprachen und Systemen für die Bildverarbeitung finden sich in *Maggiolo-Schettini 81*, *Preston 80*, *Preston 81*, *Kulpa 81a* und *Preston 83*.

Bildverarbeitungssysteme lassen sich im Hinblick auf ihre interaktive Schnittstelle nach fünf Aspekten klassifizieren.

(a) *Unterprogrammpakete ohne eine interaktive Schnittstelle.*
Sie sind überwiegend in FORTRAN implementiert und können aus einer Gastsprache (meistens ebenfalls FORTRAN) angesprochen werden. Als Beispiele seien das PAXII-System [*Johnston 70*] genannt, das von seiner Konzeption her mehr an einer Stapelverarbeitung orientiert ist und das SPIDER-System [*Tamura et al. 83*], welches bewußt ohne eine interaktive Schnittstelle entworfen wurde, um eine erhöhte Portabilität der Verarbeitungsroutinen zu erreichen.

(b) *Systeme mit einer Kommandosprache als interaktiver Schnittstelle.*
Diese Kommandosprache ist in der Regel von ihrer Struktur her recht einfach gehalten und dafür gedacht, einen interaktiven Zugang zu einer Unterprogrammbibliothek anzubieten. Diese Form von Systemen ist am meisten verbreitet und kann als eine Fortentwicklung der nach dem obigen Punkt klassifizierten Systeme angesehen werden. Als Beispielsysteme finden sich VICAR [*Castleman 79*], KANDIDATS [*Haralick & Minden 78*], DIBIAS [*Triendl et al. 82*], CELLO [*Bengtsson et al. 81*] und PIC [*Engelmann & Meinzer 84*].

(c) *Erweiterungen von bekannten Programmiersprachen.*
Diese Systeme zeichnen sich dadurch aus, daß eine bekannte Programmiersprache um spezielle Konstrukte (Datentypen, Operationen) zur Verwendung in der Bildverarbeitung erweitert wurden. PIXAL (Picture Algorithmic language) [*Levialdi et al. 81*] ist eine Erweiterung von ALGOL-60 und bietet u.a. Parallelität bei der Ausführung von Anweisungen. Die Sprache L [*Radhakrishnan et al. 81*] ist ebenfalls eine Erweiterung von ALGOL-60 und bietet u.a. spezielle Datentypen und Operationen für Bilder und Bildgruppen. Als Erweiterung von PASCAL sei noch PascalPL [*Uhr 81*] genannt. Diese Spracherweiterung fügt zu PASCAL die Fähigkeit zur parallelen Bearbeitung von Feldern hinzu. Die Sprache MAC [*Douglass 81*] wurde zum Entwurf von asynchronen, parallelen Bildverarbeitungsalgorithmen konzipiert.

Sie verwendet Konstrukte wie Prozesse, "guarded commands" und "guarded regions".

(d) *Spezielle Systeme, die auf einer eigenen Programmiersprache und oft auch auf speziellen Geräte-Konfigurationen basieren.*
Die für diese Systeme verwendete Programmiersprache wurde jeweils neu entwickelt, wobei die Autoren sich meistens an bekannten Sprachen orientiert haben. Diese Systeme grenzen sich gegenüber den unter Punkt (b) genannten Systemen dadurch ab, daß sie eine vollständige Programmiersprache anbieten. Als Beispiele seien IFL (Instruction Forming Language) [*Balston 81*], die eine starke Ähnlichkeit mit BASIC hat, und das PICAP-System [*Gudmundsson 82*] genannt, welches eine integrierte Programmierumgebung mit der Sprache PPL (Picture Processing Language) anbietet.

(e) *Vollständige Laborumgebungen, die spezielle Geräte- und Programm-Systeme zur Bildverarbeitung anbieten.*
Diese treten meistens bei Bildverarbeitungssystemen auf, die sich auf einen Aufgabenbereich konzentrieren und dabei eine vollständige Bearbeitung desselben versuchen. Ein Beispiel ist das VISION-System [*Hanson & Riseman 78*], welches zur Interpretation von statischen, monokularen Farbbildern von natürlichen Szenen entworfen wurde. Dort wird Lisp zur "höheren" Verarbeitung und FORTRAN für "einfache" Operationen verwendet. Ein anderes wissensbasiertes System [*Levine 78*] zur Interpretation von Realweltszenen verwendet modulare Unterprozesse zur Systemgestaltung. Das MORIO-System [*Dreschler 81, Dreschler & Nagel 82*] interpretiert monokulare TV-Bildfolgen von Realwelt-Vorlagen. Die Implementation erfolgte in PASCAL mit einer speziellen Erweiterung zur Ausführung auf einem homogenen Rechnernetz.

4.1.1 KANDIDATS

Das KANDIDATS-System [*Haralick & Minden 78*] ist ein gutes Beispiel für ein Bildverarbeitungssystem, das eine Unterprogrammbibliothek über eine Kommandosprache interaktiv zugänglich macht. Es besteht aus einem Überwachungsprogramm (Monitor) und einer Menge von Verarbeitungsprogrammen (über 100). Der Monitor dient als Schnittstelle zwischen dem Benutzer und dem Bildverarbeitungssystem. Es erfolgt jedoch auch

eine direkte Interaktion des Benutzers mit den Anwendungsprogrammen.

Im Standardmodus, der durch ein spezielles Ausgabesymbol dem Benutzer signalisiert wird, analysiert der Monitor die Benutzereingaben und zerlegt sie in ihre Bestandteile. Vom Monitor wird nur eine einfache Kontrolle der Eingabe durchgeführt. Im Fehlerfall (beispielsweise bei unbekannten Befehlen) erfolgt eine Interaktion mit dem Benutzer. Das dem akzeptierten Befehl zugeordnete Verarbeitungsprogramm wird aktiviert. Dabei werden die eingegebenen Parameter übergeben.

Das Verarbeitungsprogramm überprüft die erhaltenen Parameter auf ihre Integrität und Vollständigkeit. Bei Bedarf kann es über standardisierte Ein-Ausgabeprozeduren mit dem Benutzer kommunizieren. Rückmeldungen von den Verarbeitungsprogrammen erfolgen in Form eines Fehlerschlüssels. Der Benutzer kann ein aktiviertes Programm abbrechen, die Kontrolle kehrt dann wieder an den Monitor zurück.

Die Verarbeitungsprogramme werden als Funktionen auf Bildern angesehen. Sie erhalten Bilder und Parameter als Eingabe und erzeugen wieder Bilder. Die Verarbeitung von Bildern wird als Serie von diskreten Schritten angesehen. Jeder Verarbeitungsschritt erzeugt wieder ein Bild, das im weiteren Verlauf der Abarbeitung wieder verwendet werden kann.

Bilder werden als Dateien verstanden, die in einem flexiblen Format (Standard image format) von dem System erzeugt werden. Dieses Format umfaßt verschiedene Darstellungsmodi und ermöglicht unterschiedliche Zugriffsarten. Der Benutzer kann Bilder über symbolische Namen bezeichnen und im Rahmen des Systems handhaben.

Andere mit KANDIDATS vergleichbare Systeme, auf die ansonsten nicht mehr näher eingegangen wird, werden nachfolgend kurz beschrieben:

- PICASSO, PICASSO-SHOW und PAL [*Kulpa 81b*].
  PICASSO (Picture Assembly-programmed Operations) ist eine umfangreiche Bibliothek mit einfachen Operationen auf Bildern. Sie enthält ca. 170 Operationen und ist in Assembler implementiert. PICASSO-SHOW wurde als interaktive Schnittstelle entworfen, die auf der PICASSO-Bibliothek aufbaut. Sie soll leicht erweiterbar sein und zum schnellen Erstellen von kleineren Bildverarbeitungsprogrammen dienen. Als Datenstrukturen bietet sie Bilder, Vektoren und Listen. Die Kontrollstrukturen sind sehr einfach gehalten. Die Syntax erinnert stark an eine Assembler-Notation. Der zugehörige Interpretierer kennt drei Modi:

interaktiv: Direkte Ausführung jeder eingegebenen Zeile.

interpretativ: Sequentielle Ausführung einer Liste von Anweisungen als Programm.

schnelle Interpretation: Vor Ausführung wird ein Programm in eine Zwischensprache übersetzt.

Die Sprache PAL wurde an ALGOL-68 angelehnt und als spezielle Sprache für die Bildverarbeitung konzipiert. Sie enthält einige Konzepte, die auch in Ada wiederzufinden sind: Moduln zur Strukturierung, Überladen von arithmetischen Operatoren, Zugriff bis auf die Maschinenebene. PAL existiert jedoch nur als Sprachentwurf, sie wurde nie implementiert.

- KIPL [*Fukushima et al. 81*].
  Dieses auf einem Kleinrechner implementierte Bildverarbeitungssystem bietet eine Bibliothek von FORTRAN-Unterprogrammen. Die verfügbaren Kommandos sind als Makro-Prozeduren definiert und können deshalb vom Benutzer erweitert werden. Für alle Kommandos und die zugehörigen Unterprogramme existieren Parameterbeschreibungen, mit denen Werte voreingestellt werden können. Die zu übergebenden Werte können vom System auch interaktiv erfragt werden. Die Benutzerschnittstelle führt eine Kommandohistorie, die mit einem Editor verändert werden kann, wobei die daraus resultierenden Befehle erneut ausgeführt werden können. Für Bilder wird eine Historie aller Operationen gehalten, die über alle angewendeten Operationen buchführt.

- VICAR [*Castleman 79*].
  Das Video Communication and Retrieval System (VICAR) dient zur Unterstützung von allgemeiner Bildverarbeitung. Dessen Schnittstelle akzeptiert eine einfache Kommandosprache, die auch Prozeduren mit Parametern erlaubt. Es können Daten auf Magnetplatte oder -band verwaltet sowie in Assembler oder FORTRAN geschriebene Bildverarbeitungsprogramme aktiviert werden. Den Anwendungsprogrammen können Parameter übergeben und deren Ergebnisse auf einem Bilddarstellungssystem angezeigt werden.

- CELLO [*Bengtsson et al. 81*].
  Das CELLO-System wurde hauptsächlich für die Analyse von Mikroskop-Zellbildern entwickelt. Es besteht aus einem Texteditor, Über-

setzer bzw. Interpretierer für die Kommandosprache und einem Laufzeitsystem. Die interaktive Schnittstelle erlaubt die Definition von Kommandoprozeduren und deren Abarbeitung im Hintergrund (Stapelbetrieb). Alle verfügbaren Befehle aktivieren separat übersetzte FORTRAN-Unterprogramme. Von vorgegebenen Datentypen kann der Benutzer interaktiv Objekte erzeugen. Für diese wird automatisch eine spezifische Datenbasis zur Sicherung angelegt. Es können interaktiv höchstens zwei Grauwertbilder ($128 \times 128 \times 6$ Bit) sowie vier Masken ($128 \times 128 \times 1$ Bit) benutzt werden. Die meisten verfügbaren Operationen arbeiten auf Bildern und liefern als Ergebnis wieder Bilder oder Merkmalsmengen.

- DIBIAS [*Triendl et al. 82*].
  Das Digitale Interaktive Bildanalyse System (DIBIAS) ist ein Mehrbenutzer-Mehrprozeßsystem, das wahlweise interaktiv oder im Stapelbetrieb benutzt werden kann. Nach der syntaktischen Analyse der eingegebenen Befehle werden die entsprechenden Bildverarbeitungsprogramme als Unterprozesse aktiviert. Die Kommando-Interpretation bleibt während deren Ausführung weiter eingabebereit. Den Anwendungsprogrammen können als Parameter Bildnamen, skalare Werte und Vektoren übergeben werden. Der Benutzer kann weiterhin Kommandoprozeduren definieren und ausführen lassen.

  Die zur Verfügung stehenden Operationen arbeiten nach dem Prinzip, daß sie als Eingabe ein Bild erhalten und als Ausgabe wieder eines produzieren. Die Bildverarbeitungsprogramme werden als Moduln angesehen, die über Bildeingänge und -ausgänge sowie über Parameter verfügen.

4.1.2 PIC

Das Bildverarbeitungssystem PIC [*Engelmann & Meinzer 84*] ist ebenfalls ein Repräsentant aus der Gruppe der Systeme, die einen interaktiven Zugang zu einer Unterprogrammbibliothek (in FORTRAN) anbieten. Bei der Darstellung dieses Systems wird einmal PIC als eine Sprache für die Bildverarbeitung [*Meinzer & Engelmann 82, Meinzer & Engelmann 83*], dann wieder als "Interpreter" [*Engelmann & Meinzer 84*] beschrieben.

Die Frage, ob eine Kommandosprache für die interaktive Benutzung einer Unterprogrammbibliothek schon als "Sprache für die Bildverarbeitung"

angesehen werden kann, ist auch bei anderen Autoren umstritten. In *Preston 83* wird die Meinung vertreten, daß eine Unterprogrammbibliothek, die durch ein Kommando-Interpretierer benutzt werden kann, im Rahmen einer interaktiven Umgebung als "Sprache" bezeichnet werden kann. *Maggiolo-Schettini 81* vertritt dagegen die Ansicht, daß derartige Systeme durch ihre Kommandosprache keine Bildverarbeitungssprache definieren. Dort wird es als sinnvoll bezeichnet, den Kern einer existierenden Programmiersprache für die Zwecke der Bildverarbeitung zu erweitern.

In dieser Arbeit wird ebenfalls die Meinung vertreten, daß eine Kommandosprache, die einen interaktiven Zugang zu einer Unterprogrammbibliothek realisiert, noch keine Bildverarbeitungssprache darstellt. Eine Kommandosprache kann dann als Bildverarbeitungs- oder -auswertungssprache angesehen werden, wenn sie ein gewisses Mindestmaß an Daten- und Kontrollstrukturen anbietet.

Nach dieser Einschätzung kann PIC zumindestens teilweise als Bildverarbeitungssprache klassifiziert werden. PIC besteht aus einem Kommando-Interpretierer und einer Bibliothek von etwa 100 Bildverarbeitungsoperationen. Die Kommandos können interaktiv oder im Stapelbetrieb bearbeitet werden.

Ein Bestandteil des Interpretierers ist ein "Eingabeprozessor", der die Befehle vor ihrer Ausführung auf ihre syntaktische und semantische Korrektheit überprüft. Die von PIC angebotenen Kommandos sind von ihrer Struktur her recht einfach und würden die Klassifizierung als Bildverarbeitungs- bzw. -auswertungssprache nicht rechtfertigen. Diese Klassifizierung erfolgt aufgrund einer Berücksichtigung der Eigenschaften des "Eingabeprozessors", der die Formulierung von Kontrollstrukturen und Prozeduren erlaubt.

Diese Prozeduren können über einen in PIC verfügbaren Editor erstellt und auf Textdateien gesichert werden. In Prozeduren können auch andere Prozeduren aufgerufen werden. Zur Ablaufkontrolle existieren Kontrollstrukturen wie IF, GOTO, WHILE usw.

Die eigentlichen PIC-Befehle bieten auch einen Menü-Betrieb an, der einem Benutzer alle möglichen Alternativen anzeigt. Die interaktive Schnittstelle kennt nur zwei Modi, entweder ist sie eingabebereit zur Kommando-Interpretation oder es arbeitet eine Operation, und das System ist nicht eingabebereit.

Die angebotenen Operationen arbeiten fast ausschließlich auf Bildern, die im Arbeitsspeicher unter fest vorgegebenen Namen verfügbar sein müs-

sen. PIC hat im Arbeitsspeicher drei Grauwertmatrizen ($900 \times 900 \times 16$ Bit) und vier Matrizen mit Gleitkommadarstellung ($256 \times 256 \times 32$ Bit) verfügbar. Sie werden von den PIC-Operationen als globale Operanden (Bildregister) verwendet. Für den Transport von Bildern aus dem Hintergrundspeicher in den Arbeitsspeicher und zurück ist der Benutzer verantwortlich. PIC bietet dafür entsprechende Operationen an. Als Ausgabegeräte stehen alphanumerische Terminale, graphische Terminale (Tektronix), Drucker und Zeichengeräte zur Verfügung.

Das Konzept von PIC ist weiterentwickelt und in einem Neuansatz mit einer erheblichen Erweiterung der zur Verfügung stehenden Operationen in die Sprache APL eingebettet worden. Das daraus entstandene PICAPL [*Dengler & Meinzer 85*] ist ein interaktives Bild- und Bildserien-Analysesystem.

Der Benutzer von PICAPL kann nicht nur auf eine große Sammlung von Bildverarbeitungsoperationen zurückgreifen, die in einer Bibliothek zur Verfügung stehen, sondern auch den vollen Sprachumfang von APL und den Komfort des Interpretierers nutzen. Durch eine in APL vorhandene FORTRAN-Schnittstelle steht weiterhin die gesamte SPIDER-Bibliothek zur Verfügung.

4.1.3 SPIDER

Das SPIDER-System (Subroutine Package for Image Data Enhancement and Recognition) [*Tamura et al. 83*] ist ein Beispiel für eine Unterprogrammbibliothek ohne interaktiven Zugang. Ein anderes System ist PAXII [*Johnston 70*].

SPIDER besteht aus über 400 FORTRAN-IV Unterprogrammen, die unterschiedliche Bildverarbeitungsalgorithmen realisieren. Weiterhin existieren noch Operationen zur Verwaltung der Unterprogramme. SPIDER wurde mit dem Ziel entworfen, eine erhöhte Portabilität von Bildverarbeitungsalgorithmen zu erreichen und damit den Austausch derartiger Algorithmen zwischen den Forschungsgruppen zu intensivieren.

Die Autoren von SPIDER haben deshalb die Programmstruktur von Bildverarbeitungssystemen analysiert. Dabei haben sie sechs Faktoren lokalisiert, die nach ihrer Erfahrung die Portabilität von derartigen Systemen erschweren.

1. Programmiersprache.
   Neben den Treibern für die Ein-Ausgabegeräte ist ein beträchtlicher Anteil in Assembler realisiert.

2. Zugriff auf Bilddateien.
   Sobald die Bildverarbeitungsprogramme sowohl Operationen auf Bildern als auch Dateizugriffe enthalten, können sie nur dann portiert werden, wenn die Zugriffsprotokolle und Dateistrukturen vereinheitlicht werden. Dieses ist aber schwierig zu standardisieren und deshalb extrem von dem Entwurfskonzept des jeweiligen Systems abhängig.

3. Bilddatenstrukturen in Programmen.
   Es existieren vielfältige Datentypen und -formate für die Darstellung von Bildern im Hauptspeicher. Spezielle Formate erhöhen nicht die Portabilität.

4. Transfer von Bilddaten zwischen Programmen.
   Es bestehen viele Methoden, um Bilddaten zwischen verschiedenen Programmen zu vermitteln (z.B. durch Parameterübergabe, COMMON-Blöcke usw.).

5. Fehlerbehandlung.
   Die Behandlung von Laufzeitfehlern ist sehr systemabhängig.

6. Bilddimensionen und Grauwertstufen.
   Bei dem Entwurf von Bildverarbeitungssystemen sind viele Beschränkungen bzgl. der Bildgröße und -darstellung derart fest im System verankert, daß eine Änderung kaum möglich ist.

SPIDER wurde deshalb als Kern eines Bildverarbeitungssystems konzipiert, der nur aus einer Unterprogrammbibliothek besteht. Es wurde versucht, bei dem Entwurf der Bibliothek den Einfluß der oben erwähnten störenden Faktoren entweder völlig auszuschliessen oder auf ein Mindestmaß zu reduzieren.

Als Implementationssprache wurde FORTRAN IV gewählt. Es sind jedoch noch weitere 19 Regeln zur Spracheinschränkung spezifiziert, um einen Standard zu erreichen, der von möglichst vielen Übersetzern für ANSI FORTRAN-66 akzeptiert wird.

Ein interaktiver Zugang ist nicht Bestandteil des Systems. Alle Unterprogramme sind frei von Datei-Operationen oder Zugriffen auf Bildverarbeitungsperipherie.

Die meisten Operationen arbeiten auf Bildern als Grauwertmatrizen (2D). Es können auch Bildfenster, Bereiche, Histogramme (als 1 dim. Felder), Schwellwerte und andere skalare Daten verwendet werden. Alle Parameter der Unterprogramme sind als im Hauptspeicher residente Unterprogrammparameter zu übergeben.

Fehlerkontrollen werden nur rudimentär durchgeführt. Erkannte Fehler werden über einen vorgeschriebenen Fehlerparameter an die Umgebung gemeldet.

4.1.4 HIPS

Das Human Information Processing Laboratory's Image Processing System (HIPS) [*Landy et al. 84*] ist in einer Unix-Umgebung verfügbar. Es besteht aus einer kleinen Anzahl von Programmen zur Behandlung von Standard-Dateiformaten für Bildfolgen und aus einer wachsenden Bibliothek von Programmen für Bildtransformationen. Es ist ebenfalls in die Kategorie der Unterprogrammbibliotheken mit interaktivem Zugang einzuordnen. Bemerkenswert ist jedoch die Verwendung von Unix als Programmierumgebung [*Kernighan & Mashey 81*] sowie der Unix-Kommandosprache und der Prozeßkonzepte (filter, pipes) für die Bildverarbeitung. Die Vorteile bei der Anwendung derartiger Konzepte für die Bildverarbeitung wurden schon in *Stevens & Hunt 82* erläutert.

Alle HIPS-Programme sind in C implementiert und als Filter (unix filter) aufgebaut (siehe Kapitel 2.1). Die Struktur von Filtern und Leitungen wird in dem HIPS-System extensiv verwendet. Mithilfe der Kommandosprache von Unix können die HIPS-Bildverarbeitungsprogramme über Leitungen verbunden werden, um komplexe Verarbeitungsschritte zu realisieren. Die Möglichkeiten der Verknüpfung werden dabei nur durch die Fähigkeiten der Kommandosprache beschränkt.

Um einen Filter auf ein Bild oder eine Folge von Bildern anzuwenden, müssen Parameter übergeben werden. Diese lassen sich in zwei Gruppen aufteilen, erstens diejenigen, die eine Bildfolge beschreiben, und zweitens die Parameter für die durchzuführenden Operationen. Um den Programmieraufwand zu verringern, werden die Bildparameter in einem standardisierten Bildkopf zusammengefaßt, der als wichtiger Bestandteil einer jeden gespeicherten Bildfolge gilt. Weiterhin gehört zu jeder Bildfolge eine Text-Dokumentation, die ebenfalls ein Bestandteil des Bildkopfes ist. Dadurch bietet dieses Konzept zwei Möglichkeiten: Das automatische Weitergeben

von Bildparametern in einer Transformationskette und die Fähigkeit, eine Historie der angewendeten Operationen zu dokumentieren.

Die Bildparameter im Kopf umfassen die Anzahl der Bilder, Zeilen, Spalten, Bits pro Pixel, Information über das Packen von Bits und das Pixel-Format (z.B. byte, integer, floating point).

Der deskriptive Teil des Bildkopfes enthält den Namen des Urhebers, den Namen der Bildfolge, das Datum, ein beliebig langer Text zur Beschreibung der Folge und ihre Historie.

Zum Lesen, Schreiben und Aktualisieren der Bildköpfe existiert eine kleine Programmbibliothek. Alle Filter greifen nur über diese Bibliothek auf die Bildköpfe zu. Die Bilder bzw. Bildfolgen werden mit Hilfe des Dateisystems verwaltet.

Zur Darstellung von Bildern gibt es Filter, die als Gerätetreiber arbeiten. An Bild-Peripherie sind ein Grinnell-System, Film- und Videosystem und ein Drucker verfügbar.

Das Konzept, welches diesem Ansatz von HIPS zugrunde liegt, geht davon aus, daß in einer Forschungsumgebung Überlegungen zur Effizienz von Ausführungen und zur Datenintegrität zweitrangig sind gegenüber der Programm-Modularität, der einfachen und schnellen Dokumentation von Bilddateien und der leichten Programmerzeugung und Wartung.

4.1.5 BROWSE

BROWSE [*McKeown & Denlinger 82*] ist ein System zur graphischen Darstellung und interaktiven Manipulation von Rasterbildern. Es ist ein Teil des Map Assisted Photo-interpretation Systems (MAPS). BROWSE ist in dieselbe Kategorie wie HIPS einzuordnen. Es soll hier wegen der Verwendung eines Mehrfachfenster-Konzeptes in der interaktiven Schnittstelle vorgestellt werden.

BROWSE erlaubt die Darstellung von Rasterbildern in mehreren Fenstern, das Überlagern von graphischen Daten wie Kartenbeschreibungen und Bildsegmentationen sowie die Spezifikation und Erzeugung von schattierten 3D-Oberflächenmodellen. Zur Darstellung von Bildern wird eine Auflösungshierarchie angeboten. Weiterhin existieren Operationen wie Verkleinern und Vergrößern eines Fensters, Verkleinern und Vergrößern eines Fensterinhaltes, Erzeugen, Löschen, Kopieren und Verschieben von Fenstern.

BROWSE stellt auf dem Bildschirm immer nur einen Rahmen (frame) dar. Dieser umfaßt eine Menge von Bildfenstern. Bildfenster sind als rechteckige Ausschnitte aus Bildern definiert und sind immer der Bestandteil eines Rahmens. Die Rahmen und Fenster können vom Benutzer durch symbolische Namen referenziert werden.

Weiterhin gibt es Segmentations-Dateien, die Bildsegmentationen als eine Liste von Merkmalen enthalten. Die Umrisse von Segmentationen können wieder als Graphiken dem Ausgangsbild überlagert werden.

BROWSE besteht aus einer Benutzerschnittstelle und einer Programmbibliothek, die Operationen zur Graphik- und Bildmanipulation anbietet. Die Kommando-Interpretation akzeptiert ca. 40 Befehle, die zur Manipulation von Rahmen, Fenstern, Bildern und Segmentationen dienen. Die zu manipulierenden Objekte können wahlweise durch Zeigeinstrumente oder durch symbolische Namen bezeichnet werden. Es können Kommandoprozeduren in Form von Textdateien bearbeitet werden.

Die Fenster dienen gleichzeitig als Kommunikationsmedium zwischen BROWSE und den Bildverarbeitungsprogrammen. Sie werden zur Darstellung von Ergebnissen des Anwendungsprogrammes verwendet. Diese Programme werden von BROWSE als Unterprozesse unter Benutzung der Standard-Dienste von Unix aktiviert.

BROWSE kennt zwei Methoden der Kommunikation mit Bildverarbeitungsprogrammen in Form von Unterprozessen. Bei der ersten Methode wird die interne Datenstruktur zur Beschreibung eines Bildfensters verwendet, um dem Anwendungsprogramm den zu bearbeitenden Bildausschnitt mitzuteilen. Dafür wird eine temporäre Datei mit den zu übergebenden Parametern erzeugt und das Anwendungsprogramm aktiviert. Dessen Resultate werden ebenfalls in einer temporären Datei abgelegt. Nach Ende des Anwendungsprogramms kann der Benutzer ein Fenster zur Darstellung der Resultate auswählen.

Die zweite Methode erfordert Anwendungsprogramme, die sich nach den Konventionen von BROWSE richten ("BROWSE-ähnlich"). Sie können dann über gemeinsam zugreifbare Dateien (shared files) kommunizieren. Als Unterprozesse können sie auf dieselbe Information zugreifen wie das BROWSE-System. Sie können den Zustand des Sichtgerätes erfragen und Modifikationen an BROWSE zurückgeben, die sofort wirksam werden.

Dieser Mechanismus wurde auf die MAPS-Datenbasis angewandt, um aufgabenspezifische "BROWSE-ähnliche" Programme zu erzeugen. Jedes dieser Programme ist für bestimmte Aufgaben (beispielweise vom Benutzer

gesteuerte Segmentationen) spezialisiert und kann in BROWSE jederzeit aktiviert werden. Der Benutzer ist dadurch in der Lage, alle Verarbeitungsschritte im Rahmen der BROWSE-Umgebung durchzuführen.

4.1.6 MORIO

Das MORIO-System [*Dreschler 81, Dreschler & Nagel 82*] soll aus mehreren Gründen betrachtet werden. Es kann einmal als ein System angesehen werden, welches auf einer umfangreichen Laborumgebung basiert. Weiterhin stellt es ein repräsentatives Beispiel für Systeme zur Auswertung von Bildfolgen dar, das aus der Mitarbeit zahlreicher Autoren entstanden ist [*Dreschler-Fischer et al. 83*].

Der dritte Grund ist der Ausbau des MORIO-Systems zu einem Experimentalsystem, das dem Benutzer für den Aufgabenbereich der Bildfolgenauswertung von TV-Realwelt-Vorlagen umfangreiche Test- und Auswahlmöglichkeiten zur Erprobung unterschiedlicher Algorithmen anbietet. In diesem Zusammenhang werden die Möglichkeiten der interaktiven Benutzerschnittstelle dargestellt und analysiert.

Das MORIO-System ist kein universell anwendbares System zur Auswertung von beliebigen TV-Bildfolgen. Es dient vielmehr der Untersuchung einer speziellen Aufgabenstellung, die möglichst vollständig bearbeitet wird. Dabei handelt es sich um die Auswertung von monokularen TV-Bildfolgen aus Realwelt-Vorlagen. Am Beispiel mehrerer Straßenszenen werden für bewegte starre Körper eine 3D-Beschreibung und ihre Bewegung relativ zur Kamera ermittelt. Die Lösung dieser Aufgabe wurde von den Autoren des MORIO-Systems in mehrere Teilprobleme zerlegt.

Die Bildanalyse umfaßt das Feststellen von Bildbereichen, die Abbildungen von bewegten Objekten sein können, das Finden von markanten Punkten, d.h. auffällige, als körperfest angenommene Meßpunkte in den Objektabbildungen, und das Ermitteln von räumlichen Koordinaten aus den Bildkoordinaten der markanten Punkte eines starren Körpers. Die 3D-Interpretation liefert als Resultat die euklidischen Koordinaten für die Modellierungspunkte in einem körperfesten Koordinatensystem. Aus diesen Punkten wird ein erstes grobes Objektmodell berechnet. Dieses Modell ist die konvexe Hülle der Punkte, es wird auf einem Rastersichtgerät wahlweise als Linienzeichnung oder als schattierter Polyeder dargestellt. Weiterhin kann dieses Modell auch in die Bildfolge zurückprojiziert und als graphische Darstellung den Rohdaten überlagert werden [*Dreschler 81*].

Aufbauend auf den Erkenntnissen des MORIO-Systems finden weiterführende Untersuchungen statt. Zum einen ist ein Ansatz entwickelt worden, der eine genauere Objektmodellierung aufgrund von Beleuchtungsparametern als Rekonstruktion von gekrümmten Oberflächen untersucht [*Westphal 84*].

In einer anderen Arbeit wurde ein Verfahren entwickelt, das bessere Aussagen über die Korrespondenz markanter Punkte erlaubt [*Nagel & Enkelmann 84*]. Dieses erfolgt durch die Berechnung von Verschiebungsvektorfeldern, die eine Möglichkeit darstellen, Bildänderungen zu beschreiben, die durch die Relativbewegungen zwischen Objekten einer Szene und der Aufnahmeapparatur zustande kommen. Mit einem zur Ermittlung von Verschiebungsvektorfeldern entwickelten Mehrgitterverfahren konnte die Berechnung der Verschiebungsvektorfelder wesentlich verbessert werden [*Enkelmann 85, Nagel & Enkelmann 86*]. Teile dieses Ansatzes wurden bereits in das MORIO-System integriert.

Alle Programme, die im Rahmen dieser Arbeiten entstanden, wurden in PASCAL implementiert und werden auf einem speziell dafür ausgestatteten homogenen Rechnernetz zur Ausführung gebracht. Sie verwenden einen PASCAL-Dialekt, der für die Verwendung dieses Kleinrechnernetzwerkes und der verfügbaren Video-Peripherie erweitert wurde.

Die interaktive Schnittstelle des MORIO-Systems orientiert sich teilweise an der funktionellen Zerlegung des Systems. Bei dem Start des Systems muß der Benutzer in einer vom System vorgegebenen Reihenfolge einen Dialog führen, in dem globale Verarbeitungsparameter gesetzt werden. Alle Parameter besitzen eine Voreinstellung, so daß ein Benutzer bei einer Standard-Benutzung auf diesen Dialog verzichten könnte. Neben den überwiegend skalaren Parametern wählt der Benutzer für den Punktefinder und das Korrespondenzverfahren den gewünschten Algorithmus aus.

Nach dieser Initialisierungsphase findet die weitere Interaktion über ein Rastersichtgerät statt. Dort wird das Bild angezeigt, welches gerade bearbeitet wird. Zur Kommunikation steht dem Benutzer außer dem alphanumerischen Terminal eine Rollkugel mit zwei speziellen Knöpfen zur Verfügung. Mit diesen Knöpfen kann der Benutzer jederzeit dem System signalisieren, daß er in den Dialogmodus eintreten möchte.

Bei Erreichen des Dialogmodus wird dem Benutzer ein Menü angeboten. Dieses enthält Optionen wie Ende des Dialogs ("Continue"), Beenden der Bearbeitung ("Finish Run"), Netzwerkzustand ("System State"), Kommando-Interpretation ("Command Decoder"), Hinzunahme bzw. Ent-

fernen von Zustandsausgaben ("Add/Remove Displays") und Hinzunahme bzw. Entfernen von Haltepunkten ("Add/Remove Stops").

Beim Netzwerkzustand wird der Status der an der Verarbeitung beteiligten Rechner ausgegeben. Die Kommando-Interpretation aktiviert einen Interpretierer, der seinen Dialog mit dem Benutzer über das alphanumerische Terminal abwickelt. Dieser Interpretierer dient der Erstellung von Dokumentationsmaterial mit Hilfe von Daten, die nur während der Ausführung von MORIO verfügbar sind. Der Benutzer kann über Kommandos verschiedene Geräte der Video-Peripherie schalten und bestimmte Zwischenergebnisse in graphisch aufbereiteter Form dokumentieren.

Die Optionen zur Steuerung von Graphikausgaben und Haltepunkten ermöglichen eine Kontrolle des Verarbeitungsfortganges und eine Überwachung durch Anzeige und Inspektion von Zwischenergebnissen. Dabei werden zwei Konzepte zur Systemüberwachung deutlich.

Einmal findet sich in MORIO eine vorgegebene Anzahl von Zuständen, in denen dem Benutzer eine graphisch aufbereitete Darstellung von Zwischenergebnissen angeboten wird. Diese Zeitpunkte werden durch einen signifikanten Fortschritt in der Abarbeitung und die Verfügbarkeit von für den Benutzer interessanten Zwischenergebnissen definiert.

Weiterhin gibt es in MORIO neben diesen Ausgaben von Zwischenergebnissen noch die Möglichkeit, eine fest vorgegebene Anzahl von Haltepunkten zu aktivieren bzw. deaktivieren. Das Erreichen von möglichen Haltepunkten orientiert sich ebenfalls an dem Fortschritt der Verarbeitung. Während dieser Zustände wird dem Benutzer das oben beschriebene Menü angeboten.

Die Aktivierung bzw. Deaktivierung der Darstellung von Zwischenergebnissen und der Haltepunkte erfolgt durch Untermenüs des Hauptmenüs. Diese enthalten als Optionen alle aktivierten Darstellungs- bzw. Haltepunkte.

4.1.7 PIXAL

Die Sprache PIXAL [*Levialdi et al. 81*] ist ein Beispiel für eine Bildverarbeitungssprache, die als Erweiterung aus einer bekannten Programmiersprache, in diesem Fall ALGOL-60, entstanden ist.

Die Sprache PIXAL (Picture Algorithmic Language) ist eine Erweiterung von ALGOL-60. Es existiert eine Implementation für einen HP-Rechner und für eine SIMD-Maschine.

PIXAL definiert als zusätzliche Datentypen Binär- und Grauwert-Datentypen zur Darstellung von Binär- und Grauwertbildern. Zusammen mit der Möglichkeit, Bilder als Felder zu definieren, werden zwei weitere mehrdimensionale Strukturen, Masken (mask) und Rahmen (frame), eingeführt. Weiterhin können parallel auszuführende Anweisungen formuliert werden.

Masken definieren ein Muster, das mit der Umgebung des Bildelementes verglichen wird, auf das die Maske positioniert ist. Ein Rahmen erlaubt das Auffinden von einer Teilmatrix um das Feldelement herum, auf das der Rahmen positioniert ist. Dies erlaubt die Definition von Nachbarschaft, um parallele Operationen durchzuführen.

Es gibt viele Vorschläge zur Erweiterung von Programmiersprachen für die Bildverarbeitung, für vergleichende Darstellungen sei auf *Preston 81*, *Maggiolo-Schettini 81* und *Preston 83* verwiesen. Im folgenden sollen einige von PIXAL unterschiedliche Ansätze dargestellt werden:

- L [*Radhakrishnan et al. 81*].
  Die Sprache L ist eine Erweiterung von ALGOL-60 und für die Programmierung von Bildverarbeitungsanwendungen gedacht. Die wichtigsten Erweiterungen von L enthalten:

  - Sechs neue Datentypen — Bilder, Bildgruppen, Logische Bilder und Bildgruppen; Fenster in Bildern sowie virtuelle Bilder und Bildgruppen. Die Bilder sind als zwei-dimensionale Felder verfügbar. Bildgruppen können als Felder mit Bildern als Komponenten (array of image) angesehen werden. Die Fenster definieren rechteckige Ausschnitte auf Bildern. Das Konzept der virtuellen Bilder wurde eingeführt, um mit realen Bildern unterschiedliche Attribute zu assoziieren.
  - Eine Menge von zusätzlichen Deklarationen und Operationen, um Bilder zu definieren, zu speichern und zu verarbeiten. Es gibt spezielle Kontrollstrukturen, um alle durch ein Bildfenster bestimmten Bildelemente zu bearbeiten.
  - Eine Menge von unären und binären Operatoren zur Verknüpfung von Bildern. Es können Prädikate für die neuen Datentypen definiert werden.
  - Es wird das Konzept "Attribut" eingeführt, um grundlegende Merkmale von Bildern zu definieren. Einem Attribut ist ein Wert zugeordnet, der durch eine zugehörige interne Prozedur

ermittelt wird. In L können die Attribute referenziert, berechnet und gespeichert werden. Als Beispiele für Attribute sind ein Histogramm oder verschiedene statistische Werte von Bildern angeführt.

- PascalPL [*Uhr 81*].
  PascalPL ist eine Erweiterung von PASCAL. Sie bietet Konstrukte zur parallelen Bearbeitung von Feldern an. Dafür wurden spezielle Deklarationen von parallel zu bearbeitenden Feldern eingeführt. Es gibt Operatoren sowie Zuweisungen und Kontrollstrukturen zur Bearbeitung derartiger Felder. Dabei können auch beliebige rechtekkige Ausschnitte verwendet werden. Weiterhin sind spezielle Ein-Ausgabeanweisungen verfügbar. PascalPL wird durch einen in PASCAL implementierten Präprozessor nach PASCAL transformiert.

- MAC [*Douglass 81*].
  MAC wurde speziell zur Entwicklung von asynchronen, parallelen Bildverarbeitungsalgorithmen entworfen. Die Syntax ähnelt den "distributed processes" [*Brinch Hansen 78*]. MAC versucht sowohl für "verteilte" Algorithmen als auch für SIMD-Algorithmen Konzepte anzubieten. Der Autor formuliert sechs Kriterien für den Entwurf von MAC:

  1. Die Möglichkeit zur Definition von identischen oder ähnlichen Prozessen, die auf verschiedenen Teilen einer großen gemeinsamen Datenstruktur arbeiten.
  2. Die Unterstützung von Parallelität auf der Ebene der Prozeduren.
  3. Die Fähigkeit zur dynamischen Erzeugung und Zerstörung von Prozessen und ihren Kommunikationsverbindungen.
  4. Formulierung von stark miteinander verkoppelter Verarbeitung.
  5. Die Möglichkeit von simultanen Lesezugriffen auf gemeinsame Daten bei mehreren Prozessen.
  6. Die Erzwingung von sequentiellen Schreibzugriffen auf gemeinsame Daten.

  Alle diese Kriterien wurden bei der Definition von MAC berücksichtigt. Die Sprache MAC definiert Konstrukte zur Definition von kooperierenden, parallelen Prozessen und zur Kontrolle ihrer Interaktion.

Ein MAC-Programm besteht aus einem Abschnitt zur Definition von Prozessen, gefolgt von Deklarationen für Prozeß-Instanzen und -verbindungen. Ein Prozeß kann repliziert werden über jedes Element einer Datenstruktur (beispielsweise ein Feld). Sie kommunizieren untereinander durch Prozedur- bzw. Funktionsaufrufe oder durch das Lesen von gemeinsamen Daten. Von den Prozessen wird vorausgesetzt, daß sie parallel auf einem virtuellen Prozessor ausgeführt werden.

4.1.8 PICAP

Das interaktive, hochsprachliche System zur Bildverarbeitung (PICAP Picture Processing Laboratory) [*Gudmundsson 82*] ist ein Repräsentant für Systeme, die auf einer eigenen Programmiersprache und in diesem Fall auch auf einer speziellen Gerätekonfiguration basieren.

PICAP besteht aus einem Überwachungsprogramm (Monitor), Texteditor, inkrementellen Übersetzer, Interpretierer und einer Prozedurbibliothek. Das System verfügt über einen speziellen Bildverarbeitungsprozessor, der parallel Transformationen auf Bildern durch 3×3 Nachbarschaftsoperatoren durchführen kann.

Die Bilder in PICAP haben ein festes Format von $64 \times 64 \times 4$ Bit. Der Bildverarbeitungsprozessor verfügt über 12 Register (Bildspeicher), in denen jeweils ein Bild gespeichert werden kann. Alle Bildoperationen liefern automatisch Ergebnisse wie ein Histogramm in 16 Stufen, minimaler und maximaler Grauwert usw.

Die Basis des Systems ist die strukturierte, höhere Programmiersprache PPL (Picture Processing Language). Sie lehnt sich in ihrer Struktur an ALGOL-60 an, enthält aber zusätzlich viele Elemente für die Bildverarbeitung. Dabei wird die zugrundeliegende spezielle Gerätestruktur berücksichtigt. An zusätzlichen Datentypen bietet sie binäre Bilder, Bilder (8 Bit), Maskenelemente und Masken, die zur Formulierung von 3×3 Bildoperatoren verwendet werden. Es gibt logische und arithmetische Operatoren. Die Operatoren können als Elemente ganzzahlige Ausdrücke, Vergleiche und bestimmte Variablen enthalten. Die Möglichkeit, als Operatorenelemente Variablen zu erlauben, wird zur Änderung der Operatoren zur Laufzeit verwendet. Durch Verkettung von Operatoren können weitere Operatoren gebildet werden.

Die interaktive Schnittstelle akzeptiert mit Hilfe eines integrierten Edi-

tors PPL-Anweisungen. Diese werden während der zeilenweisen Eingabe bzw. Modifikation syntaktisch geprüft und in eine Zwischensprache zur Interpretation übersetzt. Vollständige Programme oder Prozeduren werden in der Zwischensprache gespeichert und können sofort interpretiert werden. Die interaktive Schnittstelle unterscheidet zwei Modi. In dem einen ist der Editor aktiv, während in dem anderen Programme ausgeführt werden können.

In einem Neuentwurf von PICAP sind als Erweiterungen flexible Maskenoperatoren (nicht nur 3×3), flexiblere Datenstrukturen zur Bearbeitung von variablen Bildformaten sowie die Einführung einer Bilddatenbank geplant.

Nachfolgend sollen noch einige mit PICAP vergleichbare Systeme kurz vorgestellt werden:

- TAL [*Vrolijk et al. 81*].
  Das Leyden Television Analysis System (LEYTAS) dient zur Voranalyse von Halsabstrichen und zur automatischen Chromosomenanalyse. Die Bildverarbeitungsperipherie besteht aus einem automatischen Mikroskop, einem Bildprozessor und einem Bildspeicher für digitisierte Grauwertbilder. Die Programmierumgebung TAL erlaubt die Kontrolle von LEYTAS und dient als eine Art Betriebssystem.

  Die interaktive Schnittstelle von TAL ist menügesteuert. Es gibt Optionen, um die Peripherie zu kontrollieren oder um Programme zu erstellen. Die Programme können im Rahmen des Systems modifiziert und ausgeführt werden. Dabei wird als Testhilfe ein Einzelschritt-Modus angeboten. Die Programmiersprache von TAL ist von ihrer Funktionalität eine Teilmenge von FORTRAN IV, die dieselbe syntaktische Struktur hat. Es besteht die Möglichkeit, ein TAL-Programm nach FORTRAN zu transformieren, um es in bestehende FORTRAN-Programme zu integrieren.

- IFL [*Balston 81*].
  Die Sprache IFL (Instruction Forming Language) wird im Rahmen eines interaktiven Bildverarbeitungssystems für Geowissenschaftler eingesetzt. An Bildverarbeitungsperipherie besitzt dieses System eine TV-Kamera und 16 Bildspeicher ($512 \times 512 \times 8$ Bit). Weiterhin existiert ein spezieller Rechner, der die Interaktion mit dem Benutzer abwickelt und als Schnittstelle zu dem übrigen System gilt. Auf diesem

Rechner ist die Sprache IFL verfügbar. IFL erlaubt die Formulierung von Programmen, die Benutzer über Kommandos aufrufen können. IFL ist eine Variante der Sprache BASIC und wird zur Abarbeitung in eine Zwischendarstellung übersetzt. Die IFL-Programme realisieren Operationen, die den Benutzern über Kommandos zur Verfügung stehen.

4.2 Verwendete Interaktionsformen

Aus den dargestellten Systemen zur Bildverarbeitung und -auswertung können vier grundlegende Formen der Benutzerinteraktion abgeleitet werden. Diese sind auch in anderen Anwendungen aufzufinden. Sie können als exemplarisch für die in Bildverarbeitungs- und -auswertesystemen auftretenden Interaktionsformen angesehen werden.

Für fast alle Bildverarbeitungssysteme läßt sich sagen, daß sie auf einer Schnittstelle basieren, die sich an Kommandos orientiert. Der Benutzer hat eine bestimmte Anzahl von Befehlen zur Auswahl, mit denen er seine Aufgaben durchführen kann. Der Umfang sowie die Form und Bedeutung der verfügbaren Befehle hängt oft von bestimmten Zuständen des Systems ab.

Die häufigste Form der befehlsorientierten Interaktion kann als konventionelle Interpretation von "einfachen" Kommandos klassifiziert werden. Diese Interaktion findet als eine lineare Folge von Befehlen statt. Sie orientiert sich an den Fähigkeiten von alphanumerischen Terminalen. Eine Variante der kommandogesteuerten Interpretation zeichnet sich dadurch aus, daß sie die im Gegensatz zu anderen verwendeten Sprachen vergleichsweise mächtige Kommandosprache von Unix verwendet. Eine alternative Form zur linearen Interpretation ist die Menüsteuerung. Dem Benutzer werden die zu einem Zeitpunkt möglichen Kommandos als Menü dargestellt, aus dem er eine Option auswählt. Eine weitere Interaktionsform, die in fast allen Systemen vorkommt, ist ein programmgesteuertes Frage-Antwort-Schema.

Diese verwendeten Interaktionsformen werden nachfolgend genauer dargestellt. Daran schließt sich eine Erörterung von verwendeten Interaktionskonzepten an. Die Bedeutung der Interaktionsformen und der Interaktionskonzepte für die Bildverarbeitungssysteme wird beurteilt. Abschließend findet eine Bewertung der betrachteten Systeme und ihrer Benutzerschnitt-

stellen nach den in Kapitel 3.3 dargelegten Konzepten statt.

4.2.1 Interpretation von einfachen Kommandos

Die häufige Verbreitung dieser Interaktionsform hat mehrere Gründe. Einmal kann sie als eine Folge der historischen Entwicklung von Bildverarbeitungssystemen angesehen werden. Die ersten Systeme wie beispielsweise das PAXII-System [*Johnston 70*] besaßen keinen interaktiven Zugang. Sie wurden als reine Unterprogrammbibliotheken konzipiert, die nur über eine Gastsprache (meistens FORTRAN) und üblicherweise im Stapelbetrieb verwendet werden konnten. Später wurde ein interaktiver Zugang zu den Unterprogrammbibliotheken geschaffen. Deren Unterprogramme konnten damit vom Benutzer interaktiv gestartet werden. Ein weiterer Grund für die häufige Verbreitung der Kommando-Interpretation liegt in dem relativ geringen Aufwand zur Erstellung einer solchen Schnittstelle.

Die von derartigen Interpretierern akzeptierte Sprache ist von ihrer syntaktischen Struktur her einfach gehalten. Die Befehle bestehen meistens aus einem Schlüsselwort, teilweise ist auch die Angabe von Parametern gestattet. Der Wertetyp der Parameter beschränkt sich in der Regel auf numerische oder textuelle Größen. Eine Überprüfung der Parameter auf ihre korrekte Syntax findet durch den Interpretierer statt, ihre semantische Integrität hat jedoch meistens das Anwendungsprogramm zu überprüfen. Ein Arbeitsspeicher ist für Daten während der Interaktion selten verfügbar. Die Befehlsinterpretation ist als kontextfrei anzusehen, d.h. die Befehle definieren in sich abgeschlossene Aktionen, die auf den Zustand der Befehlsinterpretation keine Auswirkungen haben.

Die Benutzerschnittstellen der Bildverarbeitungssysteme unterscheiden sich durch ihre Fähigkeiten und den damit verbundenem Komfort für den Benutzer. VICAR [*Castleman 79*] und KANDIDATS [*Haralick & Minden 78*] definieren eine einfache Kommandosprache ohne jeden Komfort. Das KIPL-System [*Fukushima et al. 81*] bietet die Definition von Makro-Prozeduren mit Parameterbeschreibungen und Voreinstellungen für deren Werte. Das PICASSO-System [*Kulpa 81b*] verfügt ebenso wie KIPL über einen Editor, mit dem die Kommandoprozeduren erstellt oder modifiziert werden können. CELLO [*Bengtsson et al. 81*] gestattet das Verwenden von Variablen, während PIC [*Engelmann & Meinzer 84*] zusätzlich noch Kontrollstrukturen anbietet.

DIBIAS [*Triendl et al. 82*] aktiviert die Unterprogramme als Unterpro-

zesse und bleibt dadurch interaktiv verfügbar.

Eine größere Flexibilität in der Interaktion ist in Systemen möglich, die eine eigene Programmiersprache anbieten. Diese wird meistens interpretiert und gestattet eine angemessene Formulierung von Kontrollstrukturen sowie die Deklaration von Variablen. TAL [*Vrolijk et al. 81*] verwendet eine Teilmenge von FORTRAN, IFL [*Balston 81*] ist eine Variante von BASIC, während PICAP [*Gudmundsson 82*] eine Erweiterung von ALGOL-60 interpretiert.

4.2.2 Interpretation von Unix-Kommandos

Eine andere Form der Kommando-Interpretation findet sich in Systemen, die in einer Unix-Umgebung ablaufen. Dort liegt es nahe, die Kommandosprache von Unix zur Interaktion und Manipulation von Bildverarbeitungssystemen zu verwenden. Das HIPS-System [*Landy et al. 84*] hat diesen Ansatz verfolgt, während das BROWSE-System [*McKeown & Denlinger 82*] sich an der Unix-Kommandosprache orientiert hat. Ein anderes Bildverarbeitungssystem [*Petters 85*] nutzt ebenfalls die Vorteile von Unix. Die Gerätestruktur und Programmkomponenten dieses Bildverarbeitungssystems wurden in eine Unix-Umgebung eingebettet. Das System dient der Analyse industrieller Szenen.

Die Verwendung der Kommandosprache von Unix bringt mehrere Vorteile mit sich. Der Aufwand zur Erstellung einer Kommando-Interpretation fällt weg. Der Benutzer kann sich der gewohnten Kommandosprache bedienen. Diese ist von ihren Möglichkeiten her sehr mächtig und kann schon fast als Programmiersprache angesehen werden. Durch die bekannten Unix-Konzepte (filter, pipe) ist eine schnelle und flexible Verknüpfung von Kommandos und Programmen möglich.

Der Systementwurf von Unix ist elegant, dieses kann man aber nicht von der interaktiven Schnittstelle behaupten [*Norman 81*]. Die Syntax und Semantik der Kommandosprache ist keinesfalls vorbildlich und kann auch nicht als benutzerfreundlich bezeichnet werden. Ihre elegante Anwendung durch HIPS demonstriert jedoch, wie gut sich die Unix-Konzepte der parallel arbeitenden Filter, verknüpft durch Leitungen, auf den Bereich der Bildverarbeitung anwenden lassen.

Ein Beispiel für eine typische Kommandofolge in HIPS lautet [*Landy et al. 84*] (zur Unix-Kommandosprache siehe Kapitel 2.1):

```
grerase; tvc
rseq 120 -l "ASL sent. 1" | reduce 4 | extract 96 64 20 12 > seq.
```

Diese Notation mag für einen mit dem HIPS-System vertrauten Benutzer verständlich sein, sofern die Programmnamen plausibel gewählt wurden, für einen Außenstehenden ist diese Befehlsfolge jedoch kaum zu verstehen.

Diese Befehlsfolge bewirkt das Einlesen von 120 Bildern von einem Film, die mit einem Grinnell-System digitisiert werden. Diese digitisierten Bilder werden kompaktifiziert und als Bildfolge mit einem Format von 96×64 Pixel pro Bild gespeichert.

Um den interaktiven Umgang mit HIPS zu erleichtern, wurden Parameterangaben für Kommandos weitgehend reduziert. Die meisten Programme von HIPS benötigen deshalb keine interaktiven Eingaben zur Durchführung ihrer Operationen. Dies wurde dadurch erreicht, daß die meisten benötigten Parameter aus dem Bildkopf gewonnen werden (siehe auch Kapitel 4.1.4) oder durch nützliche Voreinstellungen für ihre Werte nicht explizit angeführt werden müssen.

Ein weiterer Vorteil bei der Verwendung dieser Konzepte zeigt sich in der Ausführungszeit derartiger Systeme. Eine Studie [*Stevens & Hunt 82*] hat gezeigt, daß sich die benötigte Ausführungszeit bei einer Bildverarbeitungsanwendung durch die Verwendung von Filtern und Leitungen gegenüber der Verwendung von temporären Dateien um bis zu 1/3 verringern kann. Dies ist eine Folge der Unix-Strategie, die die Puffer für die Leitungen im Hauptspeicher anlegt. Dadurch werden bei der Kommunikation zeitaufwendige Datei-Operationen überflüssig.

An dem Beispiel von HIPS zeigen sich mehrere Vorteile der Unix-basierten, interaktiven Bildverarbeitung:

1. Das Konzept der Filter, die einfache Transformationen durchführen und die eine einfache Programmstruktur besitzen, läßt sich gut auf den Bereich der Bildverarbeitung anwenden.

2. Die Möglichkeit, einfache Bildtransformationen durch die Verwendung von Leitungen leicht zu kombinieren, unterstützt den Benutzer bei seiner interaktiven, experimentellen Arbeit der Bildverarbeitung.

3. Die Zerlegung von komplexen Bildverarbeitungssystemen in Komponenten, die über Datenkanäle kommunizieren und im Rahmen des Systems in sich abgeschlossene Teilaufgaben bearbeiten, bestätigt das in Kapitel 3.1 dargelegte Systemmodell.

4. Die leichte Erweiterbarkeit des HIPS-Systems in Form von Verarbeitungsprogrammen als Filter und ihre Verfügbarkeit im Rahmen einer Programmbibliothek demonstriert die Notwendigkeit einer Programmierumgebung für die Bildverarbeitung, die derartige Konzepte anbieten sollte.

Die Vorteile der Unix-basierten Bildverarbeitung lassen sich nach meiner Meinung auf den Bereich der Bildfolgenauswertung übertragen.

4.2.3 Steuerung durch Menüs

Eine andere Form der Kommando-Interpretation ist die Menüsteuerung. Dabei wird dem Benutzer zur Interaktion ein Menü angeboten. Dieses Menü stellt die zu einem Zeitpunkt verfügbaren Befehle dar. Der Benutzer kann aus den angebotenen Optionen eine auswählen. Diese Auswahl kann entweder durch die Bezeichnung einer Option über ein Symbol (oft ein Zahlencode) oder bei der graphischen Darstellung des Menüs auch über ein Zeigeinstrument erfolgen.

In Bildverarbeitungssystemen wird die Menüsteuerung teilweise als Alternative zur linearen Kommando-Interpretation angeboten. In PIC [*Engelmann & Meinzer 84*] kann der Benutzer entscheiden, ob er als Betriebsart die Menüsteuerung oder die herkömmliche Interpretation benutzen möchte. Das TAL-System [*Vrolijk et al. 81*] besitzt eine interaktive Schnittstelle, die vollständig auf der Menüsteuerung basiert. Das MORIO-System [*Dreschler-Fischer et al. 83*] bietet dem Benutzer eine Menüsteuerung, mit der er den Verarbeitungsablauf überwachen und beeinflussen kann.

Die zu einem Zeitpunkt verfügbaren Operationen sind meistens so umfangreich, daß sie nicht vollständig auf der begrenzten Bildschirmfläche darstellbar sind. Deshalb werden die Operationen üblicherweise zu Klassen zusammengefaßt und hierarchisch organisiert. Der Benutzer erhält somit einen Baum von Menüs angeboten. Er kann seine aktuelle Position in diesem Menübaum durch spezielle Operationen verändern. Dies erfolgt durch die Auswahl einer Folge von Menüs, die nacheinander dargestellt werden. Diese Positionierung in dem Baum von Menüs wird auch als *Navigation* bezeichnet.

Ein Beispiel für ein interaktives System, welches hauptsächlich auf dem Konzept der Menüs basiert, ist das ZOG-System. Es besitzt eine große Datenbasis von Menüs und bietet eine schnelle Reaktion auf die Menüauswahl.

Es wurde sowohl für unerfahrene Benutzer als auch für Experten konzipiert und verfügt über einen einzigen Schnittstellenmechanismus, der alle von einem Benutzer benötigten Funktionen umfaßt [*McCracken & Akscyn 84*].

Das ZOG-System wurde als allgemeines Schnittstellensystem für umfangreiche interaktive Anwendungen entworfen. Es ist in der Lage, mehrere 10000 Menüs zu verwalten und besitzt eine Reaktionszeit auf die Menüauswahl, die gut unter einer Sekunde liegt.

Das Konzept der Menüsteuerung besitzt sowohl Vorteile als auch Nachteile, die sich auch aus den Erfahrungen mit der interaktiven Schnittstelle von ZOG [*McCracken & Akscyn 84*] ergeben:

- Das Konzept der Menüsteuerung ist einfach zu realisieren und leicht zu erweitern.
- Für unerfahrene Benutzer erleichtert es den Umgang mit dem System.
- Viele Fehlerquellen bei der Benutzerinteraktion werden ausgeschlossen.
- Umfangreiche Operationsdatenbasen werden unterstützt.

Als Nachteile ergeben sich:

- Durch die Vorgabe eines Menübaums sind die verfügbaren Operationen vorstrukturiert. Diese Strukturierung muß aber nicht für alle Anwendungen adäquat sein.
- Die Suche im Menübaum und dessen Darstellung wird in den meisten Systemen in Form einer Breitensuche angeboten. Diese Zugriffsform schränkt den Benutzer aber in seiner Arbeit ein. Die Navigation durch den Menübaum wird insbesondere für erfahrene Benutzer dann lästig, wenn zur Durchführung einer Operation erst mehrere Menüs passiert werden müssen.
- Das Problem der Desorientierung kann insbesondere bei unerfahrenen Benutzern während der Navigation auftreten.

Bei Bildverarbeitungssystemen sind einige der oben erwähnten Probleme von geringer Bedeutung. Da der Umfang der vorhandenen Menüs wohl im Vergleich zu ZOG-Anwendungen als klein anzusehen ist, spielt das Problem der Desorientierung, d.h. ein Benutzer hat sich in dem Menübaum

verirrt, während der Navigation keine Rolle. Die Zahl der Menüselektionen kann durch benutzerdefinierte Abkürzungen im Menübaum reduziert werden. Durch die Verwendung von mehreren permanenten Fenstern können verschiedene Menüs parallel verfügbar sein.

Zusammenfassend läßt sich sagen, daß die Menüsteuerung in Bildverarbeitungssystemen überall dort sinnvoll eingesetzt werden kann, wo der Benutzer eine überschaubare Menge von gleichartigen Optionen zur Auswahl hat. Als Grundvoraussetzung für die Verwendung von Menüs muß gelten, daß die zur Darstellung eines Menüs benötigte Zeitspanne signifikant unter einer Sekunde liegt. Weiterhin sollte die Menüstruktur flach bleiben, d.h. eine tief strukturierte hierarchische Organisation ist zu vermeiden. Die Darstellung und Auswahl der Menüs sollte mit einem hochauflösenden Farbrastergraphik-System und entsprechenden Zeigeinstrumenten erfolgen. Um das Grundkonzept der direkten Manipulation durch den Benutzer nicht zu unterlaufen, muß die Häufigkeit der Menüaktivierung gering gehalten werden.

4.2.4 Frage-Antwort-Dialog

In fast allen Bildverarbeitungssystemen kommt eine Interaktionsform vor, die hier als Frage-Antwort-Dialog bezeichnet werden soll. Dabei geht die Initiative für die Dialogführung von der Verarbeitungskomponente aus. Diese stellt dem Benutzer in einer fest vorgegebenen Reihenfolge Fragen, die dieser zur Fortführung der Verarbeitung beantworten muß.

Dieses Frage-Antwort-Schema wird häufig zur lokalen Ablaufsteuerung oder zur Parameterinteraktion (siehe Kapitel 4.3.3) verwendet. Die von der Komponente erfragten Werte sind meistens durch eine Voreinstellung besetzt, so daß der Benutzer im Zweifelsfall diese gewählten Werte nur durch einen Tastendruck bestätigen muß.

Im KANDIDATS-System wird ein Frage-Antwort-Dialog von den aktivierten Verarbeitungskomponenten benutzt, um weitere Verarbeitungsparameter zu erfragen. Im MORIO-System existiert ein derartiger Dialog während der Systeminitialisierung. Er dient dort der Bestimmung von globalen Systemparametern.

Die häufige Verwendung dieser Dialogform liegt in der einfachen Realisierbarkeit begründet. Sie hat jedoch zwei entscheidende Nachteile. Der Ablauf und die Form der Interaktion sind in der fragenden Komponente realisiert und damit nur durch deren Veränderung zu beeinflussen. Aus der

Sicht des Benutzers ist diese Interaktionsform ermüdend, da er gezwungen wird, dieses Frage-Antwort-Schema in einer von ihm nicht beeinflußbaren Weise zu durchlaufen. Der Benutzer kann bei dieser Interaktionsform nur auf die Ausgaben des Systems reagieren. Die Interaktion stellt sich für ihn als passiver Vorgang dar, der ihm wenig Freiraum zur Dialoggestaltung läßt.

Bietet das System für die Fragen durch eine Voreinstellung schon die vom Benutzer gewünschten Antworten, so wird dieser Dialog in vielen Fällen sogar überflüssig. Der Dialog kann aber nicht wegfallen, da es bei bestimmten Anwendungen denkbar ist, daß die angebotenen Voreinstellungen für die Antworten nicht gewünscht werden.

4.3 Verwendete Interaktionskonzepte

In Kapitel 3.2 wurden die Ziele der Benutzer bei der Interaktion mit den Bildverarbeitungssystemen dargestellt. Zur Erreichung der Benutzerziele während der Interaktion wurden Konzepte entwickelt und in Bildverarbeitungssystemen realisiert. Es bleibt zu prüfen, ob die dort verwendeten Konzepte die formulierten Benutzerziele abdecken.

Diese Konzepte sollen im weiteren als *Interaktionskonzepte* bezeichnet werden. Ein Interaktionskonzept umfaßt die Bedeutung von Handlungen, die der Benutzer zur Durchsetzung seiner Ziele während der Interaktion vollführt. Ein Interaktionskonzept ist somit eine semantische Beschreibung von Interaktionshandlungen. Dabei spielt es keine Rolle, in welcher Form die dem Interaktionskonzept zugeordneten Handlungsoptionen dargestellt werden. Bei der Realisierung eines Interaktionskonzeptes können mehrere der in Kapitel 4.2 beschriebenen Interaktionsformen verwendet werden.

In den oben beschriebenen Beispielsystemen sind verschiedene Interaktionskonzepte verwendet worden. Dabei ist bemerkenswert, daß diese Konzepte nicht als völlig disjunkt angesehen werden können und daß oft sogar mehrere Konzepte bei einer Handlung auftreten können.

4.3.1 Ablaufsteuerung

Ein wichtiges Interaktionskonzept ist die Ablaufsteuerung. Die interaktive Ablaufsteuerung ist nur in Bildverarbeitungssystemen sinnvoll, die einen gewissen experimentellen Charakter besitzen. Bei einem für eine

ganz spezielle Anwendung dedizierten System ist eine interaktive Ablaufsteuerung nicht notwendig, da sie dann fest in dem System integriert ist.

Experimentalsysteme dagegen bieten allgemeiner einsetzbare Funktionen an. Zur Durchführung von Experimenten muß der Benutzer die Möglichkeit haben, eine interaktive Ablaufsteuerung auszuüben. Dabei lassen sich zwei Formen der Ablaufsteuerung unterscheiden:

- *Global*: Der Benutzer spezifiziert die Ablaufsteuerung durch die Auswahl und Aktivierung von Systemkomponenten (z.B. Filtern, Unterprogrammen, Prozessen). Die globale Ablaufsteuerung regelt somit den Ablauf zwischen den Systemkomponenten.

- *Lokal*: Der Benutzer spezifiziert die Ablaufsteuerung innerhalb einer Systemkomponente durch die Formulierung von Anweisungen oder durch die Parameterinteraktion (siehe Kapitel 4.3.3).

In den Bildverarbeitungssystemen, die eine Kommandosprache als interaktive Schnittstelle besitzen (siehe auch Kapitel 4.1), ist die globale Ablaufsteuerung meistens einfach geregelt. Dies liegt daran, daß diese Systeme üblicherweise einen interaktiven Zugang zu einer umfangreichen Bibliothek von Verarbeitungsprogrammen bieten. Deshalb ergibt sich die globale Ablaufkontrolle durch die Reihenfolge der eingegebenen Kommandos. Als Beispielsystem sei hierfür KANDIDATS [*Haralick & Minden 78*] (siehe auch Kapitel 4.1.1) genannt. Falls eine lokale Ablaufsteuerung möglich ist, wird sie durch die Parameterinteraktion ausgeübt.

Einige Systeme erlauben die Möglichkeit zur Definition von Kommandoprozeduren (beispielsweise KIPL [*Fukushima et al. 81*], VICAR [*Castleman 79*], CELLO [*Bengtsson et al. 81*] und DIBIAS [*Triendl et al. 82*]). Der Benutzer kann seine Aktionen strukturieren, indem er bestimmte Aktionen in Prozeduren zusammenfaßt und damit eine sehr einfache Ablaufsteuerung ausübt. Diese Steuerung wird effektiver und universell, wenn die Kommandosprache Konstrukte bereitstellt, die eine Ablaufkontrolle in Form von Sprüngen, Verzweigungen und Schleifen erlaubt (ein Beispiel dafür ist PIC [*Engelmann & Meinzer 84*]). Dabei ist jedoch zu beachten, daß in diesem Kontext eine *globale* Ablaufsteuerung gemeint ist im Gegensatz zu Systemen mit einer eigenen Programmiersprache (beispielsweise PICAP [*Gudmundsson 82*]). Dort wird eine derartige Ablaufsteuerung überwiegend *lokal* ausgeübt, ebenso wie in speziellen Bildverarbeitungssprachen (z.B. PIXAL [*Levialdi et al. 81*]). PICAPL [*Dengler & Meinzer 85*] gestattet

sowohl eine lokale als auch eine globale interaktive Kontrolle, da dort die Verarbeitungskomponenten als APL-Programme vorliegen und vom Anwender jederzeit modifiziert werden können.

Eine sehr kompakte und elegante Notation zur Ablaufsteuerung findet sich in dem Unix-basierten System HIPS [*Landy et al. 84*] (siehe auch Kapitel 4.1.4). Dort sind die Verarbeitungskomponenten in Form von Filtern verfügbar. Der Benutzer kann die globale Ablaufsteuerung mithilfe der Unix-Kommandosprache ausüben. Unter Verwendung von Leitungen können Filterprogramme parallel ausgeführt und als Fließbandverarbeitung kombiniert werden. Die Synchronisation der Filter zum Datenaustausch wird durch Unix überwacht. Eine sequentielle Steuerung über Schleifen o.ä. ist in der Kommandosprache leicht zu realisieren.

Ein wichtiges Ziel der Ablaufsteuerung steht im engen Zusammenhang mit der Protokollinteraktion (siehe Kapitel 4.3.4). In bestimmten Situationen ist es wünschenswert, den Verarbeitungsablauf einer Komponente oder sogar des gesamten Systems zu unterbrechen, um den momentanen Zustand der Komponenten zu erhalten. Dies ist beispielsweise dann notwendig, wenn der Benutzer den momentanen Zustand der Systemkomponenten inspizieren will. Ein System, welches diese Art der Ablaufsteuerung anbietet, ist MORIO (siehe auch Kapitel 4.1.6). Dabei wird als Interaktionsform eine Menüsteuerung verwendet.

Mit Ausnahme von HIPS und PIC sind die Strukturen zur Ablaufsteuerung bei den kommandogesteuerten Systemen nicht sehr umfangreich. Als Interaktionsform werden einfache Kommandos oder Menüs verwendet. Die globale Ablaufsteuerung wird meistens nur durch die Reihenfolge der während der Interaktion eingegebenen Befehle ausgeübt. In MORIO wird ein Menü zur Ablaufsteuerung benutzt. Die lokale Ablaufsteuerung ist entweder in den Systemkomponenten fest verankert, oder sie kann in geringem Ausmaß über die Parameterinteraktion beeinflußt werden. Die Systeme, die eine eigene oder erweiterte Programmiersprache interpretieren, ermöglichen die Ablaufsteuerung durch eine schnell wirksame Modifikation der Komponenten.

4.3.2 Datenflußsteuerung

In Bildverarbeitungssystemen und insbesondere in Bildfolgenauswertesystemen spielt die Datenflußsteuerung eine wichtige Rolle. Die zu bearbeitenden Datenmengen sind oft sehr umfangreich und die Möglichkeiten ihrer

Bearbeitung vielfältig. Die Datenflußsteuerung steht im engen Zusammenhang mit der Ablaufsteuerung. Bei der Datenflußsteuerung werden die zu bearbeitenden Daten ausgewählt und die Reihenfolge ihrer Abarbeitung festgelegt. Dies erfolgt durch die Spezifikation der Datenquellen, meistens Dateien, und der Datensenken. Diese werden konzeptuell durch Datenpfade miteinander verbunden, über die der Datenfluß geleitet wird.

Eine direkte Realisierung dieses Leitungskonzeptes findet sich in dem HIPS-System. Bei den meisten Bildverarbeitungssystemen wird der Datenfluß indirekt gesteuert. Die dort verfügbaren Operationen werden oft als Funktionen angesehen, die als Eingabe ein Bild erhalten und als Ausgabe wieder ein neues produzieren (z.B. das KANDIDATS-System).

Das KANDIDATS-System versteht Bilder als Dateien, die in einem bestimmten Format gespeichert sind. Sie übernehmen dort die Rolle der Datenquellen und -senken. Ähnlich behandelt auch DIBIAS die Bilder. Dort werden Bilder über symbolische Namen referenziert. Der Benutzer kann Verarbeitungsprogramme aktivieren und diesen als Parameter die Bildnamen übergeben. Diese Programme sind derart strukturiert, daß sie als Eingabe ein bis mehrere Bilder erhalten und eventuell ein bis mehrere Bilder als Ausgabe erzeugen. Die Möglichkeit des Benutzers zur Datenflußsteuerung beschränkt sich somit auf die Auswahl der Quellen und Senken.

Das BROWSE-System [*McKeown & Denlinger 82*] (siehe auch Kapitel 4.1.5) gestattet dem Benutzer die Datenflußsteuerung zwischen dem System und den aktivierten Verarbeitungsprogrammen über graphisch dargestellte Fenster. Das zu bearbeitende (Teil-) Bild wird in einem Fenster dargestellt. Der Benutzer kann dieses Fenster über ein Zeigeinstrument oder einen symbolischen Namen kennzeichnen und eine Komponente aktivieren. Dieser werden die zu dem Fenster gehörenden Daten übermittelt. Für die Rückgabe der Ergebnisse kann der Benutzer ebenfalls ein Fenster spezifizieren. Die in dieser Form erhaltenen Ergebnisse können dann im Rahmen des BROWSE-Systems zur weiteren Bearbeitung verwendet werden. Die Implementation der Datenflußsteuerung erfolgt über Dateien, die zum Datentransfer verwendet werden.

In dem MORIO-System ist eine Steuerung des Datenflusses nur bedingt möglich. Der Benutzer kann die Datenquellen spezifizieren, in diesem Fall eine Bildfolge. Der Datenfluß zwischen den Komponenten des Systems ist fest vorgegeben und kann interaktiv nicht verändert werden.

Als Interaktionsformen bei der Datenflußsteuerung finden sich einfache oder Unix-Kommandos (z.B. KANDIDATS, PIC, HIPS), Frage-Antwort-

Dialoge (z.B. MORIO) sowie die Unterstützung der Angabe von Quellen und Senken über graphisch dargestellte Fenster (z.B. BROWSE).

Zusammenfassend läßt sich sagen, daß die Datenflußsteuerung für die experimentelle Bildverarbeitung besonders wegen der großen anfallenden Datenmengen ein wichtiges Interaktionskonzept ist. Mit Ausnahme von HIPS sind jedoch in den bekannten Bildverarbeitungssystemen die Möglichkeiten der Benutzer gering, den Datenfluß interaktiv zu steuern. Die Notwendigkeit für eine flexible interaktive Steuerung und die daraus resultierende erhöhte Funktionalität dieser Systeme werden an dem Beispiel von HIPS und BROWSE gut demonstriert. Das geringe Angebot dieser Möglichkeiten liegt wohl in dem enormen Aufwand für die Realisierung begründet. Im Rahmen der Unix-Umgebung wird durch die Fähigkeiten des Betriebssystems dieser Aufwand relativ gering gehalten.

4.3.3 Parameterinteraktion

Ein weiteres wichtiges Konzept soll hier als Parameterinteraktion bezeichnet werden. Dieses Interaktionskonzept wird in der Bildverarbeitung häufig angewendet. Darunter wird die Steuerung von Verarbeitungskomponenten über veränderbare Datenobjekte dieser Komponenten verstanden. Diese werden als Parameter der Komponenten angesehen. Diese Form der Steuerung unterstützt das experimentelle Entwickeln und Testen von Algorithmen und erlaubt deren leichte Anpassung für spezielle Aufgaben.

Üblicherweise und besonders bei den Bildverarbeitungssystemen, die sich an einer Kommandosprache orientieren, sind die Parameter mit den Kommandos assoziiert, die die Verarbeitungskomponenten aktivieren. Die Anzahl dieser Parameter ist oft umfangreich. Eine ständige Angabe dieser Parameter durch den Benutzer behindert den interaktiven Umgang. Deshalb bieten fast alle Systeme eine Vorbesetzung dieser Parameter an. Die Vorbesetzung der Parameterwerte ist meistens durch das System vorgegeben. Einige Systeme erlauben dem Benutzer, die Voreinstellungen nach seinen eigenen Wünschen zu modifizieren. Der Benutzer kann beim Aufruf von Kommandos Parameterwerte auslassen, wenn die Parameter eine Voreinstellung besitzen. Diese Voreinstellungen werden vom System dann automatisch als Parameterwerte ergänzt. Die Definition von Voreinstellungen ist auch dadurch begründet, daß im Rahmen eines Experimentes die Werte der meisten Parameter über einen längeren Zeitraum unverändert bleiben.

Der Wertetyp dieser Parameter ist meistens skalar (d.h. ganzzahlig), reell, logisch, oder eine Zeichenkette. In DIBIAS können Bilder über Namen und Vektoren als Parameter auftreten. In PIC sind die Bildnamen keine Parameter. Die Bilder stehen unter festgelegten Namen im Arbeitsspeicher zur Verfügung und werden von den Komponenten direkt referenziert. In PIC ist jeweils einer Klasse von Operationen (z.B. Bilddarstellung, Operationen im Fourier-Raum) ein fester Satz von globalen Parametern zugeordnet. Sie besitzen eine vom System fest vorgegebene Voreinstellung, die der Benutzer interaktiv verändern kann. Durch ihren Wert werden bestimmte Operationen gesteuert. Dem Benutzer werden während der Interaktion die Abhängigkeiten zwischen den Operationen und den zugehörigen Parametern nicht verdeutlicht. Sie stellen sich dem Benutzer als global verfügbare Variablen dar. Als Interaktionsform zur Veränderung dieser Parameter werden einfache Kommandos verwendet. PIC verfügt weiterhin über einen einfachen Menümodus, der vom Benutzer aktiviert werden kann. In diesem Modus wird dem Benutzer bei fehlerhaften oder unvollständigen Eingaben eine Erläuterung über die erwarteten Kommandoparameter gegeben.

Das KIPL-System [*Fukushima et al. 81*] realisiert die Voreinstellung der Parameter in den Kommandoprozeduren. Es gestattet die Definition von "fixierten" Parametern. Das System setzt für die aktuellen Werte derartiger Parameter eine Voreinstellung ein. Der Benutzer kann die Voreinstellungen nach seinen Wünschen ergänzen oder modifizieren. Die Wirksamkeit dieser Änderungen gilt allerdings nur für die Dauer der interaktiven Sitzung. Der Benutzer kann jedoch diesen Satz von Voreinstellungen abspeichern und zu Beginn einer neuen Sitzung restaurieren.

Im DIBIAS-System besitzen die meisten Parameter eine Voreinstellung. Bei der Eingabe von vollständigen Kommandos, d.h. mit Eingabe- und Ausgabebildern, wird der Befehl ohne eine weitere Interaktion ausgeführt. Bei einer unvollständigen Eingabe wird dem Benutzer ein Menü angeboten, das ihm alle verfügbaren Parameter des eingegebenen Kommandos mit ihrer Voreinstellung und dem gültigen Wertebereich anzeigt. Unter Angabe des Parameternamens kann er einen von der Voreinstellung abweichenden Wert angeben. Nach der Beendigung seiner Parameterwahl wird das Kommando mit den im Menü angeführten Parameterwerten durchgeführt.

Beim HIPS-System wurde auf eine Parameterinteraktion fast vollständig verzichtet. Der Benutzer hat nur die Möglichkeit, durch die Angabe von Werten die mit dem Kommando assoziierten Voreinstellungen zu überschreiben. Dabei findet jedoch im Gegensatz zur Parameterinteraktion kein

weiterer interaktiver Vorgang statt, um Parameterwerte zu besetzen.

An den erörterten Beispielsystemen zeigt sich, daß die Parameterinteraktion ein wichtiges Konzept zur Steuerung von Bildverarbeitungssystemen ist. Zur Interaktion mit dem System werden dem Benutzer als Interaktionsformen einfache Befehle (z.B. PIC), Frage-Antwort-Dialog (z.B. KANDIDATS, MORIO) und Parametermenüs (z.B. CELLO [*Bengtsson et al. 81*]) angeboten.

Die meisten Systeme gestatten die Definition von Voreinstellungen für Parameterwerte. Diese Voreinstellungen können aber oft nicht permanent verändert und benutzerspezifisch gespeichert werden. Bei den meisten Systemen führt die Benutzerschnittstelle während der Parameterinteraktion nur eine syntaktische Analyse der Eingaben durch. In dem KANDIDATS-System beispielsweise bleibt es den aktivierten Prozeduren überlassen, die Vollständigkeit und Konsistenz der Parameter zu überprüfen. Diese Prozeduren führen mit dem Benutzer einen eigenen Dialog, um weitere Parameter zu erfragen. Entsprechendes gilt auch für die meisten anderen Systeme.

Diese Form der Parameterinteraktion hat mehrere Nachteile:

- Der Entwickler von Bildverarbeitungsoperationen wird mit den Details der Parameterinteraktion belastet.
- Die einheitliche Gestaltung von Dialogen wird dadurch erschwert.
- Durch die fehlende funktionelle Trennung von Interaktion und Verarbeitung ist die Form der Dialogführung durch die Operationen festgelegt und nicht unabhängig veränderbar.
- Die Komplexität der Bildverarbeitungsoperationen wird durch die Realisierung der Parameterinteraktion unnötig erhöht.
- Die Werte der Parametervoreinstellungen sind oft fest in den Operationen verankert.

4.3.4 Protokollinteraktion

In der Bildverarbeitung und insbesondere in der Bildfolgenauswertung benötigen die durchzuführenden Operationen oft einen erheblichen Aufwand an Rechenzeit. Der Benutzer muß deshalb oft lange auf die Ausgabe seiner endgültigen Ergebnisse warten. Gerade in dem Bereich der Experimentalsysteme ist jedoch der Erfolg einer Operation ungewiß.

Zur Vermeidung unnötiger Wartezeiten ist es deswegen sinnvoll, wenn der Benutzer den Fortgang der Verarbeitung beobachten und überwachen

kann. Diese Überwachung erfolgt üblicherweise durch ein Protokoll, welches in Form von Zwischenergebnissen den Fortgang der Verarbeitung signalisiert. Diese Ausgabe von Zwischenergebnissen soll natürlich nicht bei jedem Experiment und auch nicht für jede Komponente des Systems erfolgen. Deshalb ist es vernünftig, wenn der Benutzer dieses Protokoll interaktiv steuern kann.

Die Protokollinteraktion kann als eine Synthese der Ablauf- und Datenflußsteuerung angesehen werden. Sie wird aber trotzdem als eigenständiges Interaktionskonzept verstanden, da sie nicht in den funktionellen Ablauf des Systems eingreift, sondern eine Überwachung des Systemablaufs anbietet.

Bei den meisten Systemen, die auf einer Kommandosprache basieren, werden die Zwischenergebnisse auf einem Darstellungsmonitor angezeigt. Diese Darstellung kann meistens nicht beeinflußt werden, was eine Folge der kommandogesteuerten Interaktion ist. Dabei wird vorausgesetzt, daß für die Ausführung eines Kommandos nur ein relativ kurzer Zeitraum benötigt wird, nach dem der Benutzer sofort auf die Ausgabe reagieren kann.

Das HIPS-System ist zwar auch kommandogesteuert, aber dort läßt sich leicht eine komplexe Fließbandverarbeitung über Datenleitungen erzeugen. Benötigt der Benutzer im Rahmen einer Fließbandverarbeitung eine Testausgabe, so kann er dies leicht dadurch erreichen, daß er die Fließbandverarbeitung an den gewünschten Stellen um Darstellungsprogramme ergänzt.

Im MORIO-System zur Auswertung von Bildfolgen wurde — bedingt durch die Implementationssprache PASCAL und das Fehlen entsprechender Kommandosprachen — ein anderer Weg beschritten. Dort kann der Benutzer interaktiv an vom System vorgegebenen Stellen im Verarbeitungsablauf Testausgaben aktivieren (siehe auch Kapitel 4.1.6). Als Interaktionsform wird ein Menü verwendet, aus dem der Benutzer mithilfe eines Zeigeinstrumentes (Rollkugel) eine Option auswählen kann. Durch den speziellen Problemkreis der Bildfolgenauswertung verursacht, kann der Benutzer bei der Bearbeitung eines Bildes der Folge sich bestimmte Ergebnisse für das vorherige Bild der Folge ansehen. Der Benutzer kann damit während der Protokollinteraktion auf Ergebnisse eines zeitlich vorgelagerten Schrittes zurückgreifen.

Abschließend bleibt festzustellen, daß besonders für Bildfolgenauswertesysteme das Konzept der Protokollinteraktion zur Aktivierung von Testausgaben ein wesentliches Hilfsmittel für die Überwachung eines Systems darstellt.

4.4 Bewertung der Benutzerschnittstellen

In Kapitel 4.1 wurden mehrere Systeme aus dem Bereich der Bildverarbeitung und -auswertung vorgestellt. Die Benutzerschnittstellen dieser Systeme orientieren sich überwiegend an Kommandos. Zur Bewertung der Benutzerschnittstellen wurde ein neues Klassifikationsschema entwickelt, das zwischen Interaktionsformen und Interaktionskonzepten unterscheidet. Als *Interaktionsform* werden die zur Durchführung einer Interaktionshandlung verwendete Technik und ihre Form der graphischen Darstellung bezeichnet. Ein *Interaktionskonzept* hingegen beschreibt die Bedeutung einer Interaktionshandlung, die der Benutzer zur Durchsetzung seiner Ziele vollführt.

Die Benutzerschnittstellen der untersuchten Systeme gestatten die Ableitung von vier Interaktionsformen. Die Form der *einfachen Kommandos* wird häufig verwendet und stellt sich aus der Sicht des Benutzers als textuelle Eingabe einer linearen Befehlsfolge dar. Eine Erweiterung tritt in Form von *Unix-Kommandos* auf, die einen interaktiven Aufbau einer Fließbandverarbeitung mithilfe von Filtern und Datenleitungen gestatten. Eine andere Form äußert sich durch *Menüs*, die alle verfügbaren Eingabealternativen anführen und dem Benutzer die Auswahl einer Option ermöglichen. Die vierte, häufig auftretende Interaktionsform besteht aus einem festen *Frage-Antwort-Schema.*

Die Benutzerschnittstellen der untersuchten Systeme lassen vier Interaktionskonzepte erkennen, die dem Benutzer die Durchsetzung seiner Interaktionsziele gestatten. Die *lokale* und *globale Ablaufsteuerung* dient der benutzerkontrollierten Beeinflussung des Systems. Die *Datenflußsteuerung* regelt die Angabe der Datenquellen und -senken und den Datenaustausch zwischen den Systemkomponenten. Die *Parameterinteraktion* verwendet der Benutzer u.a. zur lokalen (d.h. innerhalb einer Komponente) Steuerung des Ablaufs. Zur Überwachung des Verarbeitungsablaufs wird eine *Protokollinteraktion* angeboten. Zur Realisierung dieser Konzepte werden die beschriebenen Interaktionsformen verwendet.

Diese Formen und Konzepte zur Interaktion werden von keinem der vorgestellten Systeme vollständig angeboten. Als Interaktionsmedium dienen überwiegend einfache alphanumerische Terminale, eine Verwendung von graphikfähigen Sichtgeräten findet sich kaum.

Das in Kapitel 3 vorgeschlagene Benutzermodell wird durch die Analyse der untersuchten Systeme untermauert. Die dabei formulierten Benutzer-

ziele werden durch die verwendeten Interaktionskonzepte bestätigt.

Eine Realisierung der in Kapitel 3.3 aufgestellten Konzepte zur Interaktionsgestaltung findet sich in keinem der bekannten Bildverarbeitungssysteme. Lediglich die Zerlegung eines Systems in Komponenten, die Teilaufgaben realisieren, läßt sich wiederfinden.

Als Zusammenfassung ist zu sagen, daß die untersuchten interaktiven Schnittstellen von Systemen zur Bildverarbeitung und -auswertung den in dieser Arbeit vertretenen Ansprüchen nicht genügen. Im folgenden Kapitel wird deshalb eine neue Benutzerschnittstelle für die Bildfolgenauswertung konzipiert.

Kapitel 5

Konzeption einer Benutzerschnittstelle

In diesem Kapitel wird die Konzeption einer Benutzerschnittstelle vorgestellt, die dem Benutzer eine interaktive Handhabung eines Bildfolgenauswertesystems ermöglichen soll. Diese Schnittstelle unterscheidet sich von bekannten Dialogschnittstellen für Bildverarbeitungssysteme u.a. dadurch, daß sie aus einem vorher entwickelten Benutzermodell entstanden ist und dem Benutzer eine objektorientierte Interaktion ermöglicht. Das in Kapitel 3 dargelegte Benutzermodell und die daraus abgeleiteten Gestaltungskonzepte bilden die Grundlage für diese Schnittstelle.

Die wichtigsten Konzepte zur Gestaltung dieser Schnittstelle sind eine graphische Darstellung des Bildfolgenauswertesystems und eine Interaktion, die auf dieser graphischen Darstellung basiert. Diese Interaktion ist objektorientiert und ermöglicht dem Benutzer eine direkte Manipulation des graphisch repräsentierten Systems. Weiterhin werden bei der Konzeption dieser Schnittstelle die in Kapitel 4 erarbeiteten Erkenntnisse über die Interaktion in bekannten Systemen zur Bildverarbeitung und -auswertung mit berücksichtigt.

Nachfolgend werden zuerst die graphische Darstellung des Bildfolgenauswertesystems und seiner Systemkomponenten erörtert. Danach werden die Interaktionsformen beschrieben, auf denen die Kommunikation des Benutzers mit dieser Schnittstelle basiert. Es folgt eine Aufstellung über die Information, die der Benutzer als Beschreibung der Systemkomponenten erhalten kann. Weiterhin werden die Interaktionskonzepte und ihre Realisierung zur Steuerung des Systems beschrieben. Abschließend werden eine Interaktionshistorie und eine automatische Interaktionsausführung eingeführt.

5.1 Interaktionsobjekte

Bei der Konstruktion von Experimentalsystemen für die Bildfolgenauswertung ist die Struktur derartiger Systeme ein Gegenstand der Forschung. Die Zusammensetzung eines Systems kann sich in Abhängigkeit von den durchzuführenden Experimenten ändern. Aus diesen und den in Kapitel 3.3 dargelegten Gründen ist es sinnvoll, dem Benutzer eine flexible interaktive Gestaltung des Experimentalsystems anzubieten.

In Anlehnung an das in Kapitel 3 formulierte Benutzermodell besteht das Bildfolgenauswertesystem aus einer Menge von Interaktionskomponenten. Die Interaktion des Benutzers mit dem System läßt sich in die Interaktion mit den Komponenten aufteilen. Ein Ziel der Benutzerinteraktion liegt in der Manipulation des Systems. Im Rahmen der in diesem Kapitel konzipierten Benutzerschnittstelle findet eine direkte Manipulation der graphisch dargestellten Komponenten durch den Benutzer statt.

Die Interaktionskomponenten werden deshalb als Objekte bezeichnet, die der Benutzer gemäß seiner Wünsche beeinflußt. Ein *Objekt* wird als eigenständige Funktionseinheit verstanden, in der Datenstrukturen und die entsprechenden Operationen zusammengefaßt sind. Ein Objekt besitzt eine *Schnittstelle zur Interaktion.* Nur über diese Schnittstelle kann der Benutzer mit einem Objekt kommunizieren. Sie definiert weiterhin die Informationen, zu denen der Benutzer interaktiv Zugang bekommt.

Es werden vier verschiedene Objektausprägungen eingeführt, die sich durch ihre Funktion im System voneinander unterscheiden. Ein *Verarbeitungsobjekt* zeichnet sich dadurch aus, daß es Datenobjekte verarbeitet und somit direkt zum Fortgang der Verarbeitung im System beiträgt. Die in Kapitel 3.1 eingeführten Programmkomponenten eines Systems sowie die verwendeten peripheren Geräte und Spezialprozessoren werden unter dem Begriff Verarbeitungsobjekt zusammengefaßt. Ein Verarbeitungsobjekt kann beliebig viele verschiedene Datenobjekte konsumieren und wieder neue produzieren. Es wird als *Datenquelle* bezeichnet, wenn es nur Datenobjekte produziert und als *Datensenke*, wenn es nur Datenobjekte empfängt.

Ein *Datenobjekt* repräsentiert Datenstrukturen, die zwischen Verarbeitungsobjekten ausgetauscht werden. Ein Datenobjekt führt keine Verarbeitung von anderen Datenobjekten durch.

Ein *Leitungsobjekt* dient der Kommunikation zwischen Verarbeitungsobjekten. Die Verarbeitungsobjekte können Nachrichten, die als Daten-

objekte verstanden werden, nur über entsprechende *Datenleitungen* oder *Datenkanäle* übermitteln. Diese Datenleitungen oder -kanäle werden durch Leitungsobjekte realisiert. Ein Leitungsobjekt gestattet die Verbindung von zwei Verarbeitungsobjekten. Diese Verbindung ist an eine spezielle Datenobjektausprägung (Datentyp) gebunden und ermöglicht eine unidirektionale Übertragung von derartigen Datenobjekten. Die Verbindung eines Verarbeitungsobjektes mit einem anderen Verarbeitungsobjekt ist deshalb nur dann möglich, wenn das sendende Verarbeitungsobjekt denselben Typ von Datenobjekt abgibt, der vom empfangenden Verarbeitungsobjekt erwartet wird. Ein Leitungsobjekt kann eine speichernde Funktion besitzen und als Warteschlange Datenobjekte aufnehmen. Dadurch ist eine Entkoppelung von Verarbeitungsobjekten möglich.

Ein *Datenhaltungsobjekt* dient der Speicherung von Datenobjekten. Es gestattet dem Benutzer, sich einen überschaubaren Ausschnitt von den in der Datenhaltung gespeicherten Datenobjekten zu schaffen. Ein Datenhaltungsobjekt kann entweder als Datenquelle oder als Datensenke fungieren.

Die bisher eingeführten vier Objekte werden als *elementare Objekte* bezeichnet. Ihre Struktur kann interaktiv in keine weiteren Objekte zerlegt werden. Sie ist während des Systemablaufs fest vorgegeben.

Zur interaktiven Verwaltung von Objekten werden dem Benutzer Gruppenobjekte angeboten. Ein *Gruppenobjekt* erlaubt das Zusammenfassen von Interaktionsobjekten. Es ist im Rahmen der Benutzerschnittstelle ein *virtuelles* Objekt, da es in dem zugehörigen Anwendungssystem des Bildfolgenauswertesystems keine Entsprechung findet. Mithilfe von Gruppenobjekten ist der Benutzer in der Lage, sein Bildfolgenauswertesystem interaktiv zu strukturieren und sich ihm angemessene Abstraktionsebenen zur Interaktion zu schaffen. Die genaue Funktion der Gruppenobjekte wird im Rahmen der nachfolgenden Abschnitte weiter verdeutlicht.

Damit ein Objekt über eine Leitung Datenobjekte senden bzw. empfangen kann, muß es einen entsprechenden *Leitungseingang* bzw. *Leitungsausgang* besitzen. Ein Objekt mit mindestens einem Leitungsausgang bzw. -eingang soll im weiteren auch als *Sendeobjekt* bzw. *Empfangsobjekt* bezeichnet werden.

Mithilfe der vorgestellten Interaktionsobjekte ist der Benutzer in der Lage, ein Bildfolgenauswertesystem interaktiv zu gestalten und die Interaktion abzuwickeln. Es setzt sich aus Verarbeitungs-, Datenhaltungs- und Gruppenobjekten zusammen, die über Leitungsobjekte verbunden sind und die zu bearbeitenden Datenobjekte untereinander übermitteln.

5.2 Objektrepräsentation

Ein aus Interaktionsobjekten gestaltetes System soll eine graphische Systemdarstellung aufweisen. Jedes im System enthaltene oder am Ablauf beteiligte Objekt besitzt eine graphische Repräsentation in der Systemdarstellung. Die Form der Darstellung wird durch die Objekte vorgegeben. Es ist aber auch denkbar, daß der Benutzer für Objekte eine ihm angemessener erscheinende Darstellung als die vorgegebene wählen kann.

Ein Objekt kann mehrere Arten der Darstellung besitzen. Jedes Objekt sollte in Form eines kompakten graphischen Symbols, auch *Piktogramm* genannt, dargestellt werden. Die Form der Piktogramme ist so zu wählen, daß sie den Benutzer durch ihr Aussehen an die Funktion des repräsentierten Objektes erinnern. Die Piktogramme können weiterhin vom Benutzer gewählte Namen für die Objekte enthalten. Bei den Verarbeitungsobjekten erscheint eine Anzeige über den Fortschritt ihrer Verarbeitung sowie eine Kennzeichnung ihrer Dateneingänge und -ausgänge sinnvoll. Die Leitungsobjekte werden als gerichtete Linien dargestellt, die ein Sende- und Empfangsobjekt verbinden.

Neben der Darstellung durch ein Piktogramm können Objekte auch detaillierter dargestellt werden. Dazu wird ein rechteckiges Fenster auf dem Bildschirm eröffnet. Dieses Fenster dient zur genauen Beschreibung eines Objektes und als Medium für die Kommunikation zwischen diesem Objekt und dem Benutzer.

Zur übersichtlichen Gestaltung der Systemdarstellung können Objekte mithilfe von Gruppenobjekten zusammengefaßt werden. Durch diese Gruppierung ist eine übersichtlichere graphische Wiedergabe der Systemstruktur möglich. Ein Gruppenobjekt ersetzt in der Systemdarstellung die Objekte, die es repräsentiert. Gruppenobjekte können ebenfalls wieder über Gruppenobjekte zusammengefaßt werden. Umgekehrt ist eine detailliertere Darstellung der Gruppenobjekte möglich, indem ihre Darstellung durch die durch sie zusammengefaßten Objekte ersetzt wird.

5.3 Form der Interaktion

Die Interaktion zwischen dem Benutzer und dem Bildfolgenauswertesystem ist objektorientiert, d.h. sie findet immer auf der Basis der dargestellten Objekte statt. Damit der Benutzer Objekte manipulieren und mit

ihnen kommunizieren kann, muß er diese kennzeichnen. Die Auswahl der Objekte erfolgt durch ein Zeigeinstrument. Dieses gestattet ein indirektes Markieren von Objekten über eine graphisch dargestellte Marke, deren Position auf einem Bildschirm angezeigt wird. Durch die Bewegung des Zeigeinstrumentes kann der Benutzer die Markenposition verändern.

Neben dem Zeigeinstrument stehen meistens noch ein oder mehrere Signalgeber sowie eine Tastatur zur Eingabe alphanumerischer Information zur Verfügung. Der Benutzer kann die Signale über einen Tastendruck auslösen. Diese Signale werden beispielsweise zur Bestätigung einer Objektauswahl oder als Aufforderung zur Durchführung einer Operation verwendet. Die Signalgeber werden als eine Einheit mit dem Zeigeinstrument angesehen. Im KOGS-Labor stehen zwei derartige Zeigeinstrumente zur Verfügung. Es existiert eine Rollkugel mit zwei Knöpfen und eine Maus der Fa. Apple. Die Rollkugel hat den Vorteil, daß ihre Position auf der Arbeitsfläche des Benutzers zur Auswahl von Objekten nicht verändert werden muß.

Für eine Maus dagegen benötigt der Benutzer eine freie Fläche, da durch die Bewegung der Maus die aktuelle Position auf dem Bildschirm verändert wird. Diese Maus bildet zusammen mit dem Signalgeber in Form einer Taste eine Funktionseinheit. Die Maus besitzt im Vergleich zu einer Rollkugel den Vorteil, daß der Benutzer sie mit einer Hand bewegen und gleichzeitig die Taste betätigen kann. Sie ist deshalb aufgrund ihrer leichten Handhabung gegenüber einer Rollkugel vorzuziehen.

Die Frage, wieviele verschiedene Tasten als Signalgeber zu einer Maus gehören sollen, ist umstritten. Die Smalltalk-Umgebung (siehe Kapitel 2.3) und die Cedar-Umgebung (siehe Kapitel 2.4) verwenden eine Maus mit drei Knöpfen. Bei dem Entwurf des Xerox Star Systems (siehe Kapitel 2.5) wurden ausführliche Untersuchungen über eine optimale Anzahl von Knöpfen durchgeführt [*Bewley et al. 83*]. Nach mehreren Versuchen hat man sich für die Konzeption einer Schnittstelle entschieden, die auf einer Maus mit zwei Knöpfen basiert. Die Systeme der Fa. Apple (Lisa, MacIntosh) [*Williams 83*, *Williams 84*] verwenden jedoch eine Maus mit nur einem Knopf. Eine derartige Maus steht auch für die in diesem Kapitel konzipierte Benutzerschnittstelle zur Verfügung.

Die Aktionen des Benutzers haben im Rahmen der Benutzerschnittstelle nach einem vorgegebenen Schema zu erfolgen. Zuerst werden ein oder mehrere Objekte selektiert und danach die darauf anzuwendenden Operationen. Diese Vorgehensweise entspricht dem Konzept der direkten Manipulation,

das die Objekte in den Vordergrund stellt. Es ist anzunehmen, daß der Benutzer bei der Auswahl eines Objektes bewußter und damit sorgfältiger vorgehen wird als bei der Aktivierung eines Kommandos. Diese Form der Interaktion trägt somit auch zur Reduzierung der Komplexität und Anzahl der Kommandos bei.

Ein Vorteil der objektorientierten Interaktion ist das Vermeiden von Modi. Der Benutzer kann während der Interaktion zwischen den Objekten wechseln, indem er einfach ein anderes zu bearbeitendes Objekt und eine auf dieses anwendbare Operation auswählt. Es wird keine abschließende Bestätigung des Benutzers benötigt, da die Auswahl der Operation immer die letzte Aktion einer Interaktionshandlung ist.

Unter der Auswahl eines Objektes wird in diesem Zusammenhang der folgende Schritt verstanden. Der Benutzer markiert ein auf dem Bildschirm dargestelltes Objekt. Sobald sich die durch die Maus gesteuerte Marke innerhalb eines Objektes befindet, gilt dieses Objekt als ausgewählt. Alle Eingaben des Benutzers und die dadurch ausgelösten Operationen werden nachfolgenden unter Bezug auf dieses Objekt interpretiert. Der Benutzer kann die Interaktion mit einem Objekt jederzeit unterbrechen, indem er die Marke aus dem Objektbereich entfernt.

Der Benutzer hat die Möglichkeit, mehrere Objekte nicht nur hintereinander, sondern auch gleichzeitig auszuwählen. Zur Realisierung eines derartigen Konzeptes können die Gruppenobjekte verwendet werden. Wählt der Benutzer ein Gruppenobjekt aus, so bedeutet dies die Auswahl aller durch das Gruppenobjekt repräsentierten Objekte. Die nach der Auswahl auf ein Gruppenobjekt anwendbaren Operationen und ihre Auswirkungen auf die repräsentierten Objekte sind von deren Funktion im Gesamtsystem abhängig.

Sämtliche Aktionen des Benutzers werden von der Benutzerschnittstelle in Form von graphischen Rückmeldungen bestätigt. Diese Rückmeldungen äußern sich beispielsweise bei der Manipulation von Objekten durch die ständige Aktualisierung der Systemdarstellung oder durch die Eröffnung eines Fensters mit detaillierteren Interaktionsoptionen.

Ein wichtiges Ziel bei der Konzeption dieser Schnittstelle ist die direkte Manipulation. Diese umfaßt eine Reduzierung von Kommandos zur Bearbeitung von Objekten. Stattdessen kann der Benutzer die graphisch dargestellten Objekte direkt bearbeiten. Diese Art der Bearbeitung ist natürlich nur dann sinnvoll, wenn eine angemessene Form der graphischen Manipulation denkbar ist.

Das Schalten von Datenleitungen, um die Kommunikation zwischen Objekten zu ermöglichen, erfolgt durch die Verbindung dieser Objekte über eine Linie. Die am Systemablauf beteiligten Objekte kann der Benutzer nach Belieben plazieren. Das Zusammenfassen von Objekten zu Gruppenobjekten geschieht durch die Übertragung eines Objekt-Piktogramms in ein Piktogramm eines Gruppenobjektes. Umgekehrt kann ein Objekt wieder aus einem Gruppenobjekt entfernt werden. Die Auswahl der zu bearbeitenden Datenobjekte erfolgt durch das Kopieren aus dem Bereich der Datenverwaltung in spezielle als Datenquellen verfügbare Objekte. Die Zuordnung zwischen den Ausgaben der Objekte und den verfügbaren Ausgabegeräten findet über die Kennzeichnung von Fenstern statt.

Bei Operationen, die durch eine manuelle Manipulation des Benutzers nicht realisierbar sind, werden andere Formen der Aktivierung gewählt. Die Ausführung von Operationen kann beispielsweise über graphisch dargestellte Operationsfelder (als virtueller Tastensatz) oder über eine speziell markierte Taste auf der Tastatur des Benutzerterminals erfolgen. Stehen mehrere Operationen zur Verfügung, so werden sie mithilfe eines Menüfensters angezeigt. Der Benutzer kann aus diesem Menü eine Operation auswählen und damit die gewünschte Operation auslösen.

5.4 Objektbeschreibung

Die ausführliche Beschreibung eines Objektes kann sich der Benutzer jederzeit interaktiv darstellen lassen. Diese Darstellung findet in einem Bildschirmfenster statt, das das Objekt repräsentiert. In Abhängigkeit von dem Objekttyp sind unterschiedliche Beschreibungen verfügbar.

Bei allen Objekten existiert eine *Statusbeschreibung.* Für die Verarbeitungsobjekte können die Parameterwerte dieser Objekte erfragt werden. Weiterhin steht für ausgewählte interne Datenstrukturen eine Darstellung zur Verfügung, die mit unterschiedlichem Detaillierungsgrad angefordert werden kann. Eine kontinuierliche Darstellung gibt über den Fortschritt der Verarbeitung Auskunft. Falls vorhanden, werden die Eingangs- und Ausgangsdaten des Objektes angezeigt. Bei Objekten, die als Datenquelle bzw. Datensenke fungieren, wird die Anzahl der dort gespeicherten Datenobjekte dargestellt. Für die Datenleitungen findet eine Darstellung des Datenflusses statt.

Die Statusbeschreibung kann nicht nur auf Anforderung durch den Be-

nutzer, sondern auch als ständige Anzeige dargestellt werden, die immer aktualisiert wird. Dafür hat der Benutzer die Möglichkeit, nur einen Teil der Statusbeschreibung auszuwählen. Die Auswahl der anzuzeigenden Information könnte auch vom System automatisch getroffen werden.

Neben der Statusbeschreibung ist eine *Strukturbeschreibung* der Objekte verfügbar. Bei den Gruppenobjekten besteht diese aus der Darstellung der Objekte und ihrer Verbindungen untereinander, die durch die Gruppenobjekte zusammengefaßt sind. Die strukturelle Beschreibung der Datenobjekte richtet sich nach deren Typ. Als Beispiel seien für Rasterbilder oder Bildausschnitte deren Umfang, das Format der Bildelemente, eine bildliche Darstellung der verkleinerten Wiedergabe eines repräsentativen Bildes und eine Historie über ihr Entstehen erwähnt.

Die dritte Form der Objektbeschreibung bietet eine *Erklärung* über die Funktionsweise eines Objektes und eine *Hilfestellung* für den Umgang mit diesem im Rahmen des Systems. Bei Verarbeitungsobjekten erscheint eine Erklärung der Objektparameter sinnvoll. Diese Erklärung kann durch eine Vorführung mit Beispieldaten unterstützt werden. Der Benutzer kann sich dann an den graphisch dargestellten Ergebnissen ein Bild über die Funktionsweise des Objektes machen. Die Datenobjekte können dem Benutzer über ihren Inhalt und ihre Bedeutung bei der Bearbeitung eine Auskunft geben. Weiterhin bieten sie dem Benutzer eine Operation an, die eine graphische Darstellung der durch das Objekt repräsentierten Daten erzeugt (s.o.).

5.5 Steuerung durch den Benutzer

Eine wichtige Funktion der Benutzerschnittstelle ist die Realisierung einer interaktiven Steuerung des Systems durch den Benutzer. Diese Steuerung umfaßt den Systemablauf, den Datenfluß, die Besetzung der Objektparameter und den Umfang der Protokollierung. Diese Interaktionskonzepte und ihre Verwendung in Bildverarbeitungssystemen wurden bereits in Kapitel 4.3 ausführlich behandelt. Nachfolgend wird die Realisierung dieser Konzepte im Rahmen dieser Schnittstelle dargestellt.

5.5.1 Systemablauf

Der Benutzer soll die Möglichkeit haben, eine globale Ablaufsteuerung

auszuüben. Dafür muß er den Verarbeitungsvorgang der am System beteiligten Objekte steuern. Diese Steuerung umfaßt die Initialisierung von Objekten sowie den Start der Verarbeitung. Die Verarbeitung der Objekte soll jederzeit unterbrechbar und wieder fortsetzbar sein. Dieses soll für einzelne Objekte, alle Objekte eines Typs oder alle Objekte des Systems steuerbar sein. Die gleichzeitige Steuerung von mehreren Objekten erfolgt über Gruppenobjekte. Um eine flexible Kontrolle der Interaktionsobjekte zu ermöglichen, erscheint es denkbar, daß ein Objekt gleichzeitig in mehreren Gruppenobjekten enthalten ist.

Bei den Datenquellen und Datensenken ist eine Blockierung denkbar. Diese bedeutet, daß keine Datenobjekte mehr produziert oder konsumiert werden. Für die Datenleitungen ist eine Sperrung bzw. Entsperrung ihrer Übertragungskapazität möglich. Für alle Objekte gilt, daß sie nach der Initialisierung und Freigabe durch den Benutzer eigenständig und parallel zu allen anderen Objekten arbeiten. Diese Verarbeitung wird nur dann unterbrochen, wenn die Objekte auf bestimmte Ereignisse warten oder der Benutzer eine Unterbrechung veranlaßt hat.

5.5.2 Datenfluß

Der Benutzer steuert den Datenfluß zwischen den Verarbeitungsobjekten durch das Schalten von Datenleitungen. Eine Leitung verbindet immer zwei Objekte miteinander. Der Datenausgang des sendenden Objektes wird mit dem Dateneingang des empfangsbereiten Objektes verbunden. Nach der Freigabe der Leitung durch den Benutzer können über diese zwischen den verbundenen Objekten Daten transportiert werden. Für die Übertragung hat die Datenleitung die Funktion eines Vermittlers.

Die Datenleitung ist selber ein Objekt. Durch ihre Vermittlungsfunktion zwischen den Objekten bleibt den Sende- und Empfangsobjekten die Identität ihrer Kommunikationspartner verborgen. Dadurch ist es für den Benutzer jederzeit möglich, eine Datenleitung von einem der beiden Objekte abzukoppeln und an ein anderes Objekt anzuschließen. Diese Änderung der Leitungsverbindung bleibt dem nicht betroffenen Objekt verborgen, da für dieses die Leitung als Schnittstelle dient.

Datenleitungen können auch als Warteschlangen fungieren. Sie erhalten dazu ein Speichervermögen für Datenobjekte. Diese Speicherkapazität kann vom Benutzer interaktiv steuerbar sein, beispielsweise durch die Angabe der maximal zu speichernden Datenobjekte.

Zur Steuerung des Datenflusses gehört weiterhin die Auswahl von Datenquellen und -senken. Die Auswahl der zu bearbeitenden Daten erfolgt durch deren Markierung in der Datenverwaltung und durch das Kopieren in dafür bereitgestellte Datenquellen. Nach Freigabe der Datenquelle stellt diese in ihrer Funktion als Sendeobjekt auf Anforderung den mit ihr verbundenen Empfangsobjekten die gespeicherten Datenobjekte zur Verfügung.

5.5.3 Parameter

Unter den Parametern eines Objektes werden speziell gekennzeichnete Datenstrukturen verstanden. Der Benutzer kann die Parameterwerte jederzeit erfragen. Die Parameterwerte werden einzeln oder komplett als Parameterfeld in einem dem Objekt zugeordneten Fenster dargestellt. Die Darstellung der Werte erfolgt in Abhängigkeit von dem Datentyp der Parameter alphanumerisch oder graphisch. Die Parameterwerte können auch als ein Teil einer ständigen Statusanzeige des Objektes auftreten (siehe auch Kapitel 5.4).

Die Parameter eines Objektes definieren nicht nur eine Statusbeschreibung, sondern auch eine Interaktionsschnittstelle zu dem Benutzer. Der Benutzer kann über die in dieser Schnittstelle zugänglichen Parameter einen durch die Objektstruktur vorgeprägten Einfluß auf dieses ausüben. Dieser Einfluß erfolgt durch eine Besetzung oder Modifikation der Parameterwerte.

Gruppenobjekte können ebenfalls Parameter besitzen. Im Gegensatz zu den elementaren Objekten haben die Gruppenobjekte Merkmale, die sich dynamisch aus den in ihnen enthaltenen Objekten ableiten lassen. Eines dieser Merkmale sind die Parameter des Gruppenobjektes. Die Parameter eines Gruppenobjektes können vom Benutzer sowohl interaktiv als auch durch das System nach einem festen Algorithmus bestimmt werden. Diese Auswahl kann durch die Objekte unterstützt werden, indem sie diejenigen ihrer Parameter kennzeichnen, die sich besonders als Parameter von Gruppenobjekten eignen.

Durch die Zuweisung von Parametern an Gruppenobjekte wird dem Benutzer die Möglichkeit gegeben, sich eine angemessene Auswahl von Parametern aus den durch das Gruppenobjekt beschriebenen Objekten zu definieren. Diese Parameter kann er sich anzeigen lassen oder auch verändern. Verändert der Benutzer einen Parameter eines Gruppenobjektes, so wird der angegebene Wert bei allen Objekten eingetragen, die diesen Parameter besitzen.

Der Benutzer ist somit in der Lage, gleichzeitig Parameter von mehreren Objekten durch einen einzigen entsprechenden Parameter des übergeordneten Gruppenobjektes mit einer Aktion zu verändern. Er wird nicht gezwungen, analoge Operationen nacheinander auf mehrere Objekte anzuwenden.

Für jedes Objekt muß im Rahmen des Systems, soweit es sinnvoll ist, eine Voreinstellung der Parameterwerte existieren. Diese Voreinstellung kann mit der Objektdefinition vorgegeben werden. Der Benutzer darf diese Voreinstellung ändern und hat die Möglichkeit, sich eigene Voreinstellungen zu definieren. Die von dem Benutzer gewählten Voreinstellungen sind an diesen gebunden und müssen für andere Benutzer nicht gelten. Die Voreinstellungen können auch in Abhängigkeit von dem durchzuführenden Experiment gewählt werden.

Diese von dem Benutzer oder dem Experiment abhängigen Voreinstellungen werden durch das Schaffen von benutzerabhängigen bzw. vom Experiment abhängigen Gruppenobjekten realisiert. Mit diesen Gruppenobjekten werden die vom Benutzer gewählten Parametervoreinstellungen assoziiert.

Zu der Definition der Objektparameter gehört auch eine Beschreibung, ob der Benutzer bestimmte Parameterwerte angeben muß und wann er vorgegebene Werte verändern kann. Für den Zeitpunkt der Änderung erscheinen drei Stufen sinnvoll. Es wird Parameter geben, deren Werte zu jedem Zeitpunkt veränderbar sind. Bei anderen Parametern ist es vernünftig, sie nur zu bestimmten, durch das Objekt definierten, Zeitpunkten zu modifizieren, während bei einigen Parametern nur eine einmalige Initialisierung möglich ist.

Zu jedem Parametersatz eines Objektes gehören nicht nur mögliche Voreinstellungen für die Parameterwerte, sondern auch eine semantische Beschreibung der Parameter. Diese Beschreibung enthält Einschränkungen über die Werte, die ein Parameter annehmen kann. Die Benutzerschnittstelle führt eine lexikalische und syntaktische Überprüfung der Benutzereingaben für die Parameterwerte gemäß ihres Datentyps durch. Anhand der semantischen Beschreibung der Parameter werden weitere Einschränkungen für ihren Wertebereich kontrolliert. Diese Kontrolle wird durch eine vom Objekt definierte Operation unterstützt, die die Abhängigkeiten der Parameter untereinander und die semantische Integrität des gesamten Parametersatzes überprüft.

5.5.4 Protokoll

Zur Überwachung der Verarbeitung ist die Protokollinteraktion besonders für Bildfolgenauswertesysteme ein wichtiges Hilfsmittel des Benutzers. Der Benutzer des Systems will in seiner Funktion als Experte die Zwischen- oder Teilergebnisse des Systems kontrollieren und die Verarbeitung nach Bedarf unterbrechen oder gar abbrechen können.

Die Protokollausgaben werden von speziellen Verarbeitungsobjekten erzeugt, die als *Protokollobjekte* bezeichnet werden. Es gibt zwei Arten von Protokollobjekten. Der eine Typ dient der Überwachung von Verarbeitungsobjekten, während der andere Datenleitungen kontrolliert. Der Benutzer kann den Protokollobjekten Bildschirmfenster zuweisen, auf denen ihre Ausgaben dargestellt werden.

Die Protokollobjekte sind von den zu überwachenden Verarbeitungsobjekten oder Datenleitungen abhängig, da sie deren Datenstrukturen interpretieren müssen. Der Benutzer muß die Protokollobjekte bei den zu überwachenden Objekten plazieren und anschließen oder zwischen die Datenleitungen schalten. Nach der Zuordnung eines Fensters und der Freigabe der Protokollobjekte werden sofort Protokollausgaben erzeugt.

Zur Verwirklichung der Benutzerziele erscheint es sinnvoll, den Detaillierungsgrad der Ausgaben staffeln zu können. Der Benutzer kann verschiedene Stufen eines Protokolls anfordern und damit den Umfang der Protokollausgaben interaktiv steuern. Der Benutzer ist somit in der Lage, sich das Protokoll beispielsweise für eine Fehlersuche ganz ausführlich anzeigen zu lassen, während bei einer normalen Überwachung nur wenige Ausgaben benötigt werden.

5.6 Interaktionshistorie

Bei vielen interaktiven Systemen gehört es zum Standard, daß eine Historie über die Interaktionshandlungen des Benutzers geführt wird. Diese Historie stellt sich dem Benutzer als eine chronologisch geordnete Folge der Benutzeraktionen und Systemreaktionen dar.

Die meisten Systeme, die eine Interaktionshistorie führen, sind an einer Kommandosprache orientiert. Als Interaktionshandlung wird dort die Eingabe eines Kommandos verstanden. Die Benutzerschnittstelle speichert das eingegebene Kommando in Form einer Zeichenkette als ein Element der

Historie ab. Als Beispiel für derartige Systeme seien Interlisp [*Teitelman & Masinter 81*], Unix [*Kernighan & Mashey 81*] und Cedar [*Teitelman 85*] genannt (siehe auch Kapitel 2).

Die in diesem Kapitel konzipierte Benutzerschnittstelle bietet dem Benutzer im Gegensatz zu den oben erwähnten Systemen zur Interaktion die direkte Manipulation von graphisch dargestellten Objekten an. Die Eingabe von textuell dargestellter Information durch den Benutzer umfaßt nur noch einen geringen Teil der Interaktionshandlung. Damit scheidet eine einfache Führung der Interaktionshistorie als Folge von Zeichenketten aus, die sich aus den eingegebenen Befehlen des Benutzers zusammensetzen.

Eine *Interaktionshandlung* wird im Rahmen dieses Systems als eine Operation des Benutzers verstanden. Diese Operation ist in dem Sinne elementar, daß sie in keine weiteren Handlungen unterteilt werden kann. Die Interaktionshandlung äußert sich immer als Manipulation eines graphisch dargestellten Objektes. Es gibt Handlungen wie beispielweise die Parameterinteraktion, die keine Auswirkungen auf andere Objekte haben. Andere Handlungen dagegen (z.B. die Datenflußsteuerung) stellen sich zwar als Manipulation eines Objektes (z.B. der Datenleitung) dar, aber dadurch werden auch andere Objekte berührt (z.B. Sender und Empfänger).

In der Interaktionshistorie werden nur abgeschlossene Interaktionshandlungen geführt. Eine Interaktionshandlung mit einem Objekt gilt als *abgeschlossen*, wenn sie die von der Benutzerschnittstelle definierten lexikalischen, syntaktischen und semantischen Eingabenormen erfüllt und den Objektzustand — und damit auch den Systemzustand — verändert hat. In der graphischen Darstellung des Systems wird eine abgeschlossene Interaktionshandlung durch eine dauerhafte Veränderung des Systembildes deutlich. Eine Interaktionshandlung wird bis zu ihrem vollständigen Abschluß durch den Benutzer als *offen* bezeichnet. Eine offene Interaktionshandlung hat noch keinen Objektzustand verändert und kann damit leicht rückgängig gemacht werden. Als Beispiel für Interaktionshandlungen seien das Legen einer Datenleitung, das Setzen eines Parameters, die Auswahl eines Objektes oder das Ändern eines Gruppenobjektes genannt.

Die Benutzerschnittstelle ordnet die Interaktionshistorie nach zwei Kriterien. Das erste Kriterium ist die chronologische Ordnung (in Echtzeit) aller Interaktionshandlungen. Alle Aktionen des Benutzers werden gemäß ihrer chronologischen Reihenfolge in der *Systemhistorie* gespeichert. Dabei spielt es keine Rolle, welches Objekt von diesen Aktionen betroffen wurde. Das zweite Kriterium für die Führung der Historie sind die an der Aktion

beteiligten Objekte. Alle Handlungen, die sich auf ein Objekt beziehen, werden einer objektspezifischen Historie zugeordnet. Dabei ist es gleichgültig, ob diese Aktionen hintereinander und ohne Unterbrechung oder zu verschiedenen Zeitpunkten ausgeführt wurden. Die *Objekthistorie* enthält alle Interaktionshandlungen, die ein Objekt betreffen, in ihrer chronologischen Reihenfolge.

Das Führen einer Interaktionshistorie erfordert für die Konstruktion eines entsprechenden interaktiven Systems einen nicht unerheblichen Aufwand. Dieser Aufwand ist jedoch damit zu rechtfertigen, daß sich der für den Benutzer angebotene Komfort bei der Dialoggestaltung wesentlich erhöht. Mit der Führung einer Interaktionshistorie werden Operationen angeboten, die dem Benutzer eine Inspektion und erneute Ausführung der in der Historie enthaltenen Handlungen erlauben.

Als ein Teil der Benutzerschnittstelle existiert ein dem Benutzer zugängliches Objekt, welches die Interaktionshistorie mit den zugehörigen Operationen realisiert. Diese Operationen sind somit auch Interaktionshandlungen und werden von diesem Objekt ebenfalls protokolliert.

Wie schon oben erwähnt, stellen sich die Interaktionshandlungen des Benutzers überwiegend als Manipulationen der graphischen Systemdarstellung dar. Die Historie wird deshalb als Bildfolge existieren, die die Auswirkung der Interaktionshandlungen in der graphischen Darstellung wiedergibt. Der Benutzer kann sich die Historie eines Objektes oder die des gesamten Systems zeigen lassen.

Dabei kann der Benutzer die Historie Handlung für Handlung inspizieren oder sich Teile vorspielen lassen. Während dieser Interaktionsoperation werden nur die Auswirkungen auf die Systemdarstellung angezeigt, die damit verbundenen Aktionen aber nicht tatsächlich ausgeführt. Der Benutzer kann eine Kopie der Historie editieren, d.h. er kann darin enthaltene Handlungen verändern, löschen oder neue einfügen.

Teile der möglicherweise veränderten Kopie der Historie können dann ausgeführt werden. Bei dieser Ausführung werden nicht nur die Auswirkungen auf die Systemdarstellung angezeigt, sondern das der Historie zugeordnete Objekt führt stellvertretend für den Benutzer die gekennzeichneten Interaktionshandlungen aus. Der Benutzer kann Teile oder auch die gesamte Historie sichern und zu einem beliebigen Zeitpunkt wieder restaurieren.

Damit wird dem Benutzer die Möglichkeit gegeben, seine Arbeit mit dem Bildfolgenauswertesystem zu unterbrechen und zu einem beliebigen anderen Zeitpunkt wieder unter denselben Bedingungen fortzusetzen. Durch

das Restaurieren und Ausführen einer gesicherten Historie kann das System erneut den bei der Sicherung der Historie vorhandenen Zustand erreichen.

5.7 Automatische Interaktionsausführung

Die in dem vorherigen Abschnitt vorgestellte Historie und die damit verbundenen Möglichkeiten können als eine spezielle Anwendung der nachfolgend vorgestellten automatischen Interaktionsausführung angesehen werden.

Unter einer automatischen Interaktionsausführung wird die Möglichkeit des Benutzers verstanden, fertige Ablaufpläne zu aktivieren. Ein *Ablaufplan* besteht aus einer Folge von Interaktionshandlungen. Die Ausführung eines solchen Ablaufplans ersetzt die Interaktionshandlungen des Benutzers durch die in dem Ablaufplan enthaltenen Handlungen. Dasselbe Objekt, welches Teile der Historie interpretiert, führt derartige Ablaufpläne stellvertretend für den Benutzer aus.

Der Benutzer kann die Ablaufpläne mit denselben Operationen erstellen, die auch schon für die Historie verfügbar sind. Die Historie ist selber nichts anderes als ein von der Benutzerschnittstelle ständig erweiterter spezieller Ablaufplan. Der Benutzer kann Ablaufpläne in einer Datenbasis speichern und nach Bedarf aktivieren und ausführen lassen.

Die Ablaufpläne geben dem Benutzer die Möglichkeit, sich aufbauend auf den elementaren Interaktionshandlungen eigene komplexe Operationen zu schaffen. Weiterhin können die Ablaufpläne dafür genutzt werden, um beim Systemstart Voreinstellungen für die Konfiguration des Systems zu realisieren. Ein Ablaufplan könnte die Auswahl und Anordnung der Objekte sowie deren Vernetzung über Datenleitungen für eine Standardnutzung des Bildfolgenauswertesystems enthalten.

Die automatische Interaktionsausführung kann erweitert werden, um eine flexible Überwachung und Steuerung des Systems in Abwesenheit des Benutzers zu ermöglichen. Eine Überwachung von Objekten kann durch spezielle Protokollobjekte erfolgen. Sie können ständig den Zustand eines Objektes analysieren und bei bestimmten vorgegebenen Situationen selber aktiv werden und stellvertretend für den Benutzer Interaktionshandlungen durchführen. Diese Objekte sollen auch als *Dämonen* bezeichnet werden, da sie ein Objekt "belauern" und erst bei bestimmten Ereignissen in das Geschehen eingreifen.

Durch die Bereitstellung von Dämonen ist der Benutzer in der Lage, eine automatische Überprüfung der Integrität von Zwischenergebnissen zu realisieren. Er kann auch eine Behandlung von auf einer niederen Abstraktionsebene ausgelösten Ausnahmesituationen erreichen, indem er auf einer höheren Abstraktionsebene die Dämonen verwendet, um ein Schema zur Reaktion auf derartige Ereignisse vorzugeben.

5.8 Zusammenfassung

In diesem Kapitel wurde die Konzeption einer Benutzerschnittstelle für Bildfolgenauswertesysteme vorgestellt. Diese Konzeption ist aus dem in Kapitel 3 eingeführten und in Kapitel 4 erweiterten Benutzermodell entstanden. Diese konzipierte Schnittstelle soll abschließend noch einmal an den in Kapitel 3 formulierten Anforderungen und an den in Kapitel 4 erarbeiteten Erkenntnissen gemessen werden.

Die in Kapitel 3.1 geforderte Zerlegung eines Systems in seine *Interaktionskomponenten* hat dazu geführt, daß eine einheitliche Sicht eines Bildfolgenauswertesystems geschaffen wurde. Diese basiert auf dem *Modell der kommunizierenden Objekte*, die im Rahmen eines Verarbeitungsnetzes über Datenleitungen verbunden sind. Darin treten die Interaktionskomponenten als *Verarbeitungsobjekte* und die zu bearbeitenden Daten als *Datenobjekte* auf. Die geforderten *virtuellen Interaktionskomponenten* haben in den *Gruppenobjekten* ihre Entsprechung gefunden. Sie dienen zur Strukturierung des Systems, zur Bindung von benutzerspezifischen Merkmalen an Objekte bzw. Objektmengen und zur gleichzeitigen Interaktion des Benutzers mit mehreren Objekten.

Die *graphische Darstellung des Systems* besteht aus der Darstellung der Objekte, die als Systemkomponenten das Gesamtsystem bilden, und erfolgt mithilfe eines *Farbrastergraphik-Gerätes*. Die *objektorientierte Interaktion* stellt die am Kommunikationsvorgang beteiligten Objekte in den Vordergrund. Sie wird durch die Verwendung von Fenstertechniken zur Darstellung und einer *Maus* als graphisches Eingabegerät unterstützt.

Das Konzept der *direkten Manipulation* gestattet dem Benutzer zur Systemmanipulation physische Aktionen, die sich in der Bewegung und Veränderung von graphisch dargestellten Objekten äußern. Diese direkte Manipulation wird durch eine weitgehend *modusvermeidende Interaktion* unterstützt, indem die Benutzeraktionen die Form Objektauswahl — Ope-

rationsauswahl haben und die mit dem Handlungskontext assoziierten Zustände an die beteiligten Objekte gebunden werden.

Die verfügbaren Operationen auf Objekten wurden entscheidend durch die in Kapitel 3.2 formulierten Interaktionsziele der Benutzer und die in Kapitel 4.3 dargelegten Interaktionskonzepte in Bildverarbeitungssystemen geprägt.

Alle Objekte bieten dem Benutzer eine *Erklärung* ihrer Funktionsweise und eine *Hilfestellung* für ihre Handhabung an. Der *Systemzustand* setzt sich aus der Zustandsbeschreibung der einzelnen Objekte zusammen. Jedes Objekt kann eine *Statusbeschreibung* erzeugen.

Es wurden vier Konzepte zur Steuerung des Systems verwendet. Der *Systemablauf* wird durch die Aktivierung bzw. Deaktivierung von Objekten kontrolliert. Die *Datenflußsteuerung* erfolgt durch die Schaffung von Datenquellen und Datensenken sowie über Datenleitungen zur Verbindung von Objekten. Alle Objekte können *Parameter* und Voreinstellungen für deren Werte besitzen, die der Benutzer verändern darf. Die flexible Überwachung des Systems über *Protokollausgaben* wird durch spezielle Protokollobjekte möglich, die der Benutzer zur Erzeugung der Protokolle interaktiv in das Verarbeitungsnetz des Systems einbinden kann.

Die Interaktion des Benutzers mit dem System wird in *Interaktionshandlungen* unterteilt. Diese werden sowohl in einer *Systemhistorie* als auch in entsprechenden *Objekthistorien* protokolliert. Zur Inspektion, Veränderung und erneuten Ausführung der (Teil-) Historien werden Operationen angeboten.

Die durch die Erfassung der Interaktionshistorie bereitgestellten Daten bilden eine Grundlage für eine *automatische Interaktionsausführung*, die dem Benutzer die Aktivierung von Ablaufplänen erlaubt. Zur automatischen Überwachung und Fehlerbehandlung werden *Objektdämonen* eingeführt.

Mit den in diesem Kapitel konzipierten Gruppen-, Protokollobjekten und Dämonen in Verbindung mit der Interaktionshistorie und der automatischen Interaktionsausführung wurde ein erster Schritt unternommen, dem Benutzer für den Bereich der Bildfolgenauswertung Hilfsmittel bereitzustellen, damit er sich zur Interaktion verschiedene ihm angemessene Abstraktionsebenen schaffen kann.

Kapitel 6

Realisierung eines Dialogsystems

In diesem Kapitel wird die Realisierung eines Dialogsystems zur Bildfolgenauswertung gemäß der in Abbildung 1.2 skizzierten Systemstruktur vorgestellt. Dieses Dialogsystem realisiert die im vorherigen Kapitel konzipierte Benutzerschnittstelle und definiert gleichzeitig eine Anwendungsschnittstelle. Die Prinzipien für den Entwurf von guten interaktiven Systemen sind zur Zeit noch weitgehend unbekannt. Es gibt jedoch Ansätze, das Prinzip der methodischen Programmentwicklung (software engineering) auf den Bereich der Dialogsysteme zu übertragen [*Draper & Norman 85*]. Wie bei der Konzeption der Benutzerschnittstelle wird auch die Realisierung des Dialogsystems ausgehend vom Benutzer (top down) vorgestellt.

Neben der in Kapitel 1 beschriebenen Strukturierung von interaktiven Systemen hat sich die Verwendung eines *Sprachmodells* [*Foley & van Dam 82, Foley et al. 84*] zur Beschreibung und Realisierung eines Dialogsystems bewährt [*Green 84, Olsen et al. 85*]. Dabei wird die Schnittstelle zwischen dem Benutzer und dem Dialogsystem in Form von zwei Sprachen dargestellt. Die eine Sprache beschreibt die vom Benutzer an das System gerichteten Aktionen, während die andere die Aktionen des Systems gegenüber dem Benutzer umfaßt. Jede dieser Sprachen kann in vier wesentliche Teile untergliedert und als *Sprachhierarchie* angesehen werden [*Foley & van Dam 82*].

Die *konzeptuelle Ebene* beschreibt das im System vorhandene Benutzerbild (siehe Kapitel 3). Die *semantische Ebene* spezifiziert die Funktionalität der Benutzerschnittstelle unabhängig von der Form der Interaktion. Die *syntaktische Ebene* definiert die Dialogfolge der Ein- und Ausgabeaktionen und zerlegt sie in Worte. Die *lexikalische Ebene* sorgt für die Abbildung der Worte auf die vorhandenen Ein- und Ausgabegeräte.

Der Aufwand zur Erstellung eines graphischen Dialogsystems ist als sehr hoch anzusehen. Für den Entwurf der Benutzerschnittstelle des Xerox

Star 8010 Systems wurden beispielsweise allein 30 Mannjahre aufgewendet (siehe Kapitel 2.5). Es gibt deshalb Bestrebungen, den Vorgang der Erstellung eines Dialogsystems zu erleichtern und zu automatisieren. Dafür werden Verwaltungssysteme für interaktive Schnittstellen angeboten, deren Struktur im folgenden Abschnitt diskutiert wird. Anschließend wird der objektorientierte Systementwurf vorgestellt, der dem in dieser Arbeit entworfenen Dialogsystem für Bildfolgenauswertesysteme zugrunde liegt. Nachfolgend wird aus diesem Systementwurf ein Prototyp für ein derartiges Dialogsystem abgeleitet und dessen Implementation mithilfe der Programmiersprache Ada beschrieben.

6.1 Verwaltungssysteme für interaktive Schnittstellen

Ein *Verwaltungssystem für interaktive Schnittstellen* (*user interface management system*) [*GIIT 83*] besteht aus einer Menge von Werkzeugen, die der (halb-) automatischen Erstellung eines Dialogsystems dienen [*Olsen et al. 84*]. Als Gründe für die Verwendung eines derartigen Verwaltungssystems können angeführt werden [*Hayes et al. 85, Olsen et al. 84*]:

- leichtere Konstruktion von Dialogsystemen;
- höhere Qualität der resultierenden Benutzerschnittstellen;
- Konsistenz der Benutzerschnittstellen über mehrere Anwendungen;
- Möglichkeit zur Erstellung von mehreren Benutzerschnittstellen für eine Anwendung;
- leichte Änderbarkeit von Benutzerschnittstellen;
- explizite Darstellung von Schnittstellenspezifikationen;
- schnelles Erstellen und Ausprüfen von Prototypen.

Der Umfang der benötigten Werkzeuge, die ein vollständiges Verwaltungssystem bilden, läßt sich noch nicht endgültig angeben. Es können jedoch einige Werkzeuge angeführt werden, die für die Realisierung eines Verwaltungssystems als notwendig erscheinen [*Olsen et al. 84*]. Zur Implementation werden die auf einem Rechner üblichen Programmsysteme wie

Betriebssystem, Programmiersprachen und Programmverwaltungssysteme benötigt. Zur Unterstützung der internen Realisierung des Verwaltungssystems sind Werkzeuge zur Analyse der Benutzereingaben und zur Erzeugung und Verwaltung von graphischen Ausgaben erforderlich.

Die Entwicklung von Verwaltungssystemen für interaktive Schnittstellen ist noch Gegenstand der Forschung und kann keinesfalls als abgeschlossen angesehen werden. Aus diesem Grund gibt es bis jetzt keine einheitlichen Prinzipien zur Gestaltung dieser Verwaltungssysteme.

Es existieren jedoch Kriterien für den Entwurf und die Beurteilung der Struktur derartiger Verwaltungssysteme. Diese Kriterien werden im folgenden Abschnitt näher erläutert. Anschließend folgt eine Klassifikation und Beurteilung von existierenden Verwaltungssystemen oder universellen, graphischen Dialogsystemen.

6.1.1 Struktur von Verwaltungssystemen

Die Struktur der Verwaltungssysteme kann entsprechend dem Aufbau von interaktiven Systemen nach zwei Gesichtspunkten beurteilt werden (siehe Abbildung 1.2). Der erste Punkt befaßt sich mit der weiteren Strukturierung der Dialogschnittstelle, während der zweite Punkt Aussagen über die Anwendungsschnittstelle zuläßt.

Die meisten Verwaltungssysteme können in drei logische Komponenten unterteilt werden [*Green 84*]. Diese Komponenten sind zwar nicht in allen Systemen explizit vorhanden, sie lassen sich aber meistens im konzeptuellen Entwurf wiederfinden. Dabei wird das Dialogsystem in eine Darstellungskomponente, eine Dialogkontrolle und eine Anwendungskomponente gegliedert (siehe Abbildung 6.1).

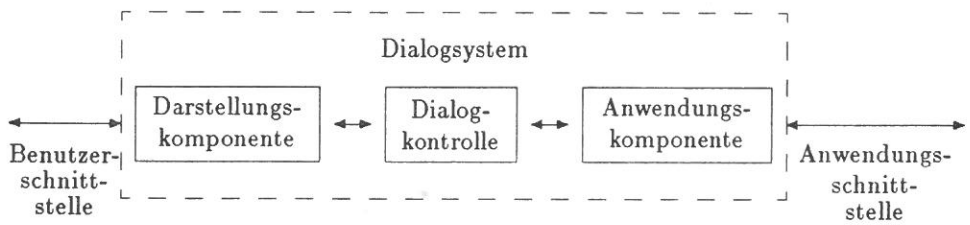

Abbildung 6.1: Struktur eines Dialogsystems

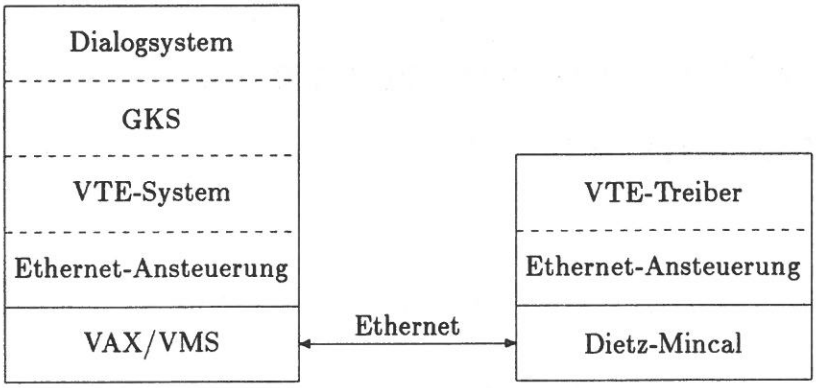

Abbildung B.1: Schichtenstruktur des Gesamtsystems

VAX/VMS		
Teilsystem	*Pakete*	*Quellzeilen*
Dialogsystem	18	7200
GKS	15	13300
VTE-System	9	10500
Ethernet-Ansteuerung	8	860
Dietz-Mincal		
VTE-Treiber	7	2300
Ethernet-Ansteuerung	4	2100

Abbildung B.2: Anzahl der Pakete und Quellzeilen

Anhang B

Schichtenstruktur des gesamten Programmsystems

Das realisierte Dialogsystem verwendet das ebenfalls in Ada implementierte GKS (siehe Kapitel 6.3.2). Das GKS baut auf dem VTE-System auf, das mithilfe einer Ethernet-Ansteuerung mit dem VTE-Treiber auf dem Dietz-Mincal 612×2 kommuniziert (siehe auch Abbildung 6.14). Diese Schichtenstruktur stellt Abbildung B.1 dar.

Alle diese Systeme sind in Ada implementiert und auf dem in Abbildung 6.14 skizzierten Rechnernetz ausführbar. Ihr Umfang gemessen an der Zahl der Pakete bzw. der Quellzeilen wird in Abbildung B.2 aufgeführt.

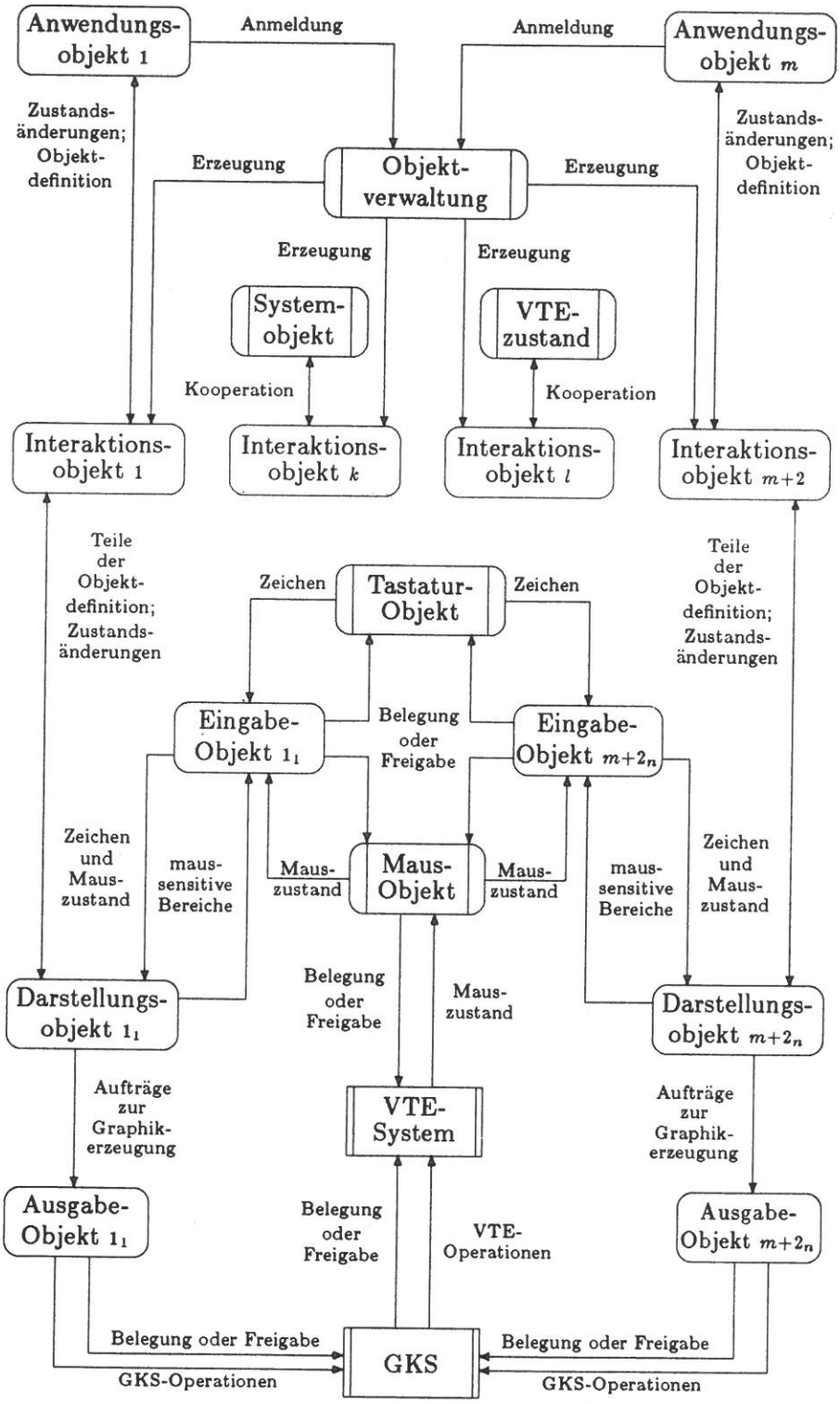

Abbildung A.10: Objektstruktur des Dialogsystems

Abbildung A.10 stellt mit den Symbolen für einfache Objekte, Objekte als Instanzen und für Teilsysteme die vollständige Objektstruktur des Dialogsystems dar. Die beschrifteten Verbindungslinien beschreiben den Fluß der wichtigsten Nachrichten, die zwischen diesen Objekten ausgetauscht werden. Die in Kapitel 6.2 eingeführten Interaktions-, Darstellungs-, Eingabe- und Ausgabe-Objekte wurden als Objektklassen, das Systemobjekt, das Objekt VTE-Zustand, die Objektverwaltung, das Tastatur-Objekt und das Maus-Objekt als einfache Objekte realisiert.

Zur Übermittlung der Nachrichten zwischen den Objekten wurde ein Nachrichtensystem implementiert, das die in Kapitel 6.2.7 aufgestellten Forderungen zur Form der Objektkommunikation erfüllt. Die Nachrichten werden einheitlich als Zeichenketten dargestellt. Für eine ausführliche Erörterung derartiger Nachrichtensysteme und die Möglichkeiten ihrer Realisierung sei auf *Faasch 86* verwiesen.

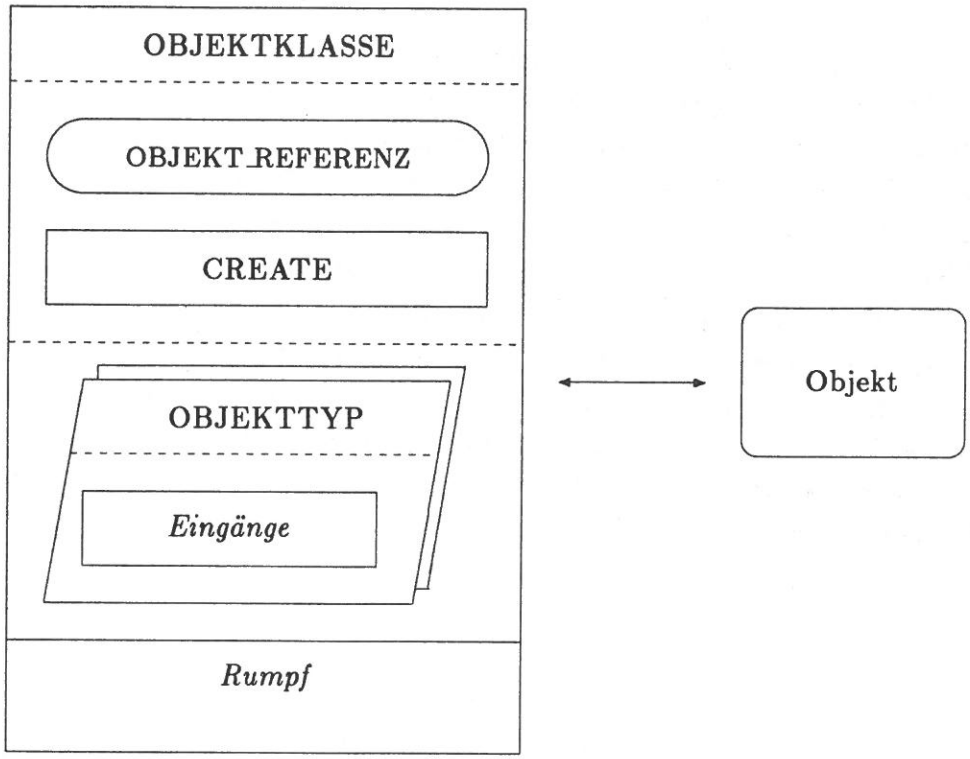

Abbildung A.8: Eine Objektklasse

und die auf ihn anwendbaren Operationen für andere Pakete sichtbar sein sollen. Variablen dieses Zeigertyps speichern Objektnamen. Die Operation "CREATE" erzeugt Instanzen dieser Klasse. Das rechte Symbol beschreibt nachfolgend Objekte, die als Instanzen einer derartigen Klasse erzeugt wurden.

Abbildung A.9 zeigt ein Symbol, das nachfolgend einen in sich abgeschlossenen Teil eines Programmsystems (z.B. GKS) repräsentiert.

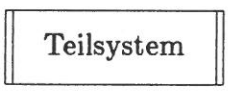

Abbildung A.9: Symbol für ein Teilsystem

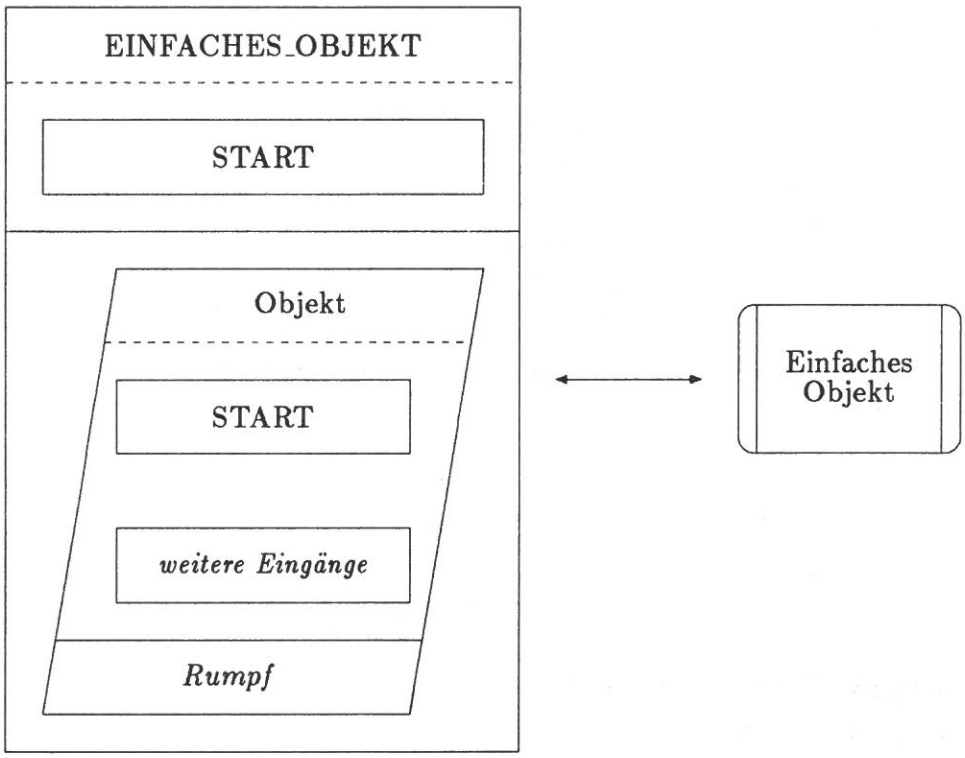

Abbildung A.7: Ein einfaches Objekt

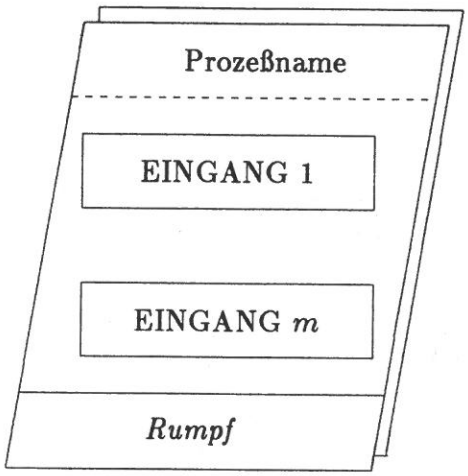

Abbildung A.5: Symbol für einen Prozeßtyp

```
task type PROZESSNAME is
   entry EINGANG1 is ...;
   entry EINGANGm is ...;
end PROZESSNAME;

task body PROZESSNAME is
      Rumpf
end PROZESSNAME;
```

Abbildung A.6: Eine Prozeßtyp-Deklaration

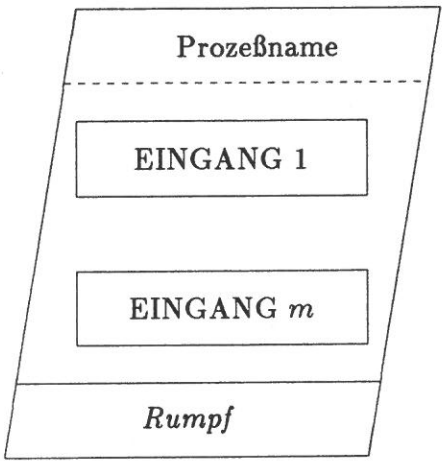

Abbildung A.3: Symbol für einen einfachen Prozeß

```
task PROZESSNAME is
   entry EINGANG1 is ...;
   entry EINGANGm is ...;
end PROZESSNAME;

task body PROZESSNAME is
      Rumpf
end PROZESSNAME;
```

Abbildung A.4: Eine Prozeß-Deklaration

```
package PAKETNAME is
   type DATENTYP1 is ...;
   type DATENTYPk is ...;
   procedure OPERATION1 ...;
   procedure OPERATIONl ...;
private
      privater, nicht sichtbarer Teil der Spezifikation
end PAKETNAME;
```

Abbildung A.2: Eine Paket-Deklaration

Prozeßnamen und dann die exportierten Prozeßeingänge (entries) enthält (siehe Abbildung A.3). Abbildung A.4 zeigt die entsprechende Ada-Deklaration.

Das Symbol für einen Prozeßtyp (task type) ist in Abbildung A.5 und die entsprechende Ada-Deklaration in Abbildung A.6 zu sehen.

Ein Objekt, das im Rahmen des Dialogsystems genau einmal vorhanden ist und nicht durch eine Klasse beschrieben wird, soll nachfolgend als *einfaches Objekt* bezeichnet werden. Dieses einfache Objekt wird im Rahmen des Dialogsystems als ein Paket realisiert, wobei die Paketspezifikation entweder leer ist oder nur eine Prozedur zum expliziten Start des Objektes enthält. Die Kommunikation mit anderen Objekten findet über Nachrichten statt und ist nicht ein Bestandteil der Paketspezifikation. Der Rumpf des Paketes enthält einen Prozeß, der die Operationen des Objektes durchführt und die Daten verwaltet (siehe Abbildung A.7). Das rechte Symbol in dieser Abbildung stellt nachfolgend ein derartig realisiertes einfaches Objekt dar.

Eine Klasse wird im Dialogsystem mithilfe eines Paketes als abstrakter Datentyp realisiert (siehe Abbildung A.8). Dieses Paket spezifiziert einen Zeigertyp (access type) "OBJEKT_REFERENZ" auf einen Prozeßtyp. Durch den Zeigertyp können dynamisch Prozesse und damit Instanzen der Klasse geschaffen werden, wobei die Zeigerwerte als Prozeßnamen und damit als Objektnamen benutzt werden. Weiterhin spezifiziert das Paket die auf diese Klasse anwendbaren Operationen. Die Deklaration des Prozeßtyps erfolgt im privaten Teil der Spezifikation, da nur der Zeigertyp

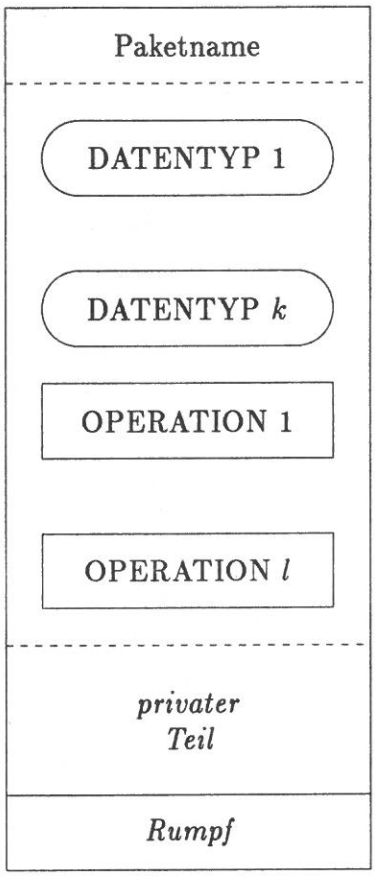

Abbildung A.1: Symbol für ein Paket

Die in Kapitel 6.2 vorgestellten Objekte des Dialogsystems wurden mithilfe von Klassen realisiert. Eine Ausnahme bilden die Objekte, die aufgrund ihrer Funktion nur genau einmal im System vorhanden sind (z.B. das Maus-Objekt).

Zur schematischen Beschreibung der Implementation der Objekte bzw. ihrer Klassen werden Bildsymbole verwendet, die sich an einer graphischen Notation für den objektorientierten Entwurf in Ada [*Booch 81*, *Booch 82*, *Booch 86*] orientieren. Ein Paket wird als ein Rechteck dargestellt, das oben den Paketnamen enthält. Dann folgen die exportierten Datentypen und Operationen, der private Teil der Spezifikation und der Paketrumpf (siehe Abbildung A.1). Dieses Symbol kann leicht in eine entsprechende Ada-Deklaration überführt werden (siehe Abbildung A.2).

Ein Prozeß (task) wird als Parallelogramm dargestellt, das oben den

Anhang A

Objektorientierte Programmierung in Ada

Die Konzepte der *objektorientierten Programmierung* [*Rentsch 82, Stefik & Bobrow 86, Booch 86*] wurden für die Implementation des Dialogsystems angewendet. Da Ada im Gegensatz zu anderen Programmiersprachen (z.B. Smalltalk-80) nicht direkt für die objektorientierte Programmierung entwickelt wurde [*Rentsch 82*], mußten für die Realisierung des Dialogsystems erst entsprechende Grundlagen geschaffen werden. Die Möglichkeiten der objektorientierten Programmierung in Ada und ihre Anwendung auf den Bereich der Bildfolgenauswertung werden ausführlich in *Faasch 86* erörtert. Die nachfolgende Darstellung der objektorientierten Programmierung beschränkt sich deshalb auf die Grundlagen, die ein besseres Verständnis der Implementation des Dialogsystems ermöglichen. Es wird ebenfalls auf eine Diskussion alternativer Implementationsstrategien verzichtet und stattdessen auf die in *Faasch 86* beschriebene Untersuchung verwiesen.

Die grundlegende Idee bei der objektorientierten Programmierung besteht darin, ein System zu realisieren, das sich nur aus Objekten zusammensetzt. Die Kommunikation der Objekte findet dabei ausschließlich über Nachrichten statt. Ein *Objekt* wird somit als eine Einheit im Programmsystem verstanden, die Prozeduren und Daten zusammenfaßt, eigene Berechnungen parallel zu und unabhängig von anderen Objekten durchführt und von anderen Objekten nur über Nachrichten angesprochen werden kann. Ein Objekt kann eigene Daten und Prozeduren zur Durchführung von Operationen besitzen. Ein Objekt kann in anderen Objekten mithilfe von Nachrichten das Ausführen von Operationen bewirken. Die Nachrichten realisieren somit eine indirekte Form der Prozeduraufrufe.

Eine *Klasse* besteht aus einer Beschreibung von einem oder mehreren strukturell gleichartigen Objekten. Ein Objekt, das aus einer Klasse hervorgegangen ist, wird als *Instanz* dieser Klasse bezeichnet.

Wasserman 85: *Extending State Transition Diagrams for the Specification of Human-Computer Interaction*, A.I. Wasserman, IEEE Transactions on Software Engineering, Vol. SE-11, No. 8, Aug. 1985, pp. 699–713.

Westphal 84: *Dreidimensionale Modellierung bewegter Objekte und Ausnutzung von Helligkeitsveränderungen zur Formbestimmung*, H. Westphal, *Dissertation*, Universität Hamburg, Fachbereich Informatik, November 1984.

Williams 83: *The Lisa computer system*, G. Williams, Byte, Vol. 8, No. 2, 1983, pp. 33–50.

Williams 84: *The Apple MacIntosh system*, G. Williams, Byte, Vol. 9, No. 2, 1984, pp. 30–40.

Wong & Reid 82: *Flair — User Interface Dialog Design Tool*, P.C.S. Wong, E.R. Reid, ACM Computer Graphics, Vol. 16, No. 3, July 1982, pp. 87–98.

Stefik & Bobrow 86: *Object-Oriented Programming: Themes and Variations*, M. Stefik, D.G. Bobrow, The AI Magazine, Vol. 6, No. 4, Winter 1986, pp. 40–62.

Stevens & Alexander 83: *Image-processing software portability using a conceptual frame store*, R.J. Stevens, S.T. Alexander, Pattern Recognition Letters, Vol. 1, No. 5/6, July 1983, pp. 359–364.

Stevens & Hunt 82: *Software pipelines in image processing*, W.R. Stevens, B.R. Hunt, Computer Vision, Graphics, and Image Processing, Vol. 20, No. 1, 1982, pp. 90–95.

Tamura et al. 83: *Design and Implementation of SPIDER — A Transportable Image Processing Software Package*, H. Tamura, S. Sakane, F. Tomita, N. Yokoya, M. Kaneko, K. Sakaue, Computer Vision, Graphics, and Image Processing, Vol. 23, 1983, pp. 273–294.

Tanimoto 82: *Advances in Software Engineering and their Relations to Pattern Recognition and Image Processing*, S.L. Tanimoto, Pattern Recognition, Vol. 15, No. 3, 1982, pp. 113–120.

Teitelman 85: *A Tour Through Cedar*, W. Teitelman, IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, Mar. 1985, pp. 285–302.

Teitelman & Masinter 81: *The Interlisp Programming Environment*, W. Teitelman, L. Masinter, Computer, Vol. 14, No. 4, April 1981, pp. 25–33.

Tesler 81: *The Smalltalk Environment*, L. Tesler, Byte, Vol. 6, No. 8, Aug. 1981, pp. 90–147.

Triendl et al. 82: *Design of an Interactive Picture Processing System*, E. Triendl, R. Fiedler, H. Helbig, G. Kritikos, D. Kübler, M. Lehner, Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing, June 14–17, Las Vegas, Nevada 1982, pp. 534–536.

Uhr 81: *A Language for Parallel Processing of Arrays, Embedded in PASCAL*, L. Uhr, In: *Duff & Levialdi 81*, pp. 53–88.

Vrolijk et al. 81: *TAL: An Interpretative Language for the Leyden Television Analysis System*, J. Vrolijk, P.L. Pearson, J.S. Ploem, In: *Duff & Levialdi 81*, pp. 125–138.

Roach & Nickson 83: *Formal Specifications For Modeling and Developing Human/Computer Interfaces*, J.W. Roach, M. Nickson, In: *Janda 83*, pp. 35–39.

Rosenthal 83: *Managing Graphical Resources*, D.S.H. Rosenthal, ACM Computer Graphics, Vol. 17, No. 1, Jan. 1983, pp. 38–45.

Rowe & Shoens 83: *Programming Language Constructs for Screen Definition*, L.A. Rowe, K.A. Shoens, IEEE Transactions on Software Engineering, Vol. SE-9, No. 1, Jan. 1983, pp. 31–39.

Rubin et al. 85: *ThinkPad: A Graphical System for Programming by Demonstration*, R.V. Rubin, E.J. Golin, S.P. Reiss, IEEE Software, Vol. 2, No. 3, Mar. 1985, pp. 73–79.

Sandewall 78: *Programming in the Interactive Environment: the "LISP" Experience*, E. Sandewall, ACM Computing Surveys, Vol. 10, No. 1, March 1978, pp. 35–71.

Schulert et al. 85: *ADM — A Dialog Manager*, A.J. Schulert, G.T. Rogers, J.A. Hamilton, In: *Borman & Curtis 85*, pp. 177–183.

Shaw et al. 83: *Descartes: A Programming-Language Approach to Interactive Display Interfaces*, M. Shaw, E. Borison, M. Horowitz, T. Lane, D. Nichols, R. Pausch, ACM Sigplan Notices, Vol. 18, No. 6, June 1983, pp. 100–111.

Shneiderman 82: *Multiparty Grammars and Related Features for Defining Interactive Systems*, B. Shneiderman, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-12, No. 2, March/April 1982, pp. 148–154.

Shneiderman 83: *Direct Manipulation: A Step Beyond Progamming Languages*, B. Shneiderman, Computer, Vol. 16, No. 8, Aug. 1983, pp. 57–69.

Shu 85: *FORMAL: A Forms-Oriented, Visual-Directed Application Development System*, N.C. Shu, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 38–49.

Smith et al. 82: *Designing the Star User Interface*, D.C. Smith, C. Irby, R. Kimball, B. Verplank, Byte, Vol. 7, No. 4, April 1982, pp. 242–282.

Olsen et al. 85: *Input/Output Linkage in a User Interface Management System*, D.R. Olsen, Jr., E.P. Dempsey, R. Rogge, ACM Computer Graphics, Vol. 19, No. 3, July 1985, pp. 191–197.

Petters 85: *Bildverarbeitung unter UNIX*, H. Petters, Robotersysteme, Vol. 1, No. 1, 1985, pp. 3–6.

Pfaff & Maderlechner 82: *Tools for Configuring Interactive Picture-Processing Systems*, G.E. Pfaff, G. Maderlechner, IEEE Computer Graphics and Applications, Vol. 2, No. 5, July 1982, pp. 35–49.

Preston 80: *Image manipulation languages — A preliminary survey*, K. Preston, Jr, In: *Gelsema & Kanal 80*, pp. 207–211.

Preston 81: *Image processing software — A survey*, K. Preston, Jr, In: *Kanal & Rosenfeld 81*, pp. 123–148.

Preston 83: *Progress in Image Processing Languages*, K. Preston, Jr, In: *Duff 83*, pp. 207–211.

Radhakrishnan et al. 81: *Design of a High-level Language (L) for Image Processing*, T. Radhakrishnan, R. Barrera, A. Guzman, A. Jinich, In: *Duff & Levialdi 81*, pp. 25–40.

Raeder 85: *A Survey of Current Graphical Programming Techniques*, G. Raeder, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 11–24.

Reid 80: *Scribe: A Documentation Specification Language and its Compiler*, B.K. Reid, *PhD. Thesis*, Carnegie-Mellon University, Pittsburgh/PA, Oct. 1980.

Reisner 81: *Formal Grammar and Human Factors Design of an Interactive Graphics System*, P. Reisner, IEEE Transactions on Software Engineering, Vol. SE-7, No. 2, Mar. 1981, pp. 229–240.

Rentsch 82: *Object-Oriented Programming*, T. Rentsch, ACM Sigplan Notices, Vol. 17, No. 9, Sep. 1982, pp. 51–57.

Ritchie & Thompson 74: *The UNIX Time-sharing System*, D.M. Ritchie, K. Thompson, Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 365–375.

Myers 84: *The User Interface for Sapphire: A Screen Allocation Package Providing Helpful Icons and Rectangular Environments*, B.A. Myers, IEEE Computer Graphics and Applications, Vol. 4, No. 12, Dec. 1984, pp. 13–23.

Myers 85: *The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces*, B.A. Myers, In: *Borman & Curtis 85*, pp. 11–17.

Nagel & Enkelmann 84: *Untersuchung von Verschiebungsvektorfeldern in Bildbereichen mit linienhaften oder partiell homogenen Grauwertverteilungen*, H.-H. Nagel, W. Enkelmann, DAGM/ÖAGM Symposium, Graz 2.–4. Okt. 1984, Informatik Fachberichte Nr. 87, W. Kropatsch (Hrsg.), Springer Verlag, Berlin, 1984, pp. 154–160.

Nagel & Enkelmann 86: *An Investigation of Smoothness Constraints for the Estimation of Displacement Vector Fields from Image Sequences*, H.-H. Nagel, W. Enkelmann, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986 (to appear).

Nagl 84: *Ada und Smalltalk-80: Ein summarischer Vergleich*, M. Nagl, Technical Report, Osnabrücker Schriften zur Mathematik, Reihe I, Heft 16, Fachbereich Mathematik, Universität Osnabrück, 1984.

Norman 81: *The Trouble with Unix*, D.A. Norman, Datamation, November 1981, pp. 139–150.

Olsen 83: *Automatic Generation of Interactive Systems*, D.R. Olsen, Jr, ACM Computer Graphics, Vol. 17, No. 1, Jan. 1983, pp. 53–57.

Olsen & Dempsey 83a: *SYNGRAPH: A Graphical User Interface Generator*, D.R. Olsen, Jr., E.P. Dempsey, ACM Computer Graphics, Vol. 17, No. 3, July 1983, pp. 43–50.

Olsen & Dempsey 83b: *Syntax Directed Graphical Interaction*, D.R. Olsen, Jr., E.P. Dempsey, ACM Sigplan Notices, Vol. 18, No. 6, June 1983, pp. 112–117.

Olsen et al. 84: *A Context for User Interface Management*, D.R. Olsen, Jr., W. Buxton, R. Ehrich, D.J. Kasik, J.R. Rhyne, J. Sibert, IEEE Computer Graphics and Applications, Vol. 4, No. 12, Dec. 1984, pp. 33–42.

Mansfeld 84: *Graphische Zustandsdarstellung für ein Bildverarbeitungssystem*, R. Mansfeld, *Studienarbeit*, Universität Hamburg, Fachbereich Informatik, August 1984.

Marcus 84: *Corporate identity for iconic interface design: the graphic design perspective*, A. Marcus, Interfaces in Computing, Vol. 2, No. 3, Nov. 1984, pp. 365–378. Auch erschienen in: IEEE Computer Graphics and Applications, Vol. 4, No. 12, Dec. 1984, pp. 24–32.

McCracken & Akscyn 84: *Experience with the ZOG Human-Computer Interface System*, D.L. McCracken, R.M. Akscyn, International Journal of Man-Machine Studies, Vol. 21, No. 4, Oct. 1984, pp. 293–310.

McKeown & Denlinger 82: *Graphical Tools for Interactive Image Interpretation*, D.M. McKeown, J.L. Denlinger, ACM Computer Graphics, Vol. 16, No. 3, Jul. 1982, pp. 189–198.

Meinzer & Engelmann 82: *A Language for the Interactive Manipulation of Digitized Images*, H.P. Meinzer, U. Engelmann, Proceedings of the 6th International Conference on Pattern Recognition, München 1982, pp. 68–70.

Meinzer & Engelmann 83: *Eine Kommandosprache zur Bildverarbeitung*, H.P. Meinzer, U. Engelmann, Angewandte Informatik, Vol. 10/83, pp. 434–439.

Melamed & Morris 85: *Visual Simulation: The Performance Analysis Workstation*, B. Melamed, R.J.T. Morris, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 87–94.

Moran 81: *The Command Language Grammar: a representation for the user interface of interactive computer systems*, T.P. Moran, International Journal of Man-Machine Studies, Vol. 15, No. 1, July 1981, pp. 3–50.

Moriconi & Hare 85: *Visualizing Program Designs Through PegaSys*, M. Moriconi, D.F. Hare, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 72–85.

Myers 83: *INCENSE: A System for displaying Data Structures*, B.A. Myers, ACM Computer Graphics, Vol. 17, No. 3, July 1983, pp. 115–125.

Kulpa 81a: *Universal digital image processing systems in Europe — a comparative survey*, Z. Kulpa, In: *Bolc & Kulpa 81*, pp. 1–20.

Kulpa 81b: *PICASSO, PICASSO-SHOW and PAL — A Development of a High-level Software System for Image Processing*, Z. Kulpa, In: *Duff & Levialdi 81*, pp. 13–24.

Landy et al. 84: *HIPS: A Unix-Based Image Processing System*, M.S. Landy, Y. Cohen, G. Sperling, Computer Vision, Graphics, and Image Processing, Vol. 25, 1984, pp. 331–347.

Levialdi et al. 81: *On the Design and Implementation of PIXAL, a Language for Image processing*, S. Levialdi, A. Maggiolo-Schettini, M. Napoli, G. Tortora, G. Uccella, In: *Duff & Levialdi 81*, pp. 89–98.

Levine 78: *A knowledge-based Computer Vision System*, M.D. Levine, In: *Hanson & Riseman 78*, pp. 335–352.

Lieberman 84: *Seeing what your programs are doing*, H. Lieberman, International Journal of Man-Machine Studies, Vol. 21, No. 4, Oct. 1984, pp. 311–331.

Lieberman 85: *There's More to Menu Systems Than Meets the Screen*, H. Lieberman, ACM Computer Graphics, Vol. 19, July 1985, pp. 181–189.

Lipkie et al. 82: *Star Graphics: An Object-Oriented Implementation*, D.E. Lipkie, S.R. Evans, J.K. Newlin, R.L. Weissman, ACM Computer Graphics, Vol. 16, No. 3, July 1982, pp. 115–124.

Lipkin & Rosenfeld 70: *Picture Processing and Psychopictorics*, B.S. Lipkin, A. Rosenfeld (eds.), Academic Press, New York, 1970.

London & Duisberg 85: *Animating Programs using Smalltalk*, R.L. London, R.A. Duisberg, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 61–71.

Maaß 84: *Mensch-Rechner-Kommunikation — Herkunft und Chancen eines neuen Paradigmas*, S. Maaß, *Dissertation*, Universität Hamburg, Fachbereich Informatik, Juli 1984. Auch erschienen als FBI-HH-B-104/84.

Maggiolo-Schettini 81: *Comparing some High-level Languages for Image Processing*, A. Maggiolo-Schettini, In: *Duff & Levialdi 81*, pp. 157–164.

Hayes et al. 85: *Design Alternatives for User Interface Management Systems Based on Experience with Cousin*, P.J. Hayes, P.A. Szekely, R.A. Lerner, In: *Borman & Curtis 85*, pp. 169–175.

Hoare 78: *Communicating Sequential Processes*, C.A.R. Hoare, Communications of the ACM, Vol. 21, No. 8, Aug. 1978, pp. 666–678.

Ingalls 81: *Design Principles Behind Smalltalk*, D.H.H. Ingalls, Byte, Vol. 6, No. 8, Aug. 1981, pp. 286–298.

Jacob 83a: *Using Formal Specifications in the Design of a Human-Computer Interface*, R.J.K. Jakob, Communications of the ACM, Vol. 26, No. 4, April 1983, pp. 259–264.

Jacob 83b: *Executable Specifications for a Human-Computer Interface*, R.J.K. Jakob, In: *Janda 83*, pp. 28–34.

Jacob 85: *A State Transition Diagram Language for Visual Programming*, R.J.K. Jacob, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 51–59.

Janda 83: *Human Factors in Computing Systems*, A. Janda (ed.), CHI'83 Conference Proceeedings, Dec. 12–15, Boston, 1983.

Johnston 70: *The PAXII Picture Processing System*, E.G. Johnston, In: *Lipkin & Rosenfeld 70*, pp. 427–512.

Kanal & Rosenfeld 81: *Progress in Pattern Recognition*, L.N. Kanal, A. Rosenfeld (eds.), Vol. 1, North-Holland Publishing Company, Amsterdam, 1981.

Kasik 82: *A User Interface Management System*, D.J. Kasik, ACM Computer Graphics, Vol. 16, No. 3, July 1982, pp. 99–106.

Kernighan & Mashey 81: *The Unix Programming Environment*, B.W. Kernighan, J.R. Mashey, Computer, Vol. 14, No. 4, April 1981, pp. 12–24.

Kieras & Polson 83: *A Generalized Transition Network Representation for Interactive Systems*, D. Kieras, P.G. Polson, In: *Janda 83*, pp. 103–106.

Krusemark & Haralick 82: *Achieving Portability in Image Processing Software Packages*, S. Krusemark, R.M. Haralick, Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing, June 14–17, Las Vegas, Nevada 1982, pp. 451–457.

Gelsema & Kanal 80: *Pattern Recognition in Practice*, E.S. Gelsema, L.N. Kanal (eds.), North-Holland Publishing Company, Amsterdam, 1980.

Gittins et al. 84: *An icon-driven end-user interface to UNIX*, D.T. Gittins, R.L. Winder, H.E. Bez, International Journal of Man-Machine Studies, Vol. 21, No. 5, Nov. 1984, pp. 451–461.

Glinert & Tanimoto 84: *PICT: An Interactive Graphical Programming Environment*, E.P. Glinert, S.L. Tanimoto, IEEE Computer, Vol. 17, No. 11, Nov. 1984, pp. 7–25.

Goldberg & Robson 83: *Smalltalk-80: The Language and its Implementation*, A. Goldberg, D. Robson, Addison-Wesley, Reading, Mass., 1983.

Green 83: *A Catalogue of Graphical Interaction Techniques*, M. Green, ACM Computer Graphics, Vol. 17, No. 1, Jan. 1983, pp. 46–52.

Green 84: *Report on Dialogue Specification Tools*, M. Green, Computer Graphics Forum, Vol. 3, No. 4, 1984, pp. 305–313.

Green 85: *The University of Alberta User Interface Management System*, M. Green, ACM Computer Graphics, Vol. 19, No. 3, July 1985, pp. 205–213.

Gudmundsson 82: *An Interactive High-Level Language System for Picture Processing*, B. Gudmundsson, Computer Graphics and Image Processing, Vol. 18, 1982, pp. 392–403.

Hanson & Riseman 78: *Computer Vision Systems*, A.R. Hanson, E.M. Riseman (eds.), Academic Press, New York, 1978.

Haralick & Minden 78: *KANDIDATS: An Interactive Image Processing System*, R.M. Haralick, G. Minden, Computer Graphics and Image Processing, Vol. 8, pp. 1–15, 1978.

Hayes 84: *Executable Interface Definitions using Form-based Interface Abstractions*, P.J. Hayes, *Technical Report* CMU-CS-84-110, Carnegie-Mellon University, Pittsburgh/PA, March 1984.

Hayes & Szekely 83: *Graceful interaction through the COUSIN command interface*, P.J. Hayes, P.A. Szekely, International Journal of Man-Machine Studies, Vol. 19, No. 3, Sep. 1983, pp. 285–306.

Enderle et al. 84: *Computer Graphics Programming — The Graphics Standard*, G. Enderle, K. Kansy, G. Pfaff, Symbolic Computation and Computer Graphics, Springer Verlag, Berlin, 1984.

Engelmann & Meinzer 84: *PIC — Ein Interpreter zur Bildverarbeitung*, U. Engelmann, H.P. Meinzer, *Technical Report* Nr. 25-2, Deutsches Krebsforschungszentrum, Institut für Dokumentation, Information und Statistik, Heidelberg, März 1984.

Enkelmann 85: *Mehrgitterverfahren zur Ermittlung von Verschiebungsvektorfeldern*, W. Enkelmann, *Dissertation*, Universität Hamburg, Fachbereich Informatik, Juli 1985.

Faasch 86: *Systemgestaltung einer Experimentalumgebung für die Bildverarbeitung in Ada*, H. Faasch, Universität Hamburg, Fachbereich Informatik, 1986 (in Vorbereitung).

Faasch & Haarslev 85: *Konzeption einer neuen Ada-Programmierumgebung für die Bildfolgenauswertung*, H. Faasch, V. Haarslev, Mustererkennung 1985, 7. DAGM-Symposium Erlangen, 24.-26. Sep. 1985, Informatik-Fachberichte Nr. 107, H. Niemann (Hrsg.), pp. 191–196.

Faasch et al. 85: *Erfahrungen mit dem Ada-HH Übersetzer*, H. Faasch, V. Haarslev, H.-H. Nagel, Universität Hamburg, Fachbereich Informatik, Ada-HH Compiler Projekt, Abschlußbericht zum DFG-Forschungsvorhaben, Mai 1985.

Foley & van Dam 82: *Fundamentals of Interactive Computer Graphics*, J.D. Foley, A. van Dam, Addison-Wesley Publishing Co., Reading/MA, 1982.

Foley et al. 84: *The Human Factors of Computer Graphics Interaction Techniques*, J.D. Foley, V.L. Wallace, P. Chan, IEEE Computer Graphics and Applications, Vol. 4, No. 11, Nov. 1984, pp. 13–48.

Fukushima et al. 81: *An Interactive Image Processing System Implemented on Small Scale Hardware*, S. Fukushima, Y. Kimura, T. Soma, Memoirs of the Faculty of Industrial Arts, Kyoto Technical University, Science and Technology, Vol. 30, Dec. 81.

GIIT 83: *Graphical Input Interaction Technique (GIIT)*, Workshop Summary, Battele Seattle Conference Center, ACM Computer Graphics, Vol. 17, No. 1, Jan. 1983, pp. 5–66.

Dengler & Meinzer 85: *PICAPL — Ein interaktives Bildanalysesystem*, J. Dengler, H.P. Meinzer, *Technical Report* Nr. 2, Deutsches Krebsforschungszentrum, Abt. Medizinische und Biologische Informatik, Heidelberg, 1985.

Douglass 81: *MAC: A Programming Language for Asynchronous Image Processing*, R.J. Douglass, In: *Duff & Levialdi 81*, pp. 41–52.

Draper & Norman 85: *Software Engineering for User Interfaces*, S.W. Draper, D.A. Norman, IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, Mar. 1985, pp. 252–258.

Dreschler 81: *Ermittlung markanter Punkte auf den Bildern bewegter Objekte und Berechnung einer 3D-Beschreibung auf dieser Grundlage*, L. Dreschler, *Dissertation*, Universität Hamburg, Fachbereich Informatik, Juni 1981. Auch erschienen als Bericht Nr. FBI-HH-B-83/81.

Dreschler & Nagel 82: *Volumetric Model and 3D-Trajectory of a Moving Car Derived from Monocular TV Frame Sequences of a Street Scene*, L. Dreschler, H.-H. Nagel, Computer Graphics and Image Processing, Vol. 20, 1982, pp. 199–228.

Dreschler-Fischer et al. 83: *Lernen durch Beobachtung von Szenen mit bewegten Objekten: Phasen einer Systementwicklung*, L.S. Dreschler-Fischer, W. Enkelmann, H.-H. Nagel, 5. DAGM-Symposium Karlsruhe, 11.–13. Okt. 1983, H. Kazmierczak (Hrsg.), Mustererkennung, VDE-Fachberichte Nr. 35, VDE-Verlag Berlin Offenbach, 1983, pp. 29–34.

Dreschler-Fischer & Haarslev 85: *Konzeption für ein Bildverarbeitungssystem zur Lösung des Korrespondenzproblems bei Stereo-Bildfolgen im Rahmen einer komfortablen ADA-Programmierumgebung*, L.S. Dreschler-Fischer, V. Haarslev, Robotersysteme, Vol. 1, 1985, pp. 29–34.

Duff 83: *Computing Structures for Image Processing*, M.J.B. Duff (ed.), Academic Press, London, 1983.

Duff & Levialdi 81: *Languages and Architectures for Image Processing*, M.J.B. Duff, S. Levialdi (eds.), Academic Press, London, 1981.

Edmonds 82: *The man-computer interface: a note on concepts and design*, E.A. Edmonds, International Journal of Man-Machine Studies, Vol. 16, No. 3, April 1982, pp. 231–236.

Buxton et al. 83: *Towards A Comprehensive User Interface Management System*, W. Buxton, M.R. Lamb, D. Sherman, K.C. Smith, ACM Computer Graphics, Vol. 17, No. 3, July 1983, pp. 35–42.

Buzzard & Mudge 85: *Object-Based Computing and the Ada Programming Language*, G.D. Buzzard, T.N. Mudge, Computer, Vol. 18, No. 3, March 1985, pp. 11–19.

Campbell & Habermann 74: *The Specification of Process Synchronization by Path Expressions*, R.H. Campbell, A.N. Habermann, In: *Operating Systems*, International Symposium, Rocquencourt 1974, Lecture Notes in Computer Science, Vol. 16, E. Gelenbe and C. Kaiser (eds.), Springer Verlag, Berlin, 1974, pp. 82–102.

Cardelli & Pike 85: *Squeak: a Language for Communicating with Mice*, L. Cardelli, R. Pike, ACM Computer Graphics, Vol. 19, No. 3, July 1985, pp. 199–204.

Carlson et al. 83: *Software Structure for Display Management Systems*, E.D. Carlson, J.R. Rhyne, D.L. Weller, IEEE Transactions on Software Engineering, Vol. SE-9, No. 4, July 1983, pp. 385–394.

Carroll & Thomas 82: *Metaphor and the Cognitive Representation of Computing Systems*, J.M. Carroll, J.C. Thomas, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-12, No. 2, March/April 1982, pp. 107–116.

Castleman 79: *Image processing software*, K.R. Castleman, Digital Image Processing, Chapter 4, Prentice-Hall, Englewood Cliffs/NJ, 1979.

Coutaz 84a: *Towards Friendly Systems, Design Concepts for Interactive Interfaces*, J. Coutaz, 1984 Southeast ACM Conference Proceedings, Atlanta, April 1984, pp. 56–61.

Coutaz 84b: *A Paradigm For User Interface Architecture*, J. Coutaz, *Technical Report* CMU-CS-84-124, Carnegie-Mellon University, Pittsburgh/PA, May 1984.

Coutaz 85a: *A Layout Abstraction For User-System Interface*, J. Coutaz, ACM SIGCHI bulletin, Vol. 16, No. 3, Jan. 1985, pp. 16–24.

Coutaz 85b: *Abstractions for User Interface Design*, J. Coutaz, IEEE Computer Vol. 18, No. 9, Sept. 1985, pp. 21–34.

Bolc & Kulpa 81: *Digital Image Processing Systems*, L. Bolc, Z. Kulpa (eds.), Lecture Notes in Computer Science, Vol. 109, Springer Verlag, Berlin, 1981.

Booch 81: *Describing Software Design in Ada*, G. Booch, ACM Sigplan Notices, Vol. 16, No. 9, Sep. 1981, pp. 42–47.

Booch 82: *Object-oriented Design*, G. Booch, ACM Ada Letters, Vol. 1, No. 3, Mar./Apr. 1982, pp. 64–76.

Booch 86: *Object-Oriented Development*, G. Booch, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986, pp. 211–221.

Borman & Curtis 85: *Human Factors in Computing Systems*, L. Borman, B. Curtis (eds.), CHI'85 Conference Proceedings, April 14–18, San Francisco, 1985.

Borufka et al. 82: *Dialogue Cells: A Method for Defining Interactions*, H.G. Borufka, H.W. Kuhlmann, P.J.W. ten Hagen, IEEE Computer Graphics and Applications, Vol. 2, No. 5, July 1982, pp. 25–33.

Bos et al. 83: *Input-Output Tools: A Language Facility for Interactive and Real-Time Systems*, J. van den Bos, M.J. Plasmeijer, P.H. Hartel, IEEE Transactions on Software Engineering, Vol. SE-9, No. 3, May 1983, pp. 247–259.

Brinch Hansen 78: *Distributed Processes: A Concurrent Programming Concept*, P. Brinch Hansen, Communications of the ACM, Vol. 21, 1978, pp. 934–940.

Brown & Sedgewick 84: *A System for Algorithm Animation*, M.H. Brown, R. Sedgewick, ACM Computer Graphics, Vol. 18, No. 3, July 1984, pp. 177–186.

Brown & Sedgewick 85: *Techniques for Algorithm Animation*, M.H. Brown, R. Sedgewick, IEEE Software, Vol. 2, No. 1, Jan. 1985, pp. 28–38.

Brown et al. 85: *Program Visualization: Graphical Support for Software Development*, G.P. Brown, R.T. Carling, C.F. Herot, D.A. Kramlich, P. Souza, IEEE Computer, Vol. 18, No. 8, Aug. 1985, pp. 25–35.

Brumfitt 84: *Environments for image processing algorithm development*, P.J. Brumfitt, Image and Vision Computing, Vol. 2, No. 4, 1984, pp. 198–203.

Literatur

Acquah et al. 82: *A Conceptual Model of Raster Graphics Systems*, J. Acquah, J. Foley, J. Sibert, P. Wenner, ACM Computer Graphics, Vol. 16, No. 3, July 1982, pp. 321–328.

Anson 82: *The Device Model of Interaction*, E. Anson, ACM Computer Graphics, Vol. 16, No. 3, July 1982, pp. 107–114.

Balston 81: *A High-level Language for Constructing Image Processing Commands*, D.M. Balston, In: *Duff & Levialdi 81*, pp. 99–116.

Bass 85: *An Approach to User Specification of Interactive Display Interfaces*, L.J. Bass, IEEE Transactions on Software Engineering, Vol. SE-11, No. 8, Aug. 1985, pp. 686–698.

Bengtsson et al. 81: *CELLO — An Interactive System for Image Analysis*, E. Bengtsson, O. Eriksson, T. Jarkrans, B. Nordin, B. Stenkvist, In: *Bolc & Kulpa 81*, pp. 21–45.

Benn & Radig 84: *Symbolische Bildbeschreibung mit nicht-normalisierten Relationen*, W. Benn, B. Radig, DAGM/ÖAGM Symposium, Graz 2.–4. Okt. 1984, Informatik-Fachberichte Nr. 87, W. Kropatsch (Hrsg.), Springer Verlag, Berlin, 1984, pp. 92–98.

Benzon 85: *The Visual Mind and the MacIntosh*, B. Benzon, Byte, Vol. 10, No. 1, Jan. 1985, pp. 113–130.

Bewley et al. 83: *Human Factors Testing in the Design of Xerox's 8010 "Star" Office Workstation*, W.L. Bewley, T.L. Roberts, D. Schroit, W.L. Verplank, In: *Janda 83*, pp. 72–77.

Bobrow 85: *If Prolog is the Answer, What is the Question ? or What it Takes to Support AI Programming Paradigms*, D.G. Bobrow, IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, Nov. 1985, pp. 1401–1408.

noch offenes Problem besteht in der angemessenen Unterstützung der Systemkonstruktion auf der Ebene der Anweisungen und Prozeduren. Die Anwendbarkeit dieses objektorientierten Ansatzes auf den Bereich der symbolischen Deutung von Bildern und Bildfolgen wurde in dieser Untersuchung ausgeklammert.

Die objektorientierte Gestaltung von Benutzerschnittstellen für Bildfolgenauswertesysteme wird als ein Schritt hin zur Entwicklung einer flexiblen, benutzerfreundlichen Ada-Programmierumgebung für die Bildfolgenauswertung verstanden. Die Kombination der objektorientierten Programmierung mit Ansätzen wie beispielsweise der funktionalen und logischen Programmierung und ihre Integration in einer einzigen Umgebung [*Bobrow 85*] erscheint zur Lösung der noch ausstehenden Probleme vielversprechend.

- *Herleitung eines Benutzermodells.*
  Die Hauptaussage des Benutzermodells lautet, daß die Entwickler von Systemen zur Auswertung von Bildern und Bildfolgen ein Systemmodell besitzen, das diese Systeme als eine Sammlung von Verarbeitungskomponenten beschreibt. Der Datenfluß zwischen diesen Komponenten kann mithilfe von Leitungen dargestellt werden. Bei der Interaktion der Entwickler mit diesen Systemen wurde erstmals die Bedeutung der Parameter- und Protokollinteraktion gewürdigt.

- *Objektorientierte und direktmanipulative Benutzerschnittstelle.*
  Die aus dem Benutzermodell abgeleitete Benutzerschnittstelle ist der erste Vorschlag, dem Entwickler eine objektorientierte Interaktion mit Bildfolgenauswertesystemen zu bieten, die auf einer graphischen Systemdarstellung beruht. Weiterhin gestattet sie dem Benutzer die interaktive Gestaltung seines Systems auf der Grundlage von Verarbeitungs-, Daten-, Leitungs-, Speicher- und Gruppenobjekten. Diese Gestaltung findet durch eine direkte Manipulation graphischer Repräsentationen dieser Objekte statt.

- *Interaktion mithilfe von unterschiedlichen Abstraktionsebenen.*
  Es wurde der Versuch unternommen, dem Benutzer Hilfsmittel bereitzustellen, die ihm eine Interaktion auf einer für ihn angemessenen Abstraktionsebene ermöglichen sollen. Dafür wurden Gruppenobjekte zur Strukturierung der Objekte und graphische Ablaufpläne vorgeschlagen. Die durch die Erfassung einer Interaktionshistorie bereitgestellten Daten bilden eine Grundlage für eine automatische Interaktionsausführung, die dem Benutzer die Aktivierung von Ablaufplänen erlaubt.

- *Dialogsystem für eine Klasse von Bildfolgenauswertesystemen.*
  Es wurde ein Dialogsystem realisiert, das für eine ganze Klasse von Bildfolgenauswertesystemen, die mithilfe eines in Arbeit befindlichen Konfigurationssystems [*Faasch 86*] entstehen, den Dialog mit dem Benutzer abwickelt. Das Dialogsystem verwendet zum besseren Verständnis farbige Graphiken. Es ist weiterhin das erste graphische Dialogsystem, das die Konzepte der objektorientierten Programmierung in Ada verwirklicht hat.

Der hier vorgestellte Ansatz unterstützt die Interaktion mit und die Gestaltung von Bildfolgenauswertesystemen auf der Ebene von Moduln. Ein

Kapitel 7

Zusammenfassung

Ein Ziel dieser Arbeit bestand darin, die Interaktion von Benutzern mit Systemen zur Auswertung von Bildern und Bildfolgen zu verbessern. Dafür wurde ein Benutzermodell entwickelt, das die Vorstellung der Benutzer von einem Bildfolgenauswertesystem anhand ihrer Interaktionsziele beschreibt. Diese Vorstellung konnte in Konzepte zur Gestaltung einer Benutzerschnittstelle umgesetzt werden.

Dabei hat sich gezeigt, daß die Modellierung der Interaktion auf der Basis von interaktiven Objekten dieser Vorstellung am besten entspricht. Um diese Modellierung noch zu erweitern, wurden die interaktiven Benutzerschnittstellen von Bildverarbeitungssystemen analysiert. Auf der Basis dieser Erkenntnisse konnte die Modellvorstellung ergänzt und in die Konzeption einer objektorientierten Benutzerschnittstelle umgesetzt werden.

Um die Leistungsfähigkeit der vorgeschlagenen Benutzerschnittstelle zu erproben, wurde ein Prototyp eines entsprechenden Dialogsystems in Ada implementiert. Diese Implementation beschränkt sich auf die grundlegenden Konzepte der Benutzerschnittstelle. Sie ist als eine Testumgebung anzusehen, die als Ausgangspunkt zur Realisierung der vollständigen Benutzerschnittstelle dienen kann.

In dieser Arbeit wurde das erste Mal die Mensch-Maschine-Kommunikation für den Bereich der Bildfolgenauswertung in systematischer Weise untersucht. Dabei konnte gezeigt werden, daß die Benutzerschnittstellen existierender Systeme zur Verarbeitung und Auswertung von Bildern und Bildfolgen keinesfalls ergonomischen Gestaltungskriterien genügen. Die aus diesem Grunde konzipierte Benutzerschnittstelle unterscheidet sich von bekannten Benutzerschnittstellen für die Bildfolgenauswertung insbesondere in den folgenden Punkten:

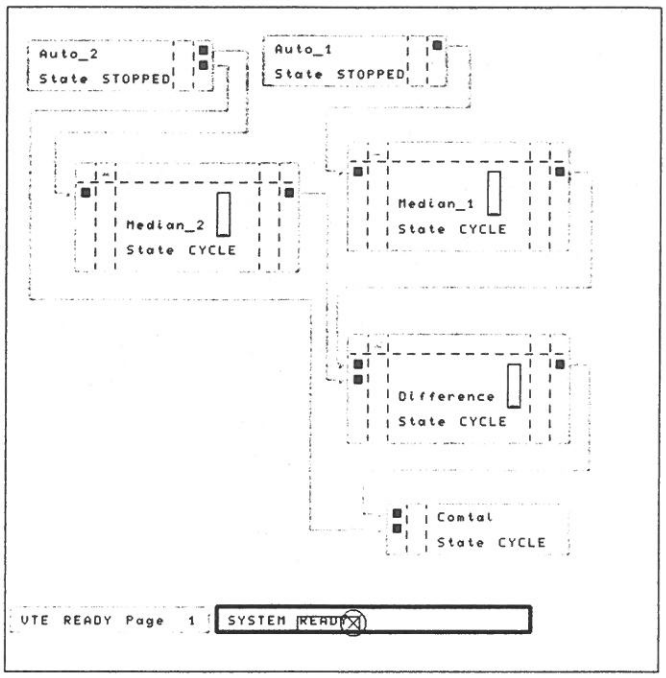

Abbildung 6.63: Auswahl des Objektes "System"

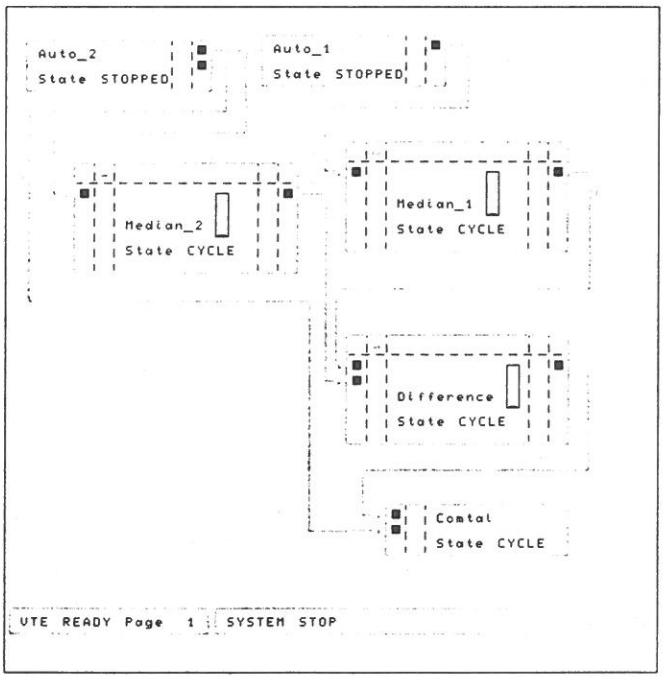

Abbildung 6.64: Abbruch des Systemablaufs

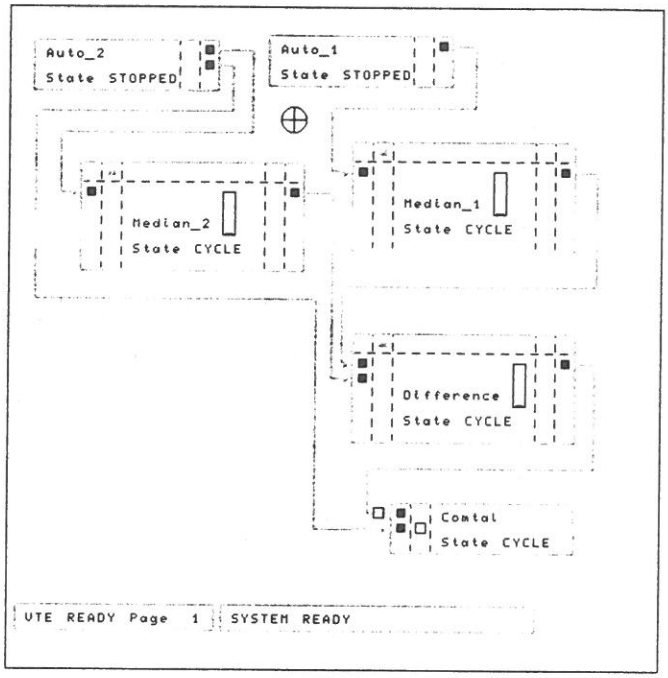

Abbildung 6.61: Systemablauf unter Verwendung von "CYCLE" (7)

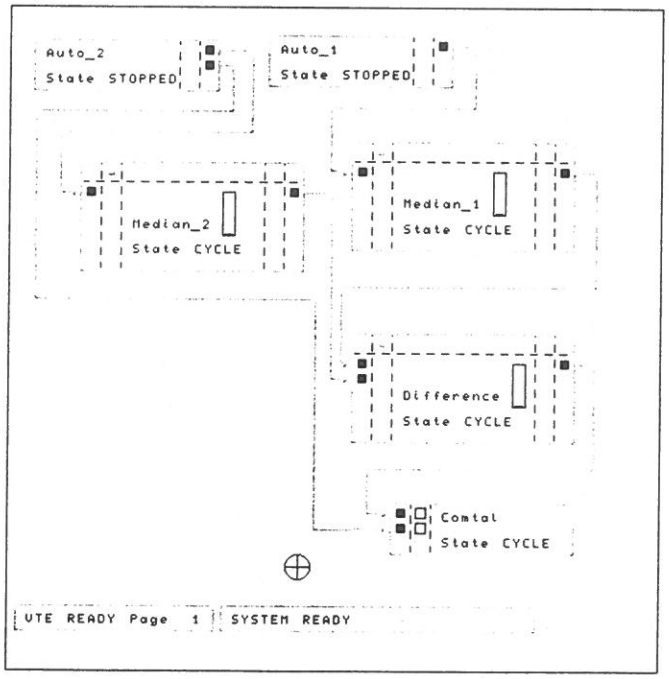

Abbildung 6.62: Systemablauf unter Verwendung von "CYCLE" (8)

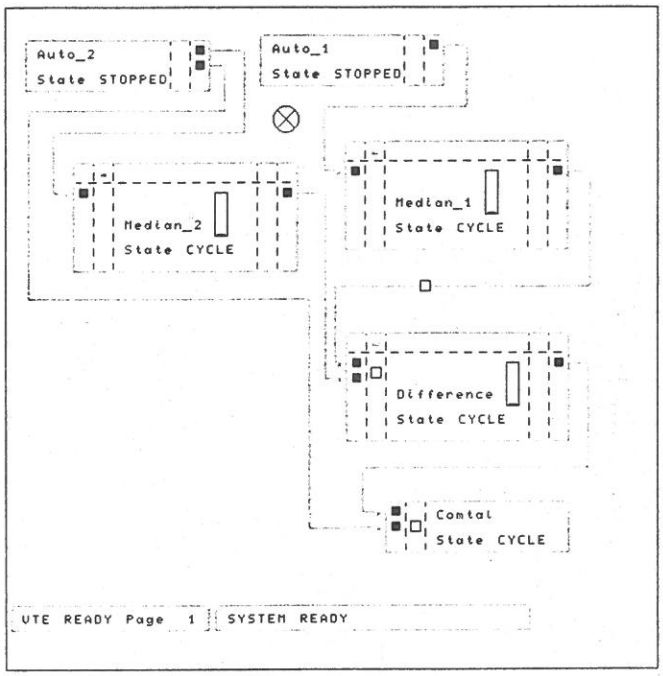

Abbildung 6.59: Systemablauf unter Verwendung von "CYCLE" (5)

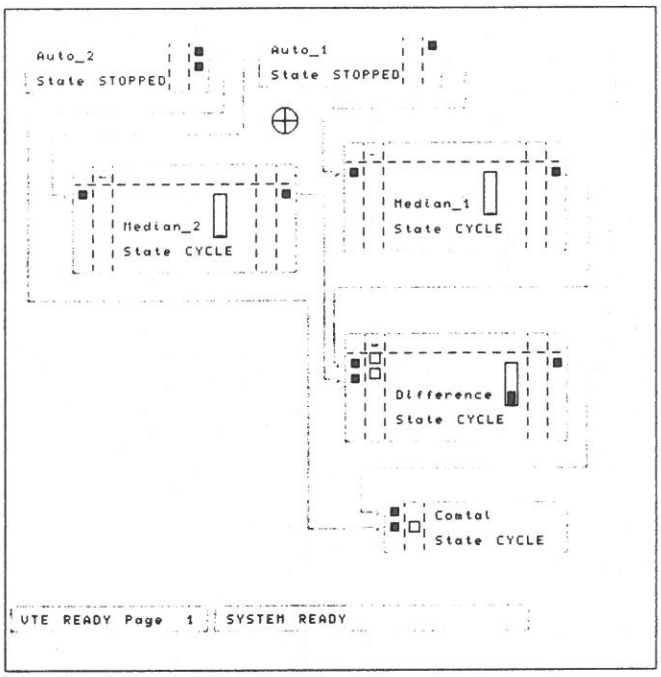

Abbildung 6.60: Systemablauf unter Verwendung von "CYCLE" (6)

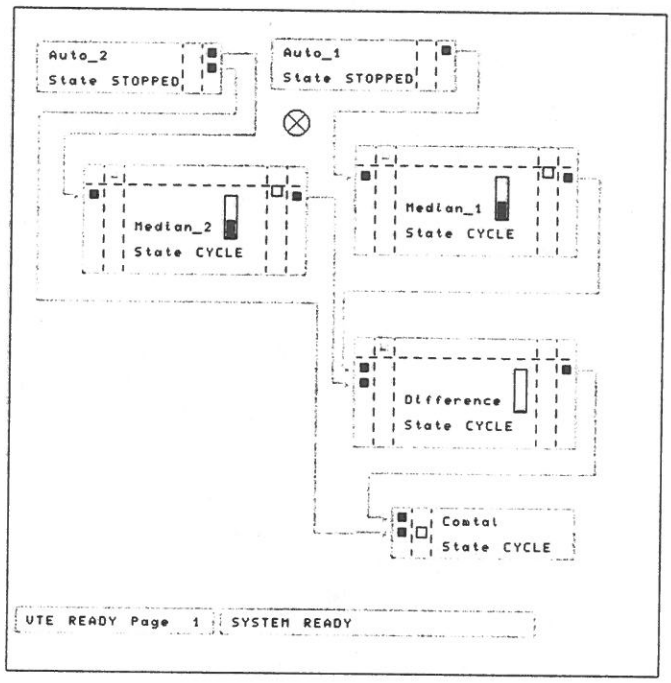

Abbildung 6.57: Systemablauf unter Verwendung von "CYCLE" (3)

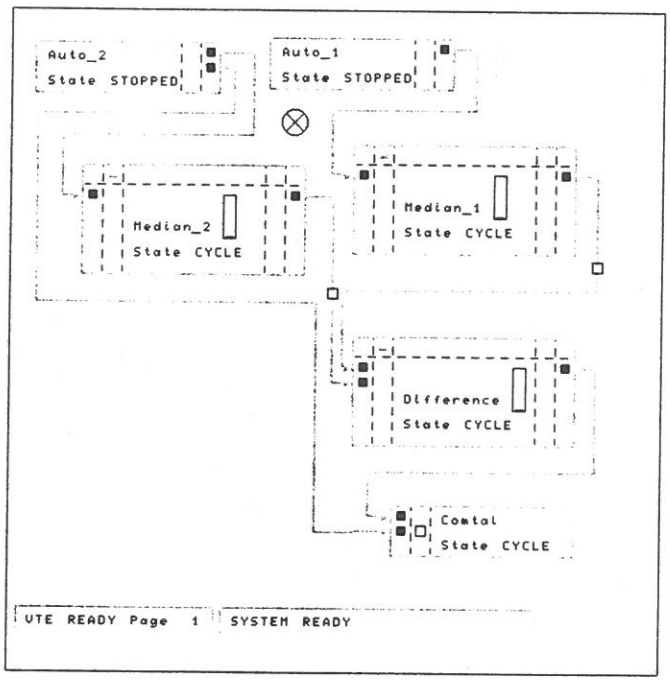

Abbildung 6.58: Systemablauf unter Verwendung von "CYCLE" (4)

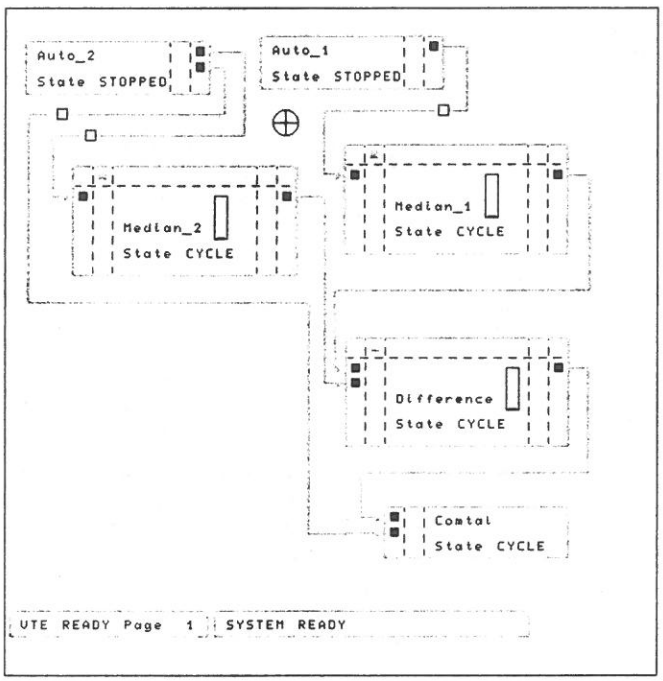

Abbildung 6.55: Systemablauf unter Verwendung von "CYCLE" (1)

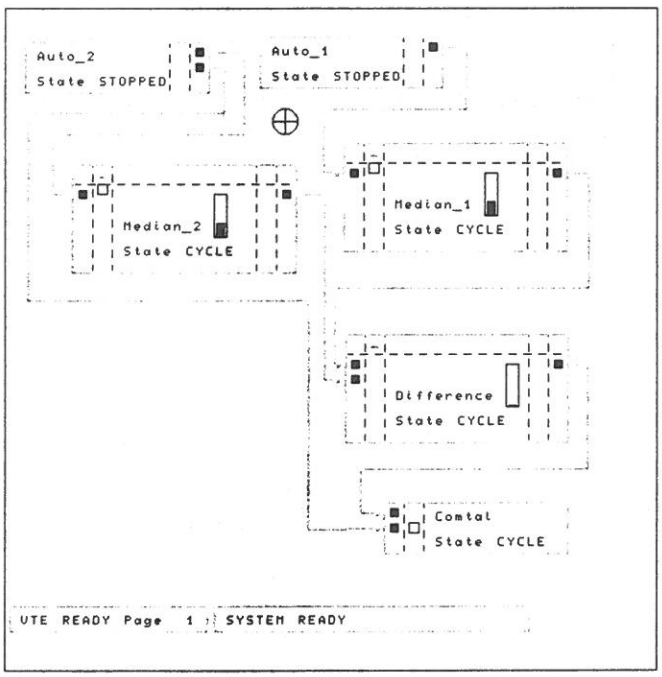

Abbildung 6.56: Systemablauf unter Verwendung von "CYCLE" (2)

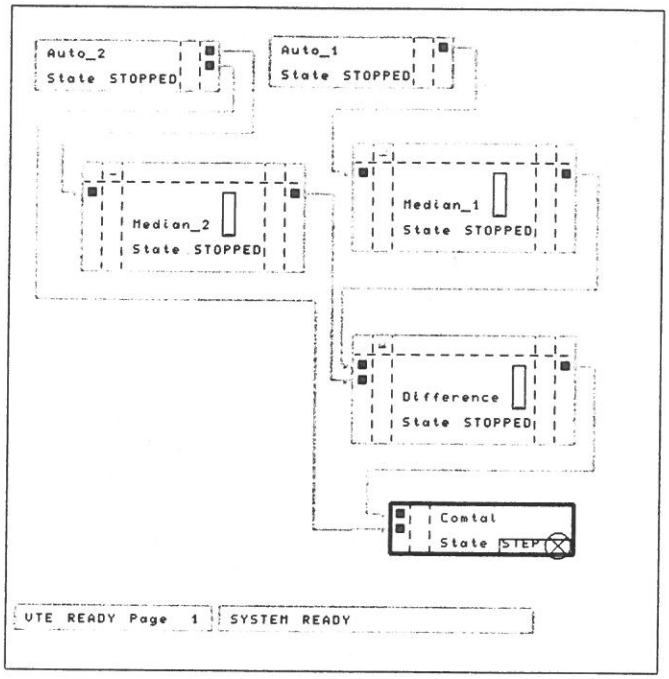

Abbildung 6.54: Nach Verarbeitung der Datenobjekte durch "Comtal"

Die Abbildungen 6.55 bis 6.63 zeigen den oben beschriebenen Systemablauf für ein weiteres Bildpaar, wobei die Verarbeitungsobjekte "Median 1", "Median 2", "Difference" und "Comtal" im Zustand "CYCLE" arbeiten.

Die Abbildungen 6.63 und 6.64 zeigen den Abbruch des Systemablaufs mithilfe des Objektes "System".

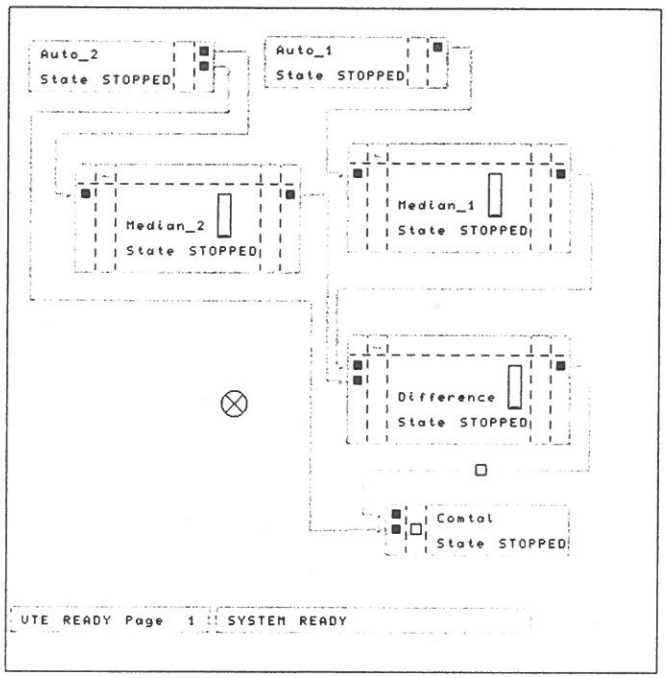

Abbildung 6.52: Datenfluß von "Difference" nach "Comtal"

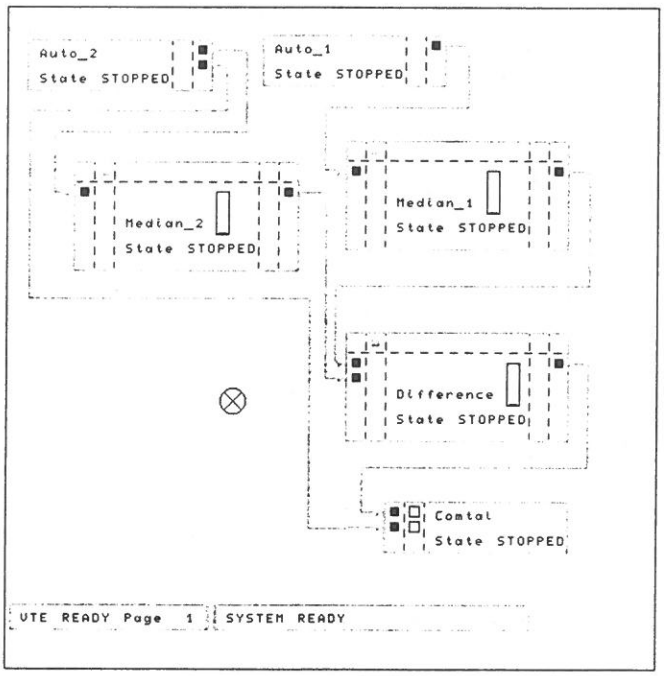

Abbildung 6.53: Beide Datenobjekte im Eingang von "Comtal"

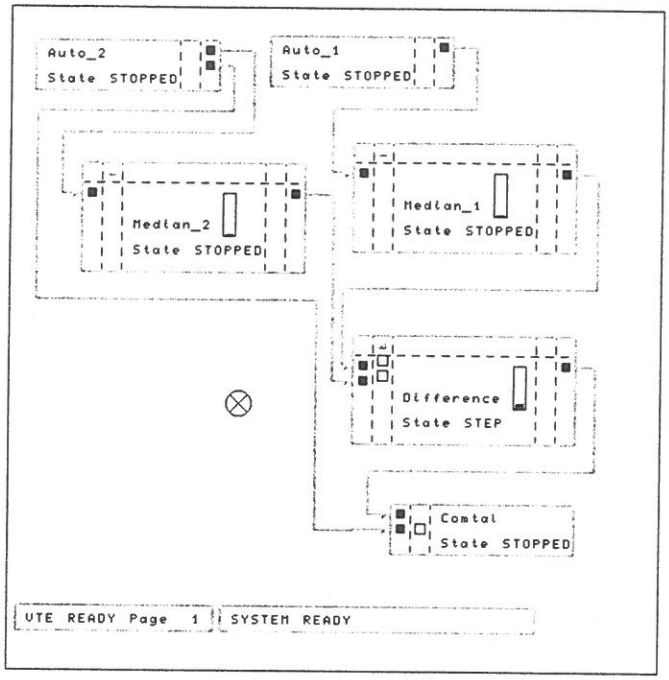

Abbildung 6.50: Verarbeitung der Datenobjekte durch "Difference"

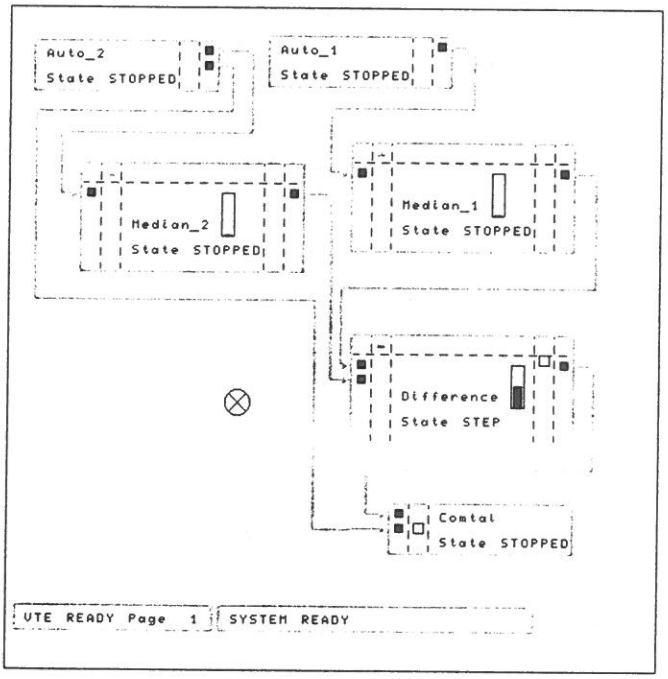

Abbildung 6.51: Erzeugung der Änderungsmaske

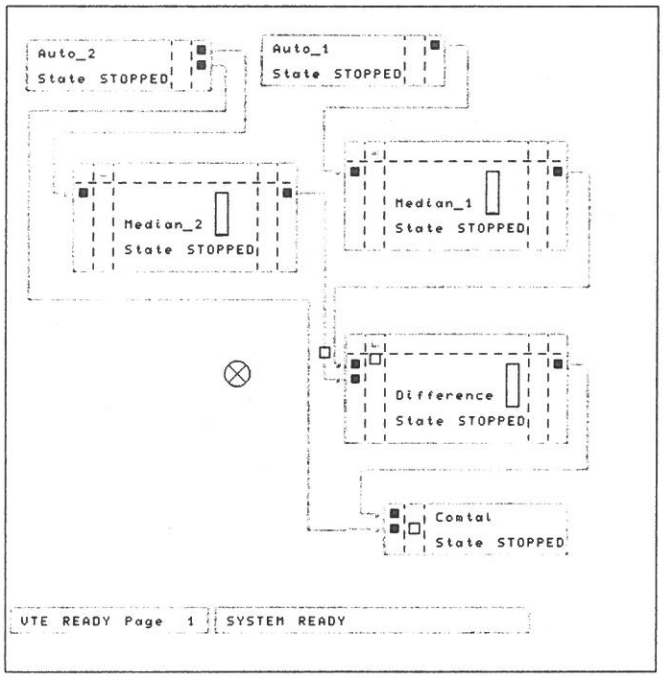

Abbildung 6.48: Datenfluß von "Median 2" nach "Difference"

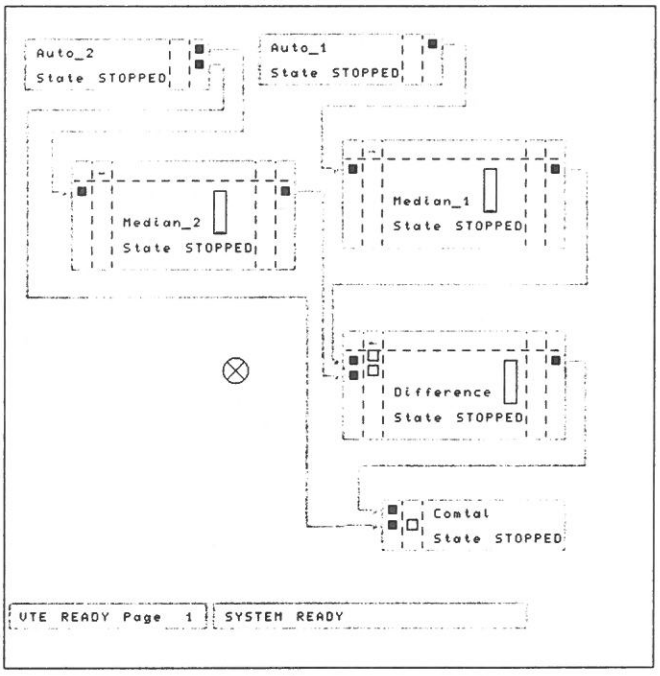

Abbildung 6.49: Datenobjekt im zweiten Eingang von "Difference"

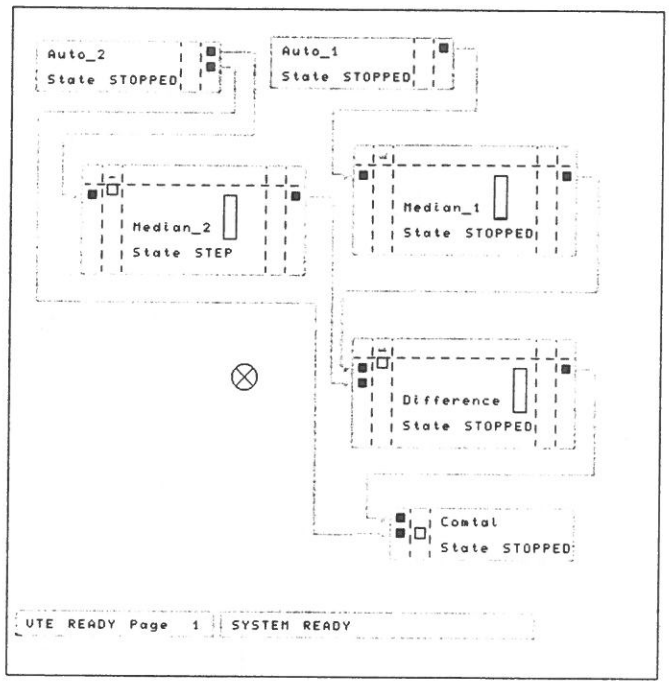

Abbildung 6.46: Das erste Datenobjekt im ersten Eingang von "Difference"

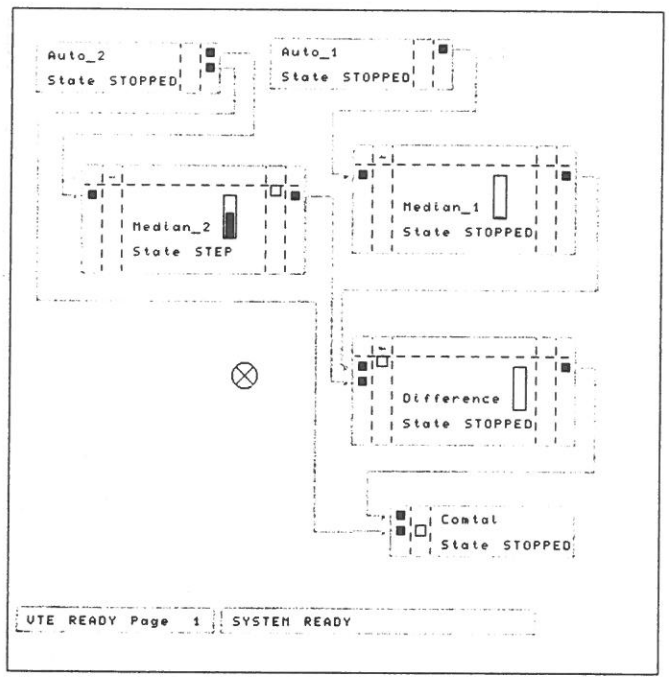

Abbildung 6.47: Verarbeitung eines Datenobjektes durch "Median 2"

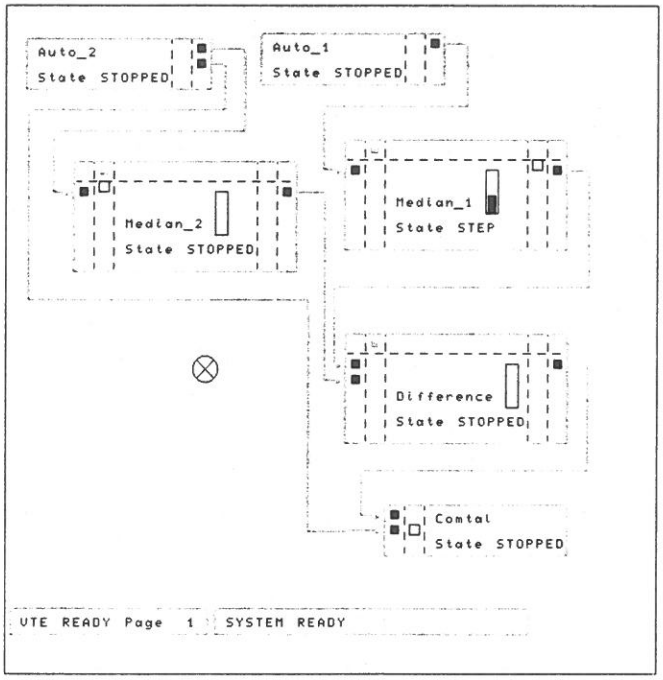

Abbildung 6.44: Verarbeitung eines Datenobjektes durch "Median 1"

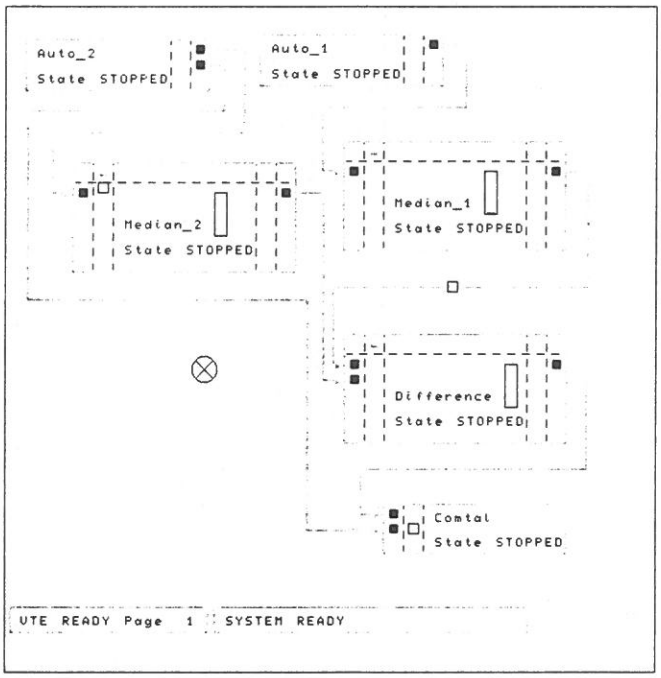

Abbildung 6.45: Datenfluß von "Median 1" nach "Difference"

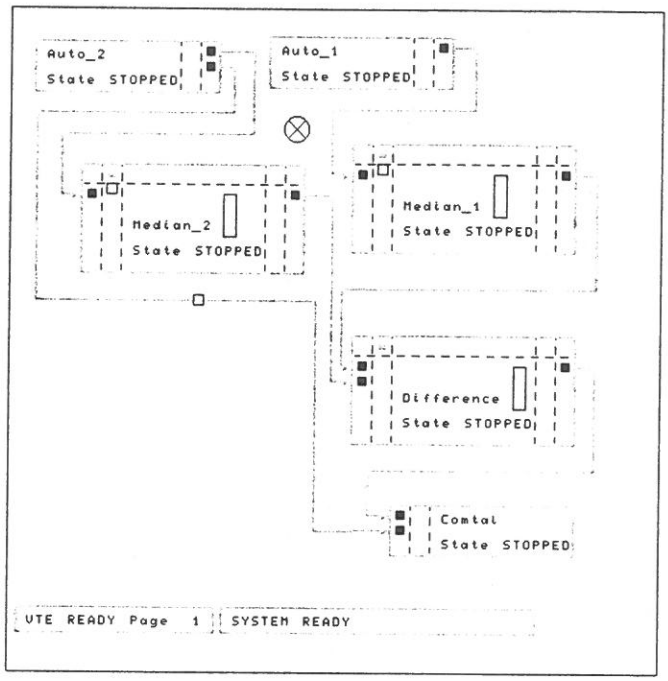

Abbildung 6.42: Das erste Datenobjekt im Eingang von "Median 2"

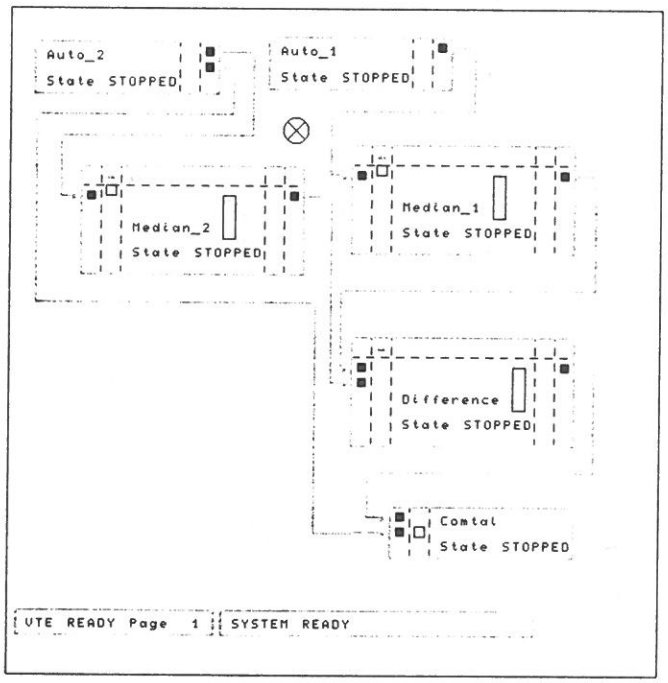

Abbildung 6.43: Das zweite Datenobjekt im zweiten Eingang von "Comtal"

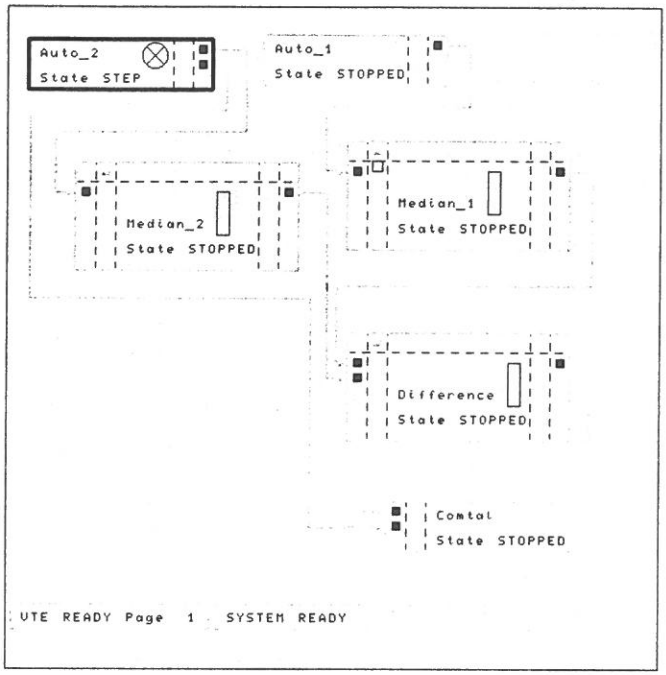

Abbildung 6.40: Setzen des Parameterwertes "STEP"

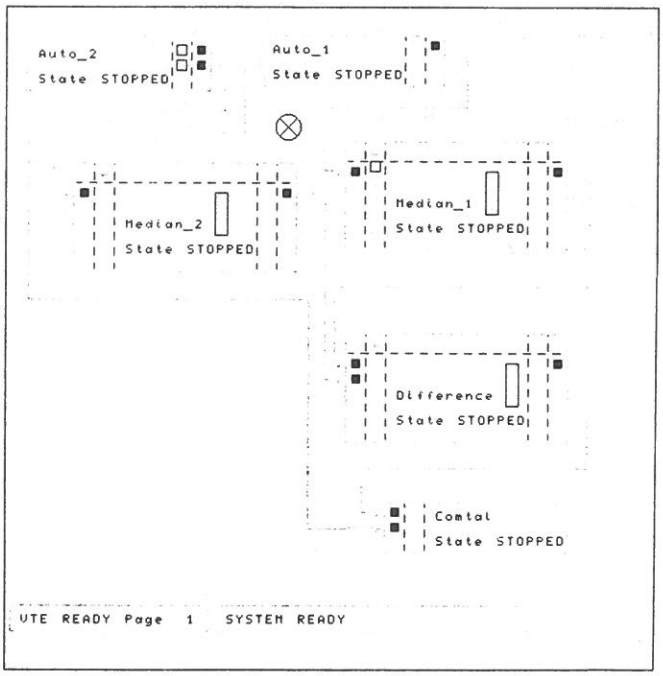

Abbildung 6.41: Datenobjekte im Ausgang von "Auto 2"

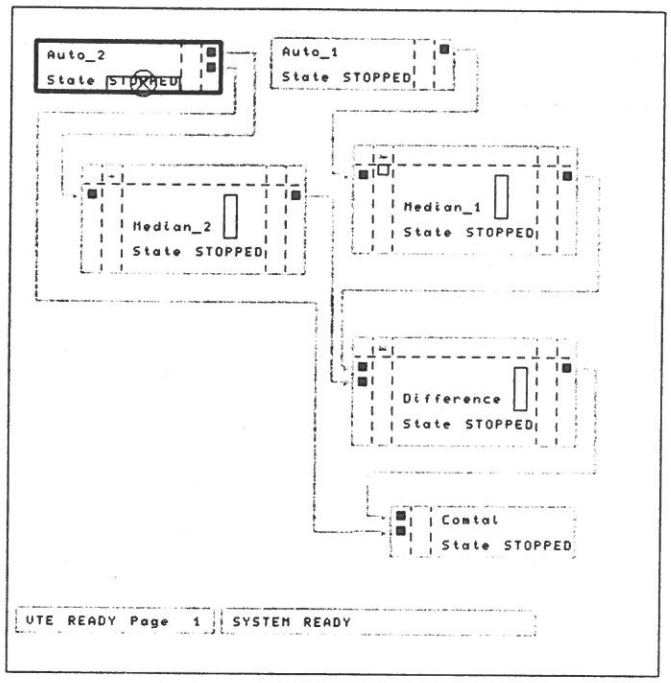

Abbildung 6.39: Auswahl des Parameters "State"

Die Abbildungen 6.39 bis 6.43 stellen das Erzeugen von Datenobjekten mit "Auto 2" dar. Die Abbildungen 6.44 bis 6.46 zeigen die Verarbeitung des im Eingang von "Median 1" dargestellten Datenobjektes und die anschließende Übertragung des Ergebnisobjektes zu "Difference".

Derselbe Vorgang mit dem Objekt "Median 2" wird in den Abbildungen 6.46 bis 6.49 dargestellt. Die Abbildungen 6.50 bis 6.54 zeigen die weitere Verarbeitung der Datenobjekte durch "Difference" und "Comtal".

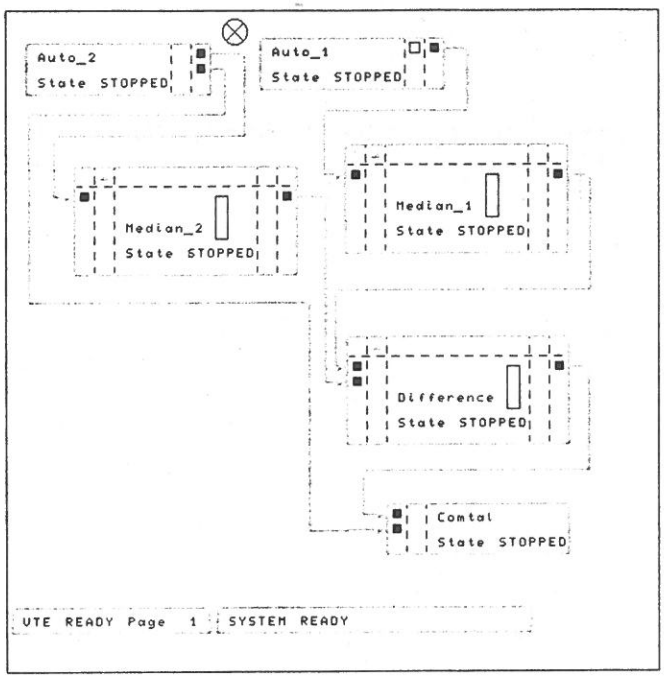

Abbildung 6.37: Datenobjekt im Ausgang von "Auto 1"

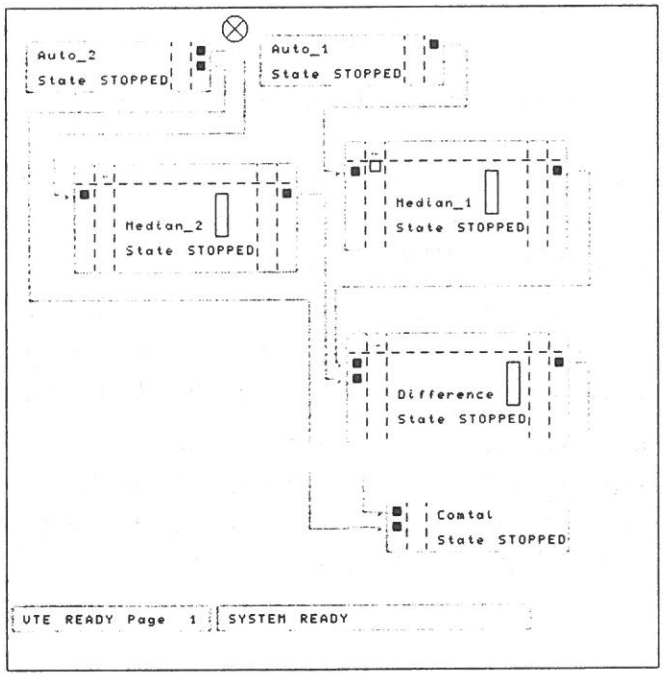

Abbildung 6.38: Datenobjekt im Eingang von "Median 1"

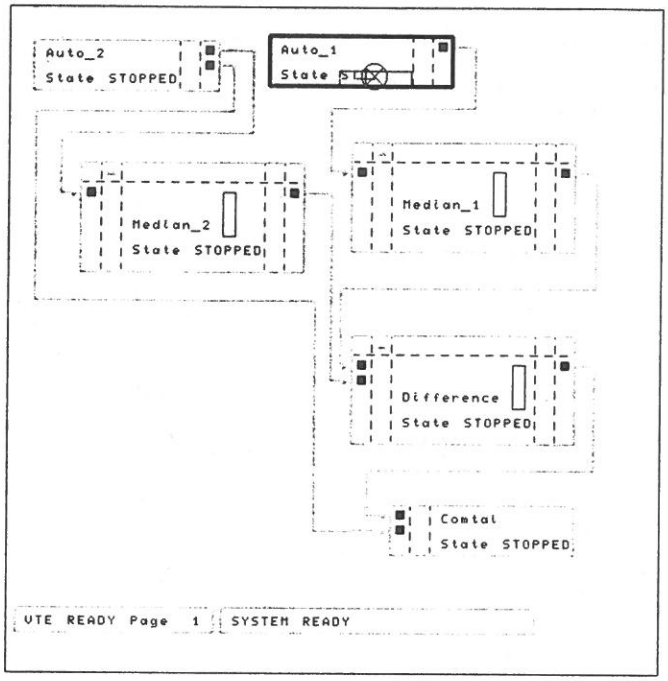

Abbildung 6.36: Eingabe des Parameterwertes "STEP"

Für die hier vorgestellte Simulation eines Anwendungssystems erhielten alle dargestellten Objekte (mit Ausnahme der Leitungen) einen Parameter zur Steuerung dieser Objekte. Besitzt dieser mit "State" bezeichnete Parameter eines Objektes den Wert "STOPPED", so führt dieses keine Verarbeitung durch. Wählt der Benutzer den Wert "STEP", so wird der Verarbeitungszyklus genau einmal durchlaufen und der Parameter erhält wieder den Wert "STOPPED". Der Parameterwert "CYCLE" bezeichnet die ständige Verarbeitungsbereitschaft eines Objektes.

Abbildung 6.36 stellt die Eingabe des Wertes "STEP" mittels der Tastatur dar. Abbildung 6.37 zeigt im Objekt "Auto 1" ein Datenobjekt vor dem Leitungsausgang (als weißes nicht ausgefülltes Quadrat).

Sobald dieses Datenobjekt zum Objekt "Median 1" übertragen wird, deutet sein Darstellungsobjekt dieses Vorgang dadurch an, daß es seine Darstellung entlang der Leitung wandern läßt. Abbildung 6.38 zeigt das Datenobjekt bereits im Eingangsbereich von "Median 1".

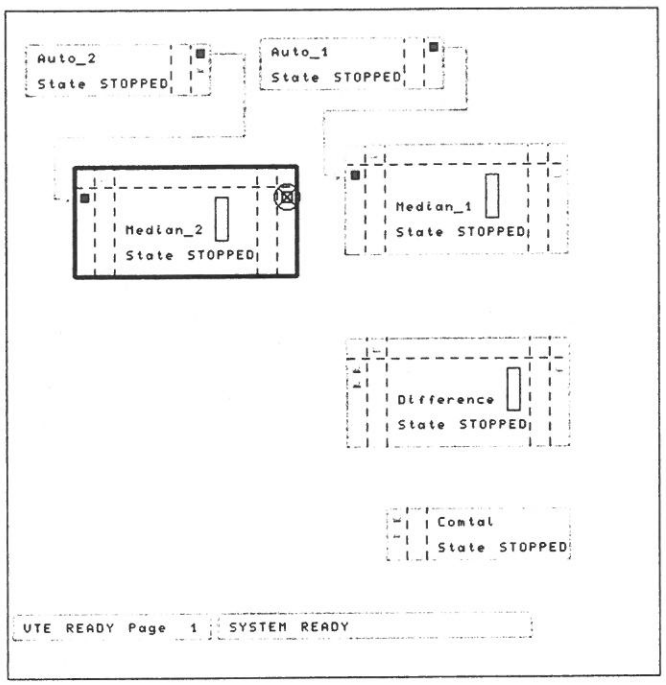

Abbildung 6.34: Ziehen von weiteren Leitungen (2)

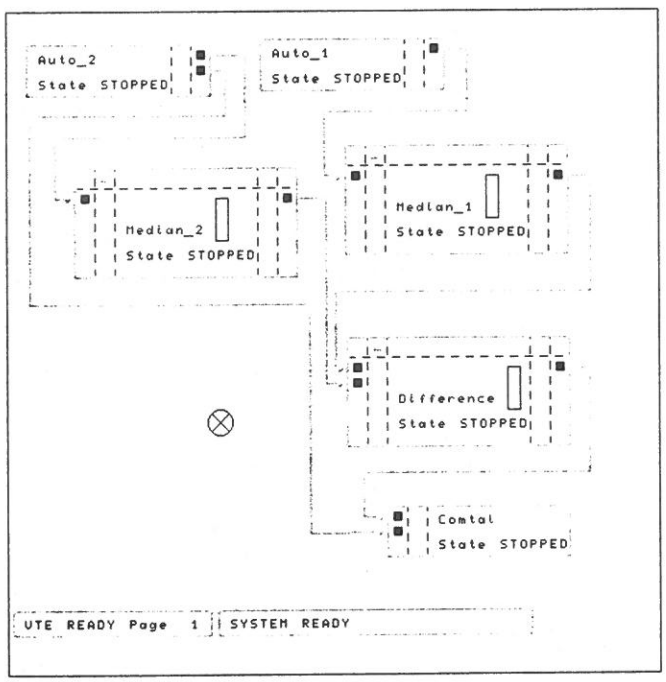

Abbildung 6.35: Ziehen von weiteren Leitungen (3)

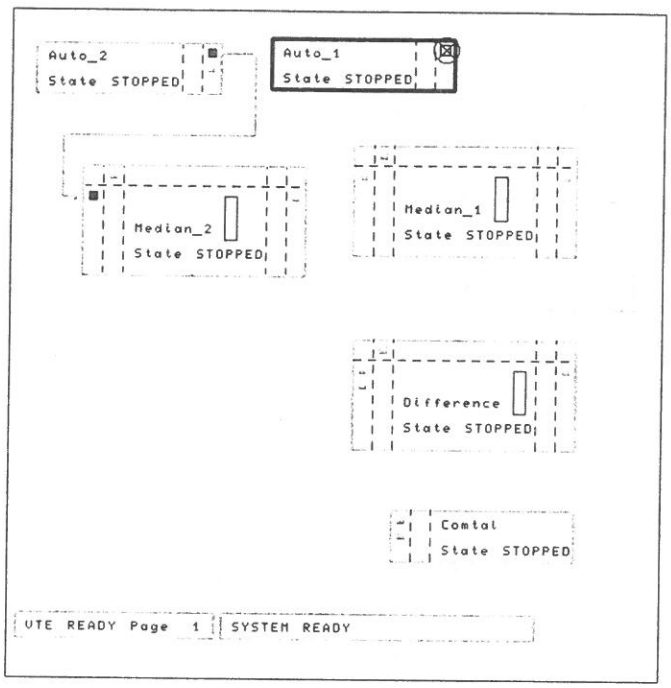

Abbildung 6.33: Ziehen von weiteren Leitungen (1)

hören dann auf, ihre Leitungseingänge durch Blinken zu markieren. Sobald die Maus den Bereich von "Median 2" verlassen hat, erhält das Leitungsobjekt seine endgültige Darstellung (siehe Abbildung 6.33).

Das Legen einer zweiten Datenleitung vom Ausgang des Objektes "Auto 1" zum Eingang des Objektes "Median 1" ist in den Abbildungen 6.33 und 6.34 zu sehen.

Abbildung 6.35 zeigt die Systemdarstellung, nachdem der Ausgang von "Median 1" bzw. "Median 2" mit dem ersten bzw. zweiten Eingang von "Difference", der Ausgang von "Difference" mit dem ersten Eingang von "Comtal" und der zweite Ausgang von "Auto 2" mit dem zweiten Eingang von "Comtal" verbunden wurde.

Die Datenobjekte und ihr Fluß durch die Leitungen werden ebenfalls graphisch dargestellt. Die Verarbeitungsobjekte besitzen in ihrer graphischen Darstellung rechts neben ihren Leitungseingängen bzw. links vor ihren Leitungsausgängen einen durch zwei senkrechte, gestrichelte Linien abgetrennten Bereich. Dieser Bereich dient der Darstellung der Datenobjekte, die das Verarbeitungsobjekt gerade über seinen Leitungseingang empfangen hat oder die es nach ihrer Bearbeitung demnächst über einen Ausgang absenden will.

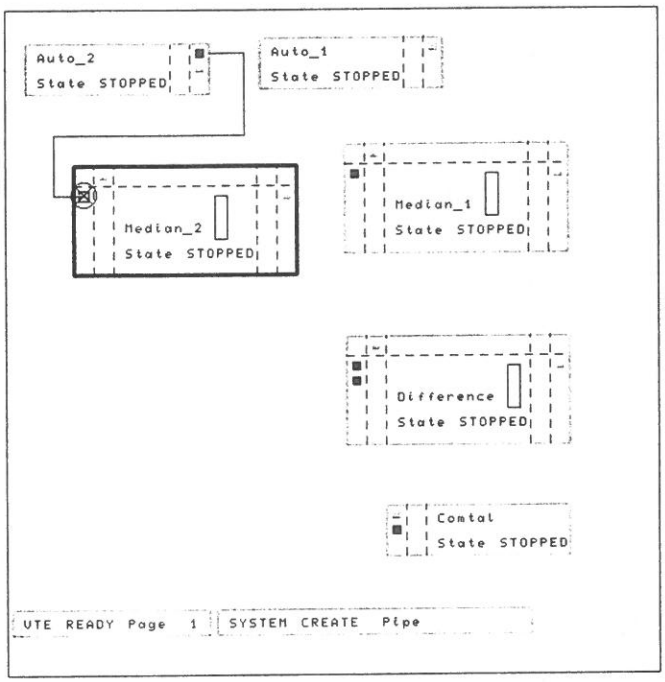

Abbildung 6.32: Auswahl eines Eingangs

wird. Die Darstellungsobjekte gestatten mithilfe der von ihren Anwendungsobjekten erhaltenen Information über den Typ der Leitungen und Anschlüsse nur eine Verbindung zwischen unbelegten, kompatiblen Ein- und Ausgängen. Jedes Darstellungsobjekt, das einen derartigen freien Eingang besitzt, signalisiert dies dem Benutzer durch das Blinken des zugeordneten Quadrats.

Sobald die Maus den Objektbereich verläßt, folgt ihr eine temporäre Leitungsdarstellung (siehe Abbildung 6.32). Diese Leitungsdarstellung wird von dem Darstellungsobjekt erzeugt, das dem Leitungsobjekt des Anwendungssystems zugeordnet ist. Die Maus wird nicht exklusiv reserviert, da sonst kein Eingang ausgewählt werden könnte. Die Farbe der Mausmarke wird von weiß nach weiß-rot blinkend geändert.

In Abbildung 6.33 hat das Objekt "Median 2" diese Leitung für einen seiner Eingänge akzeptiert und die Mausmarke die Farbe Grün erhalten. Dies äußert sich innerhalb des Dialogsystems dadurch, daß das Darstellungsobjekt von "Median 2" an das Darstellungsobjekt der Datenleitung die Identität seines ausgewählten Leitungseingangs sendet. Daraufhin teilt das Darstellungsobjekt der Datenleitung als Rundspruch allen anderen Darstellungsobjekten mit, daß der Benutzer einen Eingang ausgewählt hat. Diese

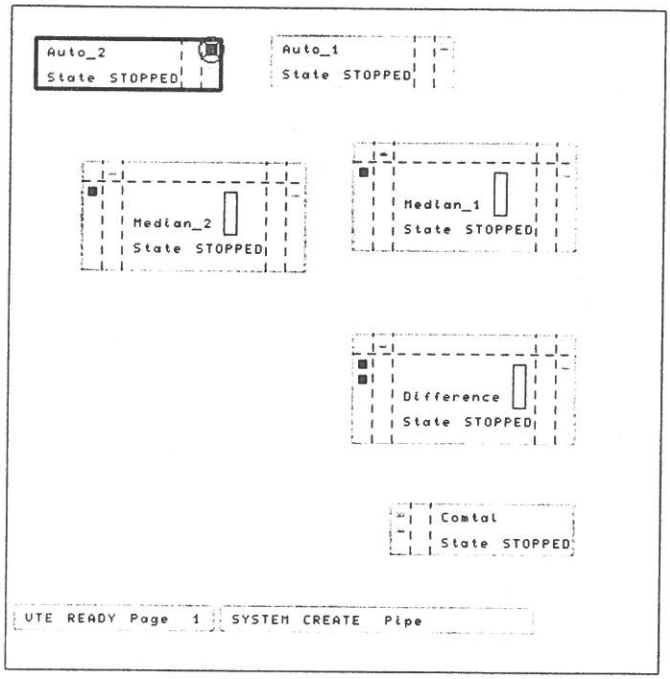

Abbildung 6.31: Erzeugen einer Leitung

Die weiteren Abbildungen demonstrieren das Legen von Datenleitungen zwischen den Objekten. Dies erreicht der Benutzer dadurch, daß er beispielsweise beim Objekt "Auto 2" den als weißes[11] Quadrat dargestellten Leitungsausgang auswählt (siehe Abbildung 6.30) und durch einen Knopfdruck ein entsprechendes Leitungsobjekt erzeugt (siehe Abbildung 6.31). Der ausgewählte Ausgang darf noch nicht belegt sein.

Im Dialogsystem finden dabei eine Reihe von Aktionen statt. Nach der Betätigung der Maus-Taste teilt das Darstellungsobjekt seinem Interaktionsobjekt die Identität des ausgewählten Leitungsausgangs mit. Das Interaktionsobjekt meldet diese Identität an sein Anwendungsobjekt weiter. Das Anwendungsobjekt erzeugt daraufhin ein neues Anwendungsobjekt, das die Funktion einer Datenleitung ausübt (siehe *Faasch 86*). Dieses neue Anwendungsobjekt meldet sich wie oben beschrieben bei dem Objekt "Objektanmeldung" an und erhält ein Interaktionsobjekt zugewiesen (siehe Abbildung 6.7).

Dieses Interaktionsobjekt erzeugt seinerseits ein eigenes Darstellungsobjekt. Dieses Darstellungsobjekt sendet als Rundspruch an alle anderen Darstellungsobjekte die Nachricht, daß für eine Leitung ein Eingang gesucht

[11] Hier grau

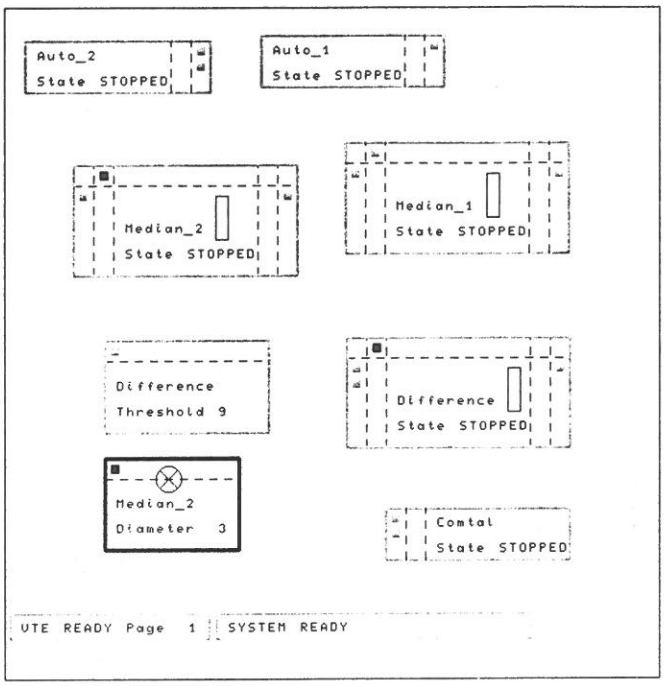

Abbildung 6.29: Entfernen von zweiten Darstellungen (2)

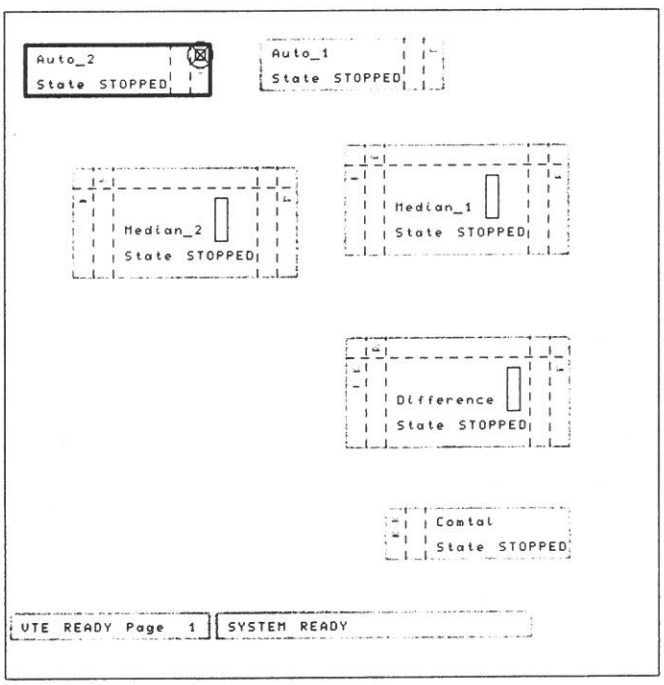

Abbildung 6.30: Auswahl eines Leitungsausgangs

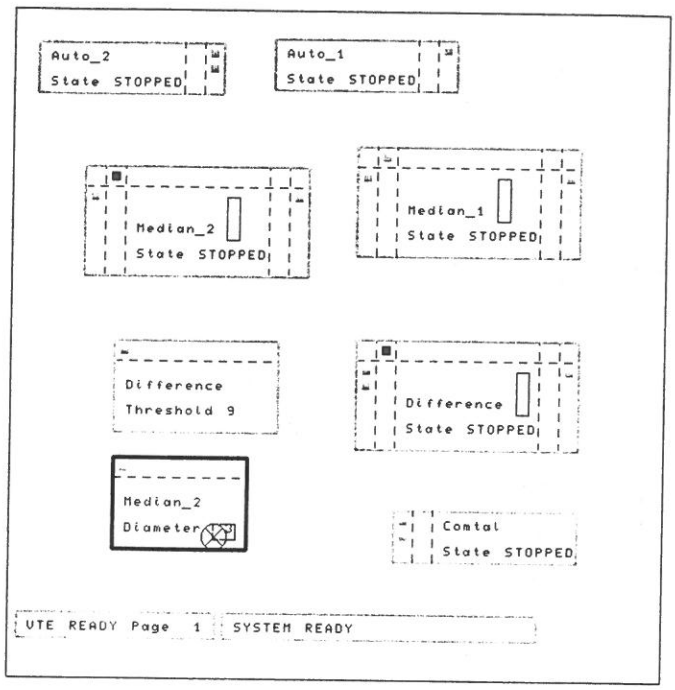

Abbildung 6.27: Tastendruck im Objekt "Median 2"

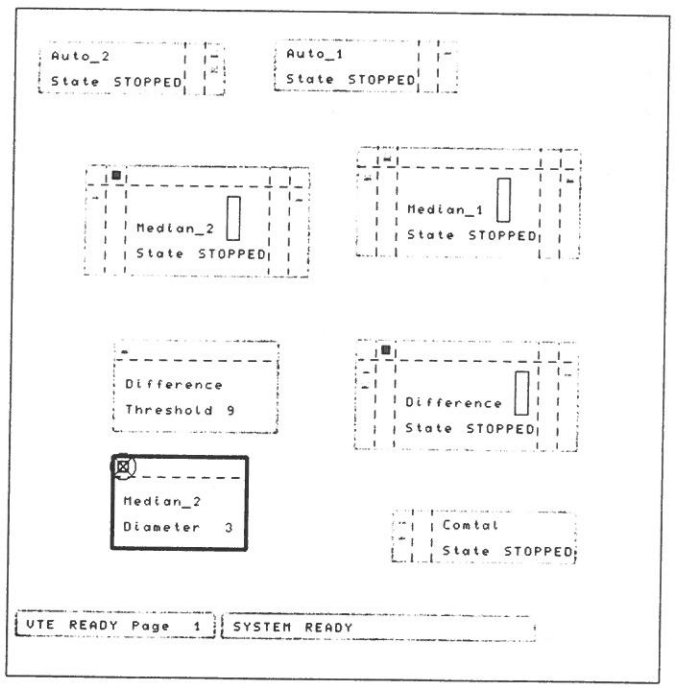

Abbildung 6.28: Entfernen von zweiten Darstellungen (1)

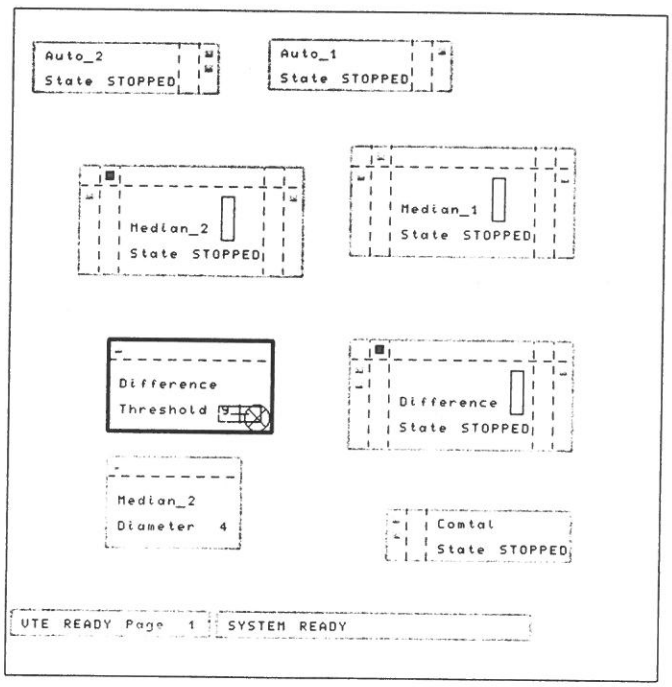

Abbildung 6.26: Änderung eines Parameters durch Zeicheneingabe

Der Benutzer kann die Parameterwerte entweder über die Tastatur oder über einen Tastendruck der Maus (zum Anzeigen des nächst höheren oder niedrigeren Wertes) eingeben. Bei der Eingabe von der Tastatur zeigt ein kleines hellblaues Fadenkreuz die nächste zu besetzende Textposition an (siehe Abbildung 6.26).

Abbildung 6.27 zeigt das Objekt "Difference" nach Abschluß der Interaktion und das Objekt "Median 2" während der Interaktion mit dem Benutzer.

Die zweiten Darstellungen der Objekte können durch das Betätigen ihres linken oberen Knopfes wieder vom Bildschirm gelöscht werden (siehe Abbildung 6.28 und 6.29). In Abbildung 6.30 wurden alle vollständigen Darstellungen entfernt.

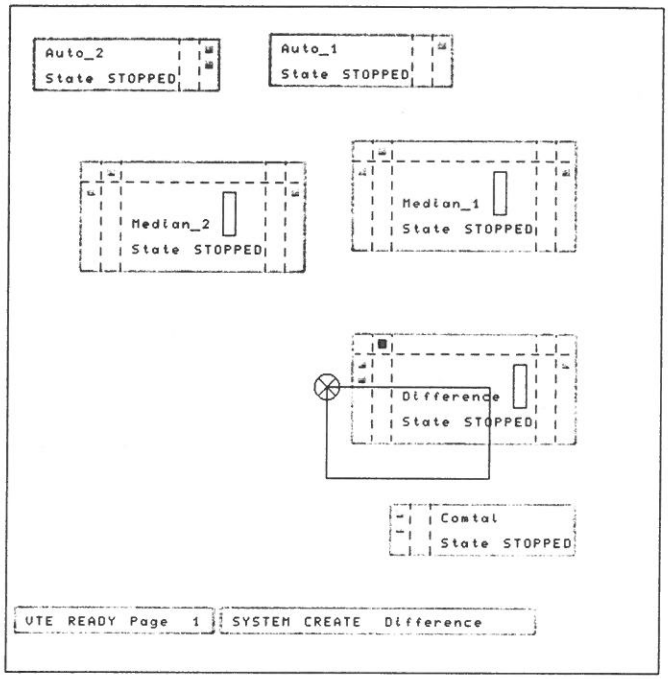

Abbildung 6.24: Erzeugung einer zweiten Darstellung (1)

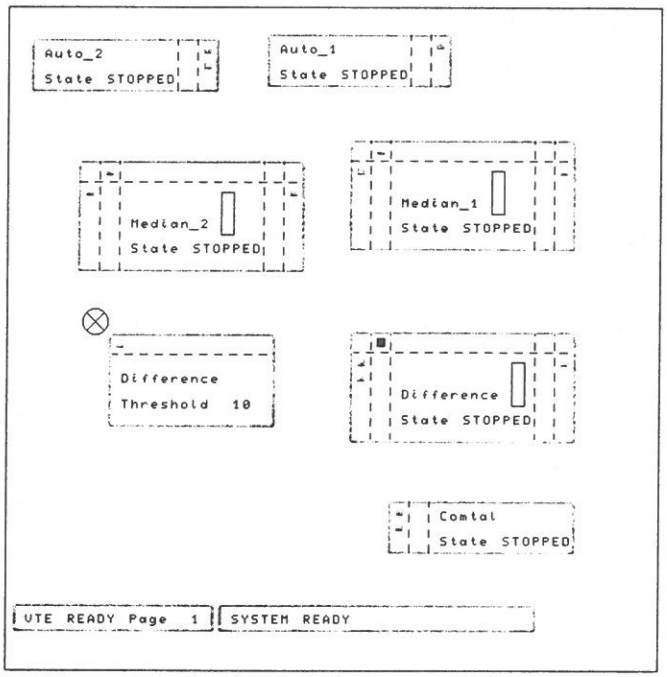

Abbildung 6.25: Erzeugung einer zweiten Darstellung (2)

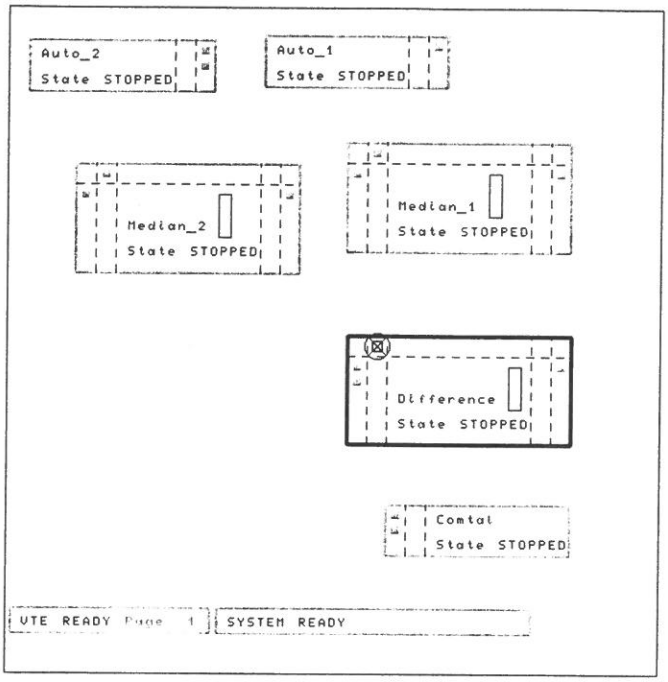

Abbildung 6.23: Auswahl des "Vergrößerungsknopfes"

signalisiert durch eine hellgrüne Umrandung) und durch einen Tastendruck betätigen. Daraufhin erhält der Knopf eine hellrote[9] Farbe, das Objekt "System" empfängt von dem neuen Darstellungsobjekt eine Meldung und die Farbe der Mausmarke wechselt nach rot-weiß blinkend, da das neue Darstellungsobjekt an der momentanen Position der Marke nicht alloziert werden darf (siehe Abbildung 6.24).

Der Benutzer muß diesem Darstellungsobjekt nun ebenfalls — wie schon oben beschrieben — eine Position auf dem Bildschirm zuweisen (siehe Abbildung 6.25). Abbildung 6.26 zeigt die Systemdarstellung, nachdem für die Objekte "Difference" und "Median 2" eine zweite Darstellung erzeugt wurde. Diese Darstellungen besitzen einen virtuellen Knopf, der ganz links oben als weißes[10] ausgefülltes Quadrat dargestellt wird. Mit diesem Knopf kann eine derartige Darstellung wieder gelöscht werden.

In Abbildung 6.26 wurde im Objekt "Difference" der Wert des Parameters "Threshold" ausgewählt (hellgrün umrandet). Die Voreinstellung dieses Parameters ist der Wert "10". Der Typ der veränderbaren Parameter kann z.Z. skalar, ganzzahlig, reellwertig oder eine Zeichenkette sein.

[9]Hier von grau nach schwarz

[10]Hier grau

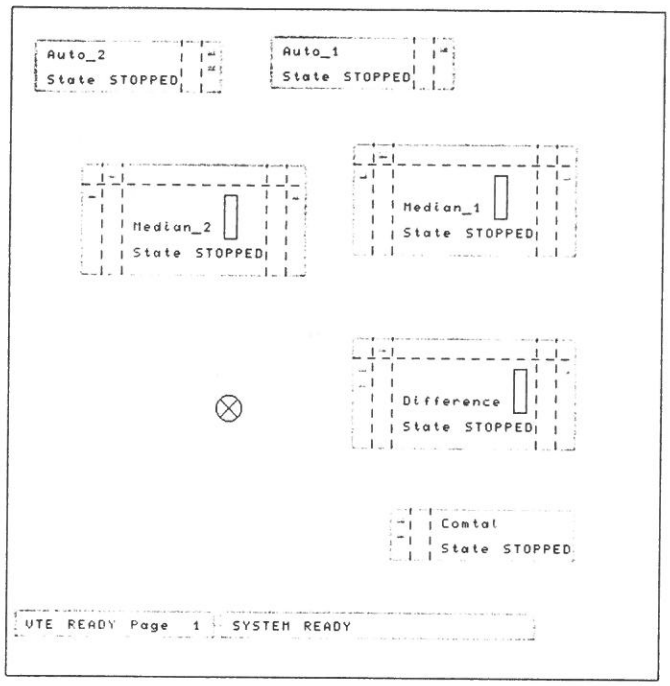

Abbildung 6.22: Endgültige Position aller Objekte

Leitungseingänge werden als farbig[6] ausgefüllte Quadrate immer am linken Rand, die Leitungsausgänge immer am rechten Rand dargestellt (abgetrennt durch eine senkrechte, gestrichelte blaue Linie). Das Objekt "Difference" besitzt beispielsweise zwei Eingänge und einen Ausgang. Am oberen Fensterrand werden virtuelle Knöpfe dargestellt (ebenfalls durch eine waagerechte, gestrichelte blaue Linie abgetrennt).

Sobald der Benutzer die Maus in den Bereich eines Objektes bewegt, zeigt dieses seine Interaktionsbereitschaft dadurch an, daß es die Farbe seiner Fensterumrandung von Dunkelrot[7] nach Hellgrün[8] ändert (siehe Abbildung 6.23). Gleichzeitig hat das Darstellungsobjekt mithilfe seines Eingabe-Objektes das Maus- und das Tastatur-Objekt exklusiv für sich reserviert. Alle nachfolgenden Eingaben des Benutzers werden von diesem Objekt solange interpretiert, bis die Maus das Objektfenster wieder verläßt.

Neben einer kompakten Darstellung können die Anwendungsobjekte auch eine zweite Darstellung mit weiteren Parametern besitzen. Der Benutzer muß dafür den linken oberen Knopf auswählen (vorläufige Auswahl

[6]Hier immer schwarz

[7]Hier grau

[8]Hier schwarz

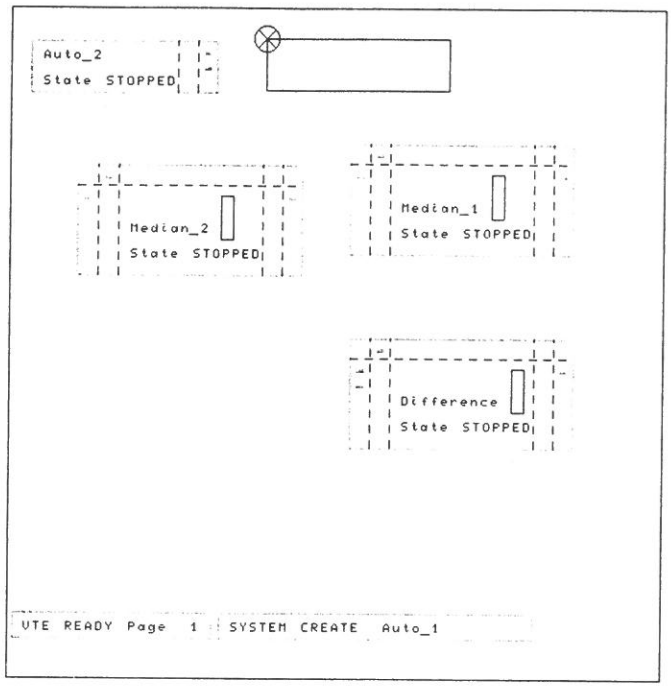

Abbildung 6.20: Erzeugung von "Auto 1"

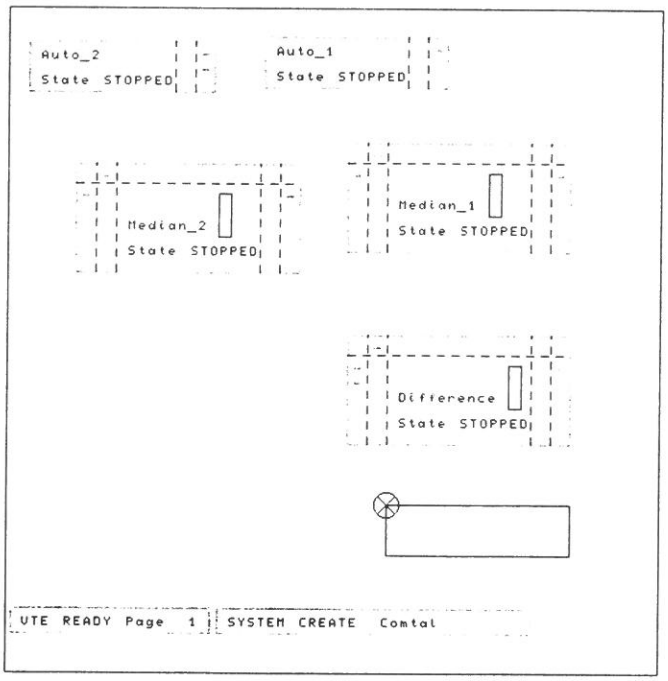

Abbildung 6.21: Erzeugung von "Comtal"

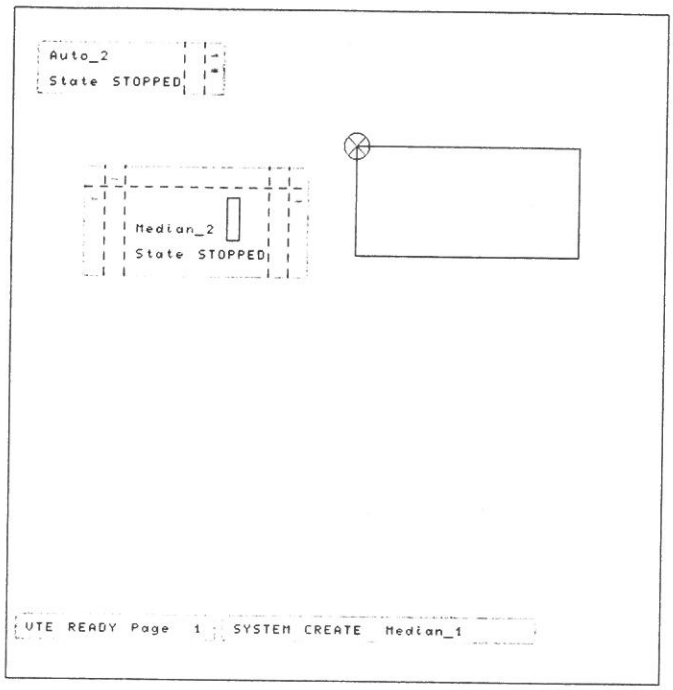

Abbildung 6.18: Erzeugung von "Median 1"

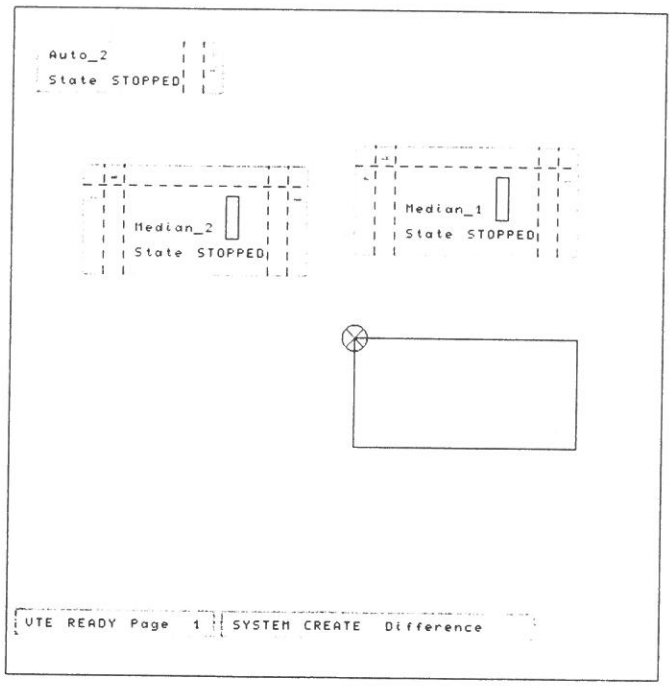

Abbildung 6.19: Erzeugung von "Difference"

siert er dies durch einen einfachen Tastendruck mit der Maus. Daraufhin wird eine kompakte Darstellung des Objektes erzeugt (siehe Abbildung 6.17).

Um ein Überlappen von Bildschirmfenstern zu verhindern, kommuniziert das Darstellungsobjekt mit allen anderen im System vorhandenen Darstellungsobjekten. Da alle Darstellungsobjekte Mitglieder einer "einfachen Gruppe" (siehe Kapitel 6.2.7) sind, kann das noch nicht positionierte Darstellungsobjekt als Rundspruch seine momentane Bildschirmposition und den Umfang seines Rechtecks an seine Gruppenmitglieder versenden. Diese kontrollieren beim Empfang dieser Nachricht, ob in der übersandten Position eine Überlappung mit ihrem eigenen Fenster besteht. Stellt ein Objekt eine Überlappung fest, so teilt es dies dem Absender des Rundspruchs mit. Als Folge dieser Kommunikation wird die Farbe[5] der Mausmarke in Rot (als Ablehnung dieser Position) oder in Grün (als Zustimmung) geändert. Das zu positionierende Darstellungsobjekt akzeptiert eine Position nur dann, wenn kein Mitglied seiner Gruppe eine Überlappung feststellt.

Damit der Benutzer bei der Auswahl einer Bildschirmposition mit der Mausmarke auch in maussensitive Bereiche von anderen Darstellungsobjekten eintreten kann, ohne daß diese aktiviert werden, reserviert das zu positionierende Darstellungsobjekt mithilfe seines Eingabe-Objektes exklusiv das Mausobjekt. Die Freigabe des Maus-Objektes erfolgt, sobald dem Darstellungsobjekt seine endgültige Bildschirmposition zugewiesen wurde.

Die Abbildungen 6.18 bis 6.22 zeigen die Erzeugung der oben beschriebenen sechs Objekte, wobei Abbildung 6.22 die endgültige Position aller Objekte anzeigt.

Die Darstellung der Objekte besteht prinzipiell aus einem farbig umrandeten Fenster und einem darin enthaltenen Objektnamen. In Abhängigkeit von der durch die Anwendungsobjekte übersandten Objektdefinition können diese Fenster noch weitere Darstellungen enthalten.

Objekte können den Fortschritt ihrer Verarbeitung prozentual anzeigen. Die Darstellungsobjekte zeigen den Fortschritt durch eine wachsende Säule an (z.B. bei den Objekten "Median 1", "Median 2" und "Difference" rechts neben dem Objektnamen). Wie wichtig gerade eine Rückmeldung eines Programmsystems über die noch verbleibenden Verarbeitungsschritte für den Benutzer ist, hat eine in *Myers 85* beschriebene Untersuchung gezeigt.

Weiterhin können Objekte Anschlüsse für Datenleitungen besitzen. Die

[5]Hier immer schwarz

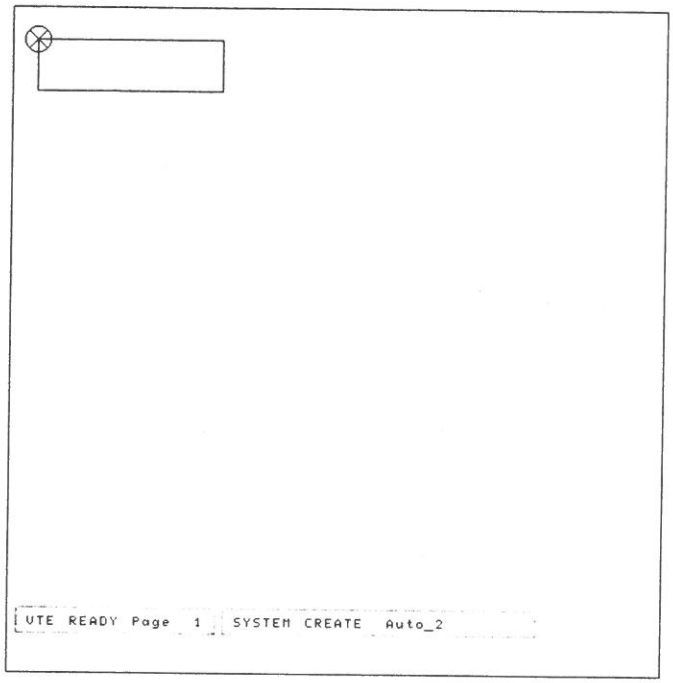

Abbildung 6.16: Endgültige Position von "Auto 2"

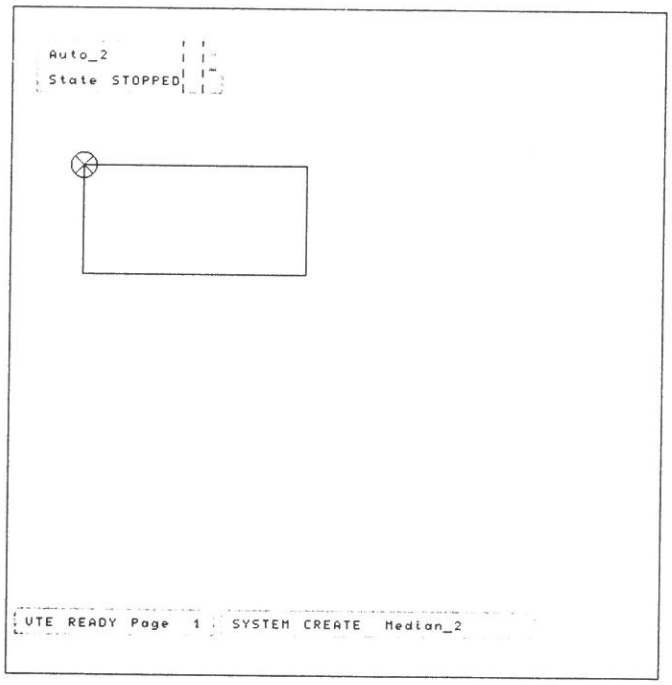

Abbildung 6.17: Erzeugung von "Median 2"

das Zustandssystem aufgerufen werden, während der zweite die für die Zustandsdarstellung zu verwendende Bildspeicherseite im Gerät VTE angibt. Das zweite Objekt (System) wird von anderen Interaktionsobjekten zur Ausgabe von Meldungen verwendet. Weiterhin besitzt es einen Parameter, mit dem der Benutzer die Ausführung des Dialogsystems beenden kann.

Das Objekt "System" definiert im Dialogsystem eine Sterngruppe. Dabei hat es die Funktion des Sternobjektes, während alle Interaktionsobjekte die anderen Gruppenmitglieder bilden (siehe auch Abbildung 6.13). Wählt der Benutzer den Parameterwert "STOP", so sendet das Objekt "System" an alle seine Gruppenmitglieder einen entsprechenden Rundspruch, damit diese ihre Tätigkeit im Dialogsystem ordnungsgemäß abschließen und beenden. Es sei an dieser Stelle noch einmal darauf hingewiesen, daß durch die objektorientierte Programmierung des Dialogsystems eine zentrale Dialogkontrolle nicht existiert.

Sobald ein Interaktionsobjekt nach seiner Erzeugung von seinem Anwendungsobjekt dessen Objektdefinition erhält (siehe Abbildung 6.7), erzeugt es ein Darstellungsobjekt, das seinerseits gemäß der Objekthierarchie (siehe Abbildung 6.10) ein Eingabe- und ein Ausgabe-Objekt erzeugt. Damit das Darstellungsobjekt eine Darstellung seines Anwendungsobjektes erzeugen kann, muß ihm auf dem Bildschirm ein ausreichender Platz angewiesen werden. Dafür sieht das Dialogsystem z.Z. zwei Alternativen vor. Entweder teilt das Anwendungsobjekt seinem Interaktionsobjekt in der Objektbeschreibung eine gewünschte Position mit oder es läßt diese unspezifiziert. Bei der Angabe der Position verläßt sich das Darstellungsobjekt darauf, daß an der gewünschten Stelle genügend Platz frei ist. Fehlt die Angabe der Position, so muß der Benutzer dem Darstellungsobjekt eine Position zuweisen. Gleichzeitig signalisiert das Darstellungsobjekt mithilfe des Objektes "System" seine Erzeugung unter Angabe seines Namens.

Um den Benutzer bei der Auswahl einer Bildschirmposition zu unterstützen, erzeugt das Darstellungsobjekt ein hellblaues[4] Rechteck, das die aus der Objektbeschreibung berechnete Größe seines Bildschirmfensters anzeigt. Die von der Maus bezeichnete Markenposition (als Fadenkreuz mit einem umschließenden Kreis) gibt dabei die linkere obere Ecke des Rechtecks an. Dieses Rechteck folgt den Bewegungen der Maus (siehe Abbildung 6.15, 6.16).

Sobald der Benutzer die gewünschte Position ausgewählt hat, signali-

[4]Hier schwarz

gen von bewegten Objekten einer Straßenverkehrsszene sind. Eine ähnliche Problemstellung findet sich beispielsweise im MORIO-System [*Dreschler 81*].

Die zur Durchführung dieser Aufgabe nachempfundene Konfiguration umfaßt sechs Anwendungsobjekte. Es handelt sich um zwei Objekte als Datenquellen, die das erste ("Auto 1") bzw. das zweite Bild ("Auto 2") eines Bildpaars bereitstellen, um drei Verarbeitungsobjekte, wobei zwei eine Medianfilterung auf einem Bild durchführen ("Median 1", "Median 2") und das dritte elementweise die betragsmäßige Differenz zweier Bilder mit einer anschließenden Binarisierung anhand eines Schwellwertes vornimmt ("Difference"), und um ein Objekt als Datensenke, das ein Rastersichtgerät zur überlagerten Darstellung eines Bildes und einer binären Maske der Änderungsbereiche repräsentiert. Eine ausführliche Erörterung solcher Objekte und ihrer Rolle im Anwendungssystem ist in *Faasch 86* zu finden.

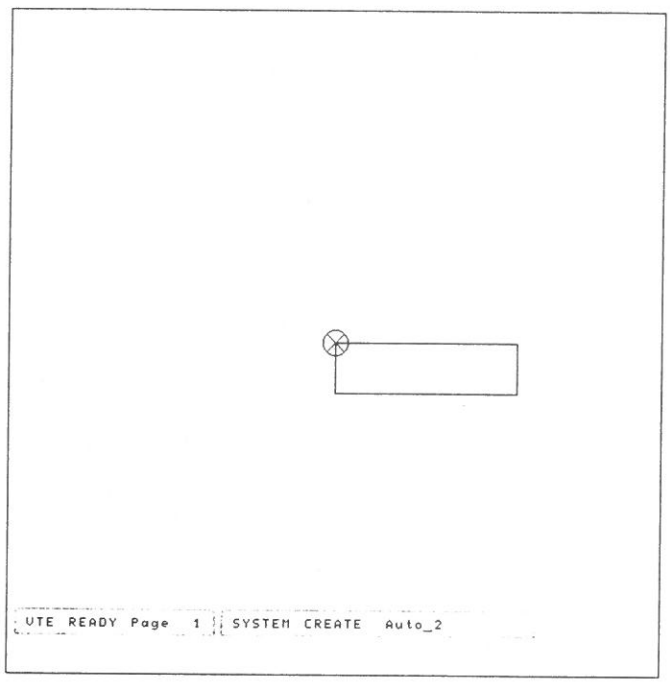

Abbildung 6.15: Erzeugung von "Auto 2"

Das Dialogsystem enthält in seiner momentanen Konstellation zwei spezielle Interaktionsobjekte (siehe Abbildung 6.15). Das eine Objekt (VTE) repräsentiert das VTE-Zustandssystem und wird in der linken unteren Ecke des Bildschirms dargestellt. Es besitzt als Schnittstelle zum Benutzer zwei veränderbare Parameter. Mit dem Ändern des ersten Parameters kann

wird als Ausgabegerät ein Graphikspeicher mit 808 × 1024 Rasterpunkten verwendet [*Lipkie et al. 82*].

Die zweite große Einschränkung ist in der Verwendung des GKS zu sehen. Im vorherigen Abschnitt wurde das ungenügende Abstraktionsniveau der GKS-Operationen aufgezeigt. Für diesen Prototyp war es deshalb mit einem vertretbaren Aufwand nicht möglich, Graphiken mit einer Qualität und Ausdrucksfähigkeit wie beispielsweise beim Xerox Star oder MacIntosh-System zu erzeugen. Weiterhin kommt erschwerend hinzu, daß die verfügbare GKS-Implementation bei der Texterzeugung nur den im VTE integrierten Textgenerator unterstützt. Dieser Textgenerator gestattet die Zeichenerzeugung nur an festgelegten Punkten. Diese Punkte definieren bei der kleinsten Schriftart ein Raster aus 22 Zeilen und 64 Spalten zur Texterzeugung.[2]

Als Ausgleich für diese Einschränkungen steht mit dem VTE ein Rastergraphikgerät zur Verfügung, das die Erzeugung von echten Farbgraphiken (RGB-Darstellung) erlaubt. Diese Möglichkeit wurde bei diesem Prototyp ausgiebig verwendet, um für den Benutzer verständlichere Graphiken zu erzeugen.[3]

Nachfolgend wird die durch den Prototyp realisierte Systemdarstellung näher erläutert. Gleichzeitig wird das Zusammenspiel der einzelnen Objekte des Dialogsystems zur Durchführung ihrer Aufgaben beschrieben. Dieser Prototyp beruht darauf, daß Anwendungssysteme über die vorhandene Anwendungsschnittstelle mit ihm kooperieren. Da ein System zur Konfiguration entsprechender Anwendungssysteme sich noch in der Entwicklung befindet (siehe *Faasch 86*), kann die graphische Systemdarstellung nur durch die teilweise Simulation eines Anwendungssystems demonstriert werden. Als Anwendungsbeispiel wurde das Problem der Erkennung von Änderungsbereichen zwischen jeweils einem Bildpaar aus einer Bildfolge — bestehend aus digitisierten TV-Bildern — gewählt. Unter Änderungsbereichen werden dabei Bildbereiche verstanden, die Abbildun-

[2]Diese Einschränkung wurde bei der Weiterentwicklung des Prototyps teilweise aufgehoben.

[3]Da eine Reproduktion dieser Farbgraphiken im Rahmen dieses Berichtes nicht zur Verfügung stand, werden diese Abbildungen nachfolgend als Schwarzweiß-Graphiken dargestellt. Dabei gilt, daß alle farbigen Linien jetzt als schwarze oder graue Linien auf weißem Hintergrund dargestellt werden. Um dem Leser trotzdem ein Bild von der Verwendung der Farbe vermitteln zu können, wurde die ursprüngliche Beschreibung der Graphiken beibehalten und nur — falls nötig — zum besseren Verständnis in Form von Fußnoten ergänzt.

VAX-11 Rechner unter dem Betriebssystem VMS übersetzt und auf den Kleinrechnern Dietz-Mincal-621×2 zur Ausführung gebracht.

Im Rahmen dieser Arbeit wurden diese Systeme erweitert, auf ein VAX-11/VMS System portiert und mithilfe des dort zur Verfügung stehenden Ada-Übersetzers für VAX-11/VMS ausgeführt. Da aber die gesamte Video-Peripherie und damit auch das VTE noch über die Kleinrechner angesteuert werden, mußte eine Schnittstelle zwischen dem VAX-11 System und den Kleinrechnern geschaffen werden. Es wurde in Ada ein Programmsystem entwickelt, das die Verknüpfung der beiden Rechnersysteme auf der Basis einer bestehenden Rechnerkopplung über Ethernet realisiert. Mithilfe dieses aus Ada-Programmen ansprechbaren Systems kann die Video-Peripherie benutzt werden (siehe Abbildung 6.14).

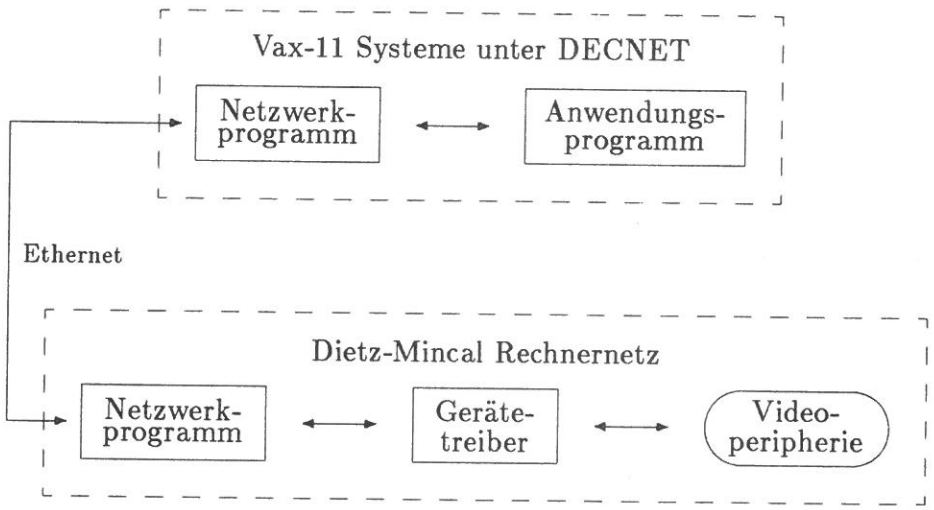

Abbildung 6.14: Rechnernetz

6.3.3 Graphische Systemdarstellung

Wie schon oben erwähnt, steht für diesen Prototyp als graphisches Ausgabegerät nur das VTE zur Verfügung. Das VTE besitzt für seine Graphik- und Bildspeicher eine Auflösung von 512 × 512 Rasterpunkten. Bei der Realisierung des Prototyps hat sich gezeigt, daß eine derartige Auflösung — wie schon vorweg vermutet — zur Erzeugung einer Systemdarstellung eine starke Einschränkung darstellt. Beim Xerox Star System beispielsweise

al. 85 werden derartige Techniken im Hinblick auf die graphische Darstellung von Programmen erörtert.

Eine solche Unterstützung ist in Graphikpaketen zu sehen, die eine umfangreiche Sammlung von Prozeduren zur graphischen Ein- und Ausgabe enthalten. Das Graphische Kernsystem (GKS) [*Enderle et al. 84*] kann als ein derartiges Werkzeug angesehen werden. GKS basiert überwiegend auf dem Modell der Vektor- und Liniengraphik. Es gibt zwar auch Möglichkeiten, die Eigenschaften von Rastergraphikgeräten zu nutzen (z.B. das Pixel-Feld), diese sind aber beschränkt und reichen beispielsweise nicht aus, das Graphikgerät VTE zu modellieren. In *Acquah et al. 82* wird ein konzeptuelles Modell vorgestellt, das sowohl die Behandlung der Vektorgraphik als auch der Rastergraphik einheitlich zuläßt. Ein anderer Vorschlag [*Stevens & Alexander 83*] entwickelt ein minimales Modell von Rastergeräten, um die Portabilität von Bildverarbeitungsprogammen zu erhöhen.

Das Abstraktionsniveau der vom GKS bereitgestellten Operationen ist jedoch für die Implementation eines Dialogsystems, das sich auf eine umfangreiche Verwendung von hierarchischen Bildschirmfenstern und bildhaften Symbolen in Form von Piktogrammen stützt, kaum ausreichend (siehe auch *Rosenthal 83*). In *Myers 84* wird ein Fenstersystem SAPPHIRE beschrieben, das die für diese Implementation nötigen Abstraktionen zur Verfügung stellen könnte. Für die Realisierung dieses Prototyps stand ein derartiges System zwar nicht zur Verfügung, dafür aber eine Teilimplementation des GKS. Die Auswahl des GKS hat den Vorteil, daß die Portabilität durch die Verwendung eines international anerkannten Standards erhöht wird. Gleichzeitig verringert sich der Aufwand für die Graphikerzeugung durch die vom GKS angebotenen Möglichkeiten (Ausgabefunktionen wie Linien, Marken, Füllgebiete und Koordinatentransformationen).

Diese Teilimplementation des Graphischen Kernsystems (Ebene L0a) war in Hamburg eine der ersten Anwendungen von Ada und wurde im Rahmen von zwei Projekten in Zusammenarbeit mit Studenten entwickelt. Als Ausgabegerät wurde das Graphikgerät VTE gewählt, für das im Vorwege ein Treiberprogramm in Ada erstellt wurde. Parallel dazu wurde im Rahmen einer Studienarbeit [*Mansfeld 84*] ein Dialogsystem zur graphischen Zustandsdarstellung des VTE entwickelt. Diese Arbeit kann als ein erster Versuch angesehen werden, den Zustand eines komplexen Graphikgerätes interaktiv und in Form von graphischen Darstellungen für den Benutzer zugänglich zu machen. Alle diese Programmsysteme werden unter Verwendung eines in Hamburg entwickelten Kreuzübersetzers für Ada auf einem

DFG-Forschungsvorhabens wurde durch die Teilimplementation eines Ada-Übersetzers[1] für die Kleinrechner Dietz-Mincal 621×2 die Eignung der Programmiersprache Ada für die Bildverarbeitung untersucht [*Faasch et al. 85*].

Die Sprache Ada unterstützt alle oben aufgestellten Forderungen. Sie besitzt ein Prozeßkonzept (task), das die Formulierung von parallelen Prozessen erlaubt. Diese Prozesse kommunizieren über ein Rendevous-Konzept. Die Datenabstraktion kann mithilfe von Paketen (package) leicht formuliert werden. Zur Programmabstraktion stehen die generischen Programmeinheiten (Pakete und Unterprogramme) zur Verfügung. Mithilfe der privaten Typen und der Pakete können schutzwürdige Bereiche realisiert werden. Durch das Verwenden von Paketen für Schnittstellenspezifikationen und von Untereinheiten zur Programmstrukturierung ist eine methodische Programmentwicklung (top down, bottom up) möglich. Aus diesen Eigenschaften ergibt sich, daß ebenfalls die Formulierung von Objekten angemessen durchführbar ist.

Ein Vergleich zwischen Smalltalk-80 und Ada hat gezeigt, daß Ada nicht die Flexibilität von Smalltalk besitzt [*Nagl 84*]. Dafür wird in Ada ein größeres Maß an Sicherheit und eine gute Unterstützung der methodischen Programmentwicklung geboten. Ada ist deshalb ebenfalls wie Smalltalk sehr gut zur objektorientierten Programmierung geeignet [*Buzzard & Mudge 85, Booch 86*].

Die Forderung nach einer Projektverwaltung wird durch das für diesen Prototyp zur Verfügung stehende Ada-System weitgehend erfüllt, da das zum Ada-Übersetzer gehörende Bibliothekssystem ACS eine Verwaltung der Ada-Programmeinheiten mithilfe von Projektbibliotheken gut unterstützt.

6.3.2 Graphiksystem

Für die Realisierung eines graphischen Dialogsystems ist aber nicht nur die Wahl einer geeigneten Implementationssprache, sondern auch eine Unterstützung der Graphikerzeugung wichtig. Ein Überblick über Techniken zur Realisierung von graphischen Interaktionen, der auch ihre Auswirkungen auf den Benutzer diskutiert, findet sich in *Foley et al. 84*. In *Brown et*

[1] Dieser Übersetzer entstand in Zusammenarbeit mit der Ada-Implementierungsgruppe der Universität Karlsruhe unter Leitung von Prof. Dr. G. Goos.

6.3 Implementation eines Prototyps

Der Systementwurf aus dem vorherigen Abschnitt wurde als Ausgangspunkt genommen, um einen Prototyp für ein Dialogsystem zu implementieren. Dieser Prototyp kann zwar nicht als Verwaltungssystem für interaktive Bildfolgenauswertesysteme bezeichnet werden, aber er gestattet im Rahmen von vertretbaren Einschränkungen die Gestaltung von Benutzerschnittstellen für Experimentalsysteme zur Bildfolgenauswertung.

Dieser Prototyp soll als ein erster Ansatz angesehen werden, den Kern eines Verwaltungssystems zu formulieren.

6.3.1 Wahl der Programmiersprache Ada

Die Qualität eines Dialogsystems für graphische Benutzerschnittstellen ist nicht nur von einer sorgfältigen Konzeption, sondern auch von der Wahl einer geeigneten Programmiersprache abhängig. Als Anforderung an eine Implementationssprache ergibt sich aus dem Kontext der Dialogsysteme, daß die Sprache eine angemessene Formulierung von parallelen Prozessen erlaubt, die über einen asynchronen Nachrichtenaustausch kommunizieren [*Olsen et al. 84*]. Diese Forderung wird in dem hier vertretenen Ansatz noch erweitert, indem die Sprache auch Mechanismen zur Formulierung von Objekten und eine Behandlung von asynchron ausgelösten Ereignissen unterstützen soll. Weitere Forderungen sind die Formulierung von Daten- und Programmabstraktionen, Schutzbereichen sowie die Unterstützung einer methodischen Programmentwicklung. Für die Programmumgebung wird eine Projektverwaltung sowie die Unterstützung durch Graphiksysteme gefordert.

Die Entwicklung eines solchen Dialogsystems zur Auswertung von Bildfolgen ist nur dann sinnvoll, wenn gleichzeitig entsprechende Anwendungssysteme gemäß der in Abbildung 1.2 dargestellten Struktur entwickelt werden. Dieses Dialogsystem ist deshalb als ein Teil einer geplanten Programmierumgebung für Experimentalsysteme zur Bildfolgenauswertung [*Faasch & Haarslev 85*] zu sehen. In diesem Zusammenhang ist auch die Entwicklung des SISSY-Systems zu nennen, das sich dieser Werkzeuge bedienen will [*Dreschler-Fischer & Haarslev 85*].

Die Wahl einer einheitlichen Programmiersprache für alle diese Systeme ist deshalb auf Ada gefallen. Diese Wahl wurde dadurch unterstützt, daß bereits seit längerem Erfahrungen mit Ada existieren. Im Rahmen eines

– *Sterngruppe*
Im Gegensatz zur einfachen Gruppe existiert in einer Sterngruppe immer ein einziges Objekt, das sich dadurch auszeichnet, daß es als einziges Mitglied der Gruppe an alle Gruppenmitglieder einen Rundspruch versenden kann. Alle anderen Gruppenmitglieder können nur diesem speziellen Objekt eine Nachricht zusenden. In Abbildung 6.13 wird am Beispiel der Eingabe-Objekte und des Maus-Objektes eine solche Gruppe skizziert. Dabei ist das Maus-Objekt das ausgezeichnete Objekt der Gruppe, während die Eingabe-Objekte die übrigen Gruppenmitglieder stellen. Solange das Maus-Objekt nicht von einem Eingabe-Objekt exklusiv belegt wird, versendet das Maus-Objekt nach einer Änderung des Mauszustandes diesen als Rundspruch an alle Eingabe-Objekte. Deren Belegungs- oder Freigabemeldungen gehen jedoch nur an das Maus-Objekt.

nur ein einziges Mal Mitglied sein darf. Der Erwerb einer Gruppenmitgliedschaft sollte zu jedem Zeitpunkt möglich sein.

Bei den Objektgruppen lassen sich zwei Organisationsformen unterscheiden:

- *Einfache Gruppe*
  In dieser Gruppenform sind alle Mitglieder in dem Sinne gleichberechtigt, daß jedes Gruppenmitglied an alle übrigen Mitglieder der Gruppe einen Rundspruch versenden kann. Ein Beispiel für eine derartige Gruppe sind die Darstellungsobjekte. Bei der Zuweisung einer Fensterposition auf dem Bildschirm kann eine Überlappung von Fenstern dadurch vermieden werden, daß ein Darstellungsobjekt seine aktuelle Fensterposition an alle Gruppenmitglieder versendet und eine Position nur dann akzeptiert, wenn von den Gruppenmitgliedern kein Widerspruch empfangen wird.

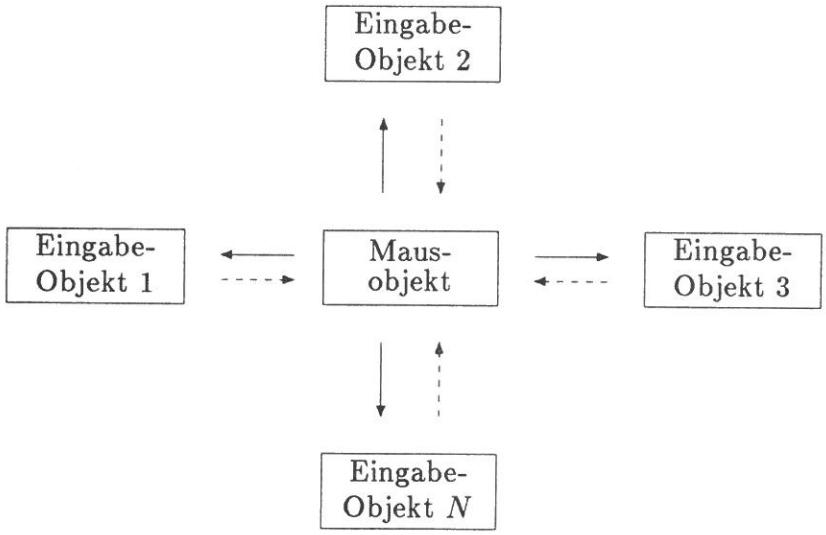

Abbildung 6.13: Beispiel für eine Sterngruppe

richten über den Zustand des Anwendungsobjektes austauschen.

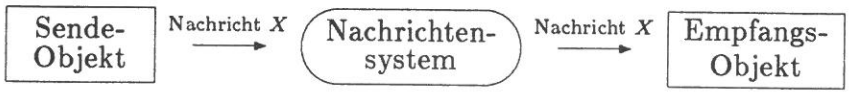

Abbildung 6.11: Kommunikation zwischen zwei Objekten

- *Nachrichtenaustausch innerhalb von Objektgruppen.*
  Im Rahmen des Dialogsystems ergibt sich die Notwendigkeit, daß eine Gruppe von Objekten untereinander Nachrichten austauschen wollen. Es gibt bei dieser Kommunikationsform immer ein Objekt der Gruppe (Sende-Objekt), das dieselbe Nachricht in Form eines Rundspruchs an alle Mitglieder dieser Gruppe versenden will, ohne die Identität seiner Gruppenmitglieder kennen zu müssen (siehe Abbildung 6.12).

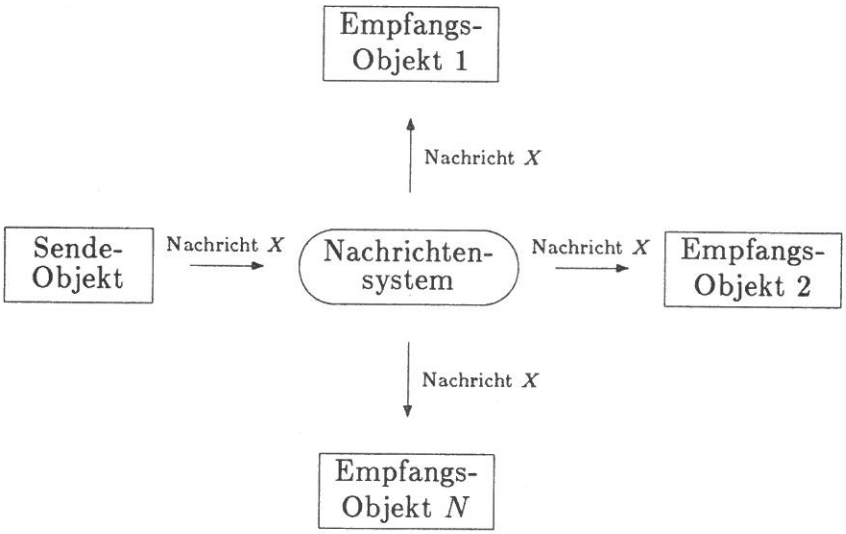

Abbildung 6.12: Gruppenkommunikation

Deshalb muß das Nachrichtensystem einen Mechanismus zur Verfügung stellen, der das Einrichten derartiger Gruppen und das Versenden von Nachrichten innerhalb einer Gruppe gestattet. Die Kommunikation der Gruppenmitglieder untereinander findet nur noch über einen vom Nachrichtensystem bereitgestellten eindeutigen Gruppennamen statt. Es erscheint sinnvoll, daß jedes Objekt in jeder Gruppe

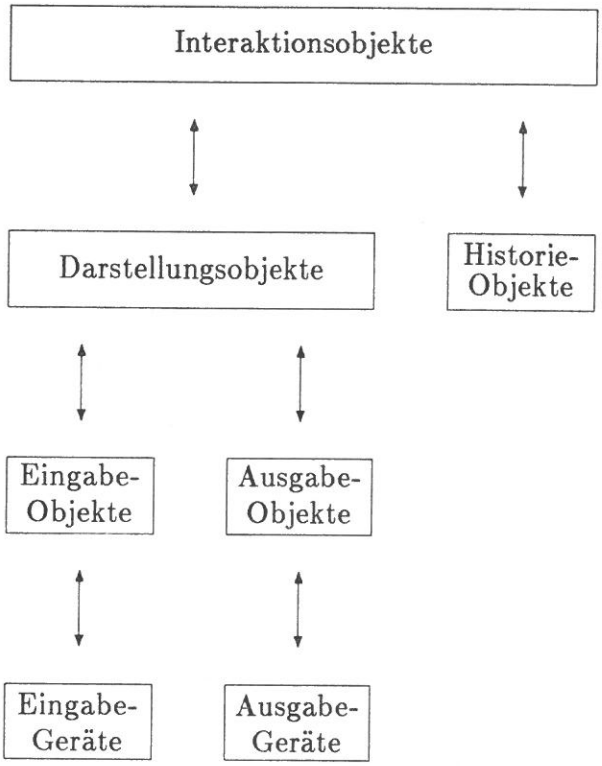

Abbildung 6.10: Objekthierarchie

6.2.7 Objektkommunikation

Zur Durchführung der in den vorherigen Abschnitten beschriebenen Aufgaben müssen die Objekte des Dialogsystems untereinander Nachrichten austauschen können. Dieser Nachrichtenaustausch sollte über ein Nachrichtensystem erfolgen. Aus dem Systementwurf der Objekte ergeben sich Forderungen an die von einem Nachrichtensystem zur Verfügung zu stellenden Kommunikationsprotokolle:

- *Nachrichtenaustausch zwischen genau zwei Objekten.*
  Es muß die Möglichkeit geben, daß ein bestimmtes Objekt (Sende-Objekt) einem anderen Objekt (Empfangsobjekt) eine Nachricht senden kann (siehe Abbildung 6.11). Dafür muß das Sende-Objekt die Identität des Empfangsobjektes im Rahmen des Nachrichtensystems kennen. Als ein Beispiel für eine derartige Kommunikation sind das Anwendungsobjekt und das ihm zugeordnete Interaktionsobjekt zu nennen, die untereinander über die Anwendungsschnittstelle Nach-

jekt das Maus- und das Tastatur-Objekt exklusiv belegen. Dieses kann notwendig sein, um in gewissen Zuständen in maus-sensitive Bereiche von anderen Objekten eintreten zu können, ohne daß diese aktiviert werden. Weiterhin wird bei einer exklusiven Belegung die Zahl der versandten Nachrichten reduziert.

6.2.6 Geräte-Objekte

Die vom Dialogsystem verwendeten graphischen Ein- und Ausgabegeräte werden von *Geräte-Objekten* modelliert. Diese Geräte sind im Dialogsystem von den anderen Objekten nur über die Geräte-Objekte ansprechbar.

Das Eingabegerät Maus besitzt einen Zustand, der mithilfe des VTE-Systems zu jedem Zeitpunkt gelesen werden kann. Dieser Zustand besteht aus der bezeichneten Markenposition und dem Status der Maus-Taste. Das *Maus-Objekt* überwacht und verarbeitet in periodischen Abständen den Zustand der Maus. Sobald sich der Zustand verändert, wird der neue Zustand der Maus in Form eines Ereignisses an alle Eingabe-Objekte oder, falls das Maus-Objekt von einem Eingabe-Objekt exklusiv belegt wurde, nur an dieses gesandt.

Das *Tastatur-Objekt* erhält als Ereignisse die Zeichen, die über die Tastatur vom Benutzer eingegeben werden. Das Tastatur-Objekt kann ebenfalls exklusiv belegt und wieder freigegeben werden.

Das GKS-System verwaltet das Ausgabegerät VTE. Das GKS-System wird über ein *GKS-Objekt* von den Ausgabe-Objekten angesprochen. Das GKS-Objekt definiert dafür einen Semaphor, der die unteilbare Benutzung der GKS-Operationen garantiert. Die GKS-Operationen werden auf die Operationen des VTE abgebildet. Dieses wird über ein *VTE-Objekt* angesprochen, welches ebenfalls einen Semaphor zur unteilbaren Benutzung definiert.

Die Geräte-Objekte sind ebenfalls wie die Eingabe- und Ausgabe-Objekte nach dem logischen Schema aus Abbildung 6.1 der Darstellungskomponente und im Sprachmodell der lexikalischen Ebene zuzuordnen.

Die vollständige Objekthierarchie findet sich in Abbildung 6.10.

entscheiden, ob diese Eingaben an sein Darstellungsobjekt gerichtet sind. Dies ist gemäß der Konzeption der Benutzerschnittstelle genau dann der Fall, wenn die durch die Maus bezeichnete Markenposition innerhalb des Bildschirmfensters liegt, das seinem Darstellungsobjekt zugeordnet ist. Ein derartiger Bereich wird deshalb als maus-sensitiv bezeichnet.

Zur Durchführung dieser Aufgabe erhält ein Eingabe-Objekt nach seiner Initialisierung von seinem Darstellungsobjekt einen Baum von maussensitiven Bereichen, der sich aus der Struktur des zugeordneten Bildschirmfensters ergibt. Unter einem *maus-sensitiven Bereich* wird dabei ein Teilfenster verstanden, das vom Benutzer manipulierbare Information darstellt. Für jedes Fenster ist vermerkt, wie die Eingabe-Ereignisse vom Eingabe-Objekt behandelt und an das Darstellungsobjekt gemeldet werden sollen. Diese Beschreibung besagt, ob für dieses Teilfenster eine Zeicheneingabe von der Tastatur möglich ist. Andernfalls werden Eingaben von der Tastatur ignoriert. Weiterhin ist angegeben, ob nur der Eintritt und der Austritt der Maus aus dem Teilfenster sowie ein Tastendruck oder auch zusätzlich jede Positionsveränderung innerhalb des Teilfensters an das Darstellungsobjekt signalisiert werden sollen.

Im Rahmen der Ereignishierarchie transformieren die Eingabe-Objekte die elementaren Ereignisse der Maus- und Tastatur-Objekte in Ereignisse, die sich an Konzepten des Darstellungsobjektes orientieren. Das Maus-Objekt meldet als Ereignis eine Positionsänderung der von ihm bezeichneten Marke auf dem Bildschirm oder die Betätigung der Maus-Taste. Das Tastatur-Objekt löst das Ereignis Zeicheneingabe aus. Die Eingabe-Objekte bearbeiten diese Ereignisse allein dadurch, daß sie feststellen, ob diese Ereignisse sie betreffen.

Nach der Positionsmeldung des Maus-Objektes wird geprüft, ob diese Position in einem der maus-sensitiven Bereiche enthalten ist. Bei Auswahl eines solchen Bereiches wird an das Darstellungsobjekt ein Ereignis weitergegeben, welches die Identifikation des ausgewählten Bereiches, den Tastenzustand und u.U. auch die genaue Position innerhalb des Bereiches beschreibt.

Das vom Tastatur-Objekt ausgelöste Ereignis "Zeicheneingabe" wird nur dann bearbeitet, wenn sich die momentane Position der Maus in einem der maus-sensitiven Bereiche befindet. In diesem Fall ergibt sich aus der Beschreibung des Bereiches, ob eine Zeicheneingabe möglich ist. Das eingegebene Zeichen wird dann dem Darstellungsobjekt übermittelt.

In Abhängigkeit von seinem Darstellungsobjekt kann das Eingabe-Ob-

zuzuordnen und realisiert in dem Sprachmodell die syntaktische und semantische Ebene.

6.2.4 Ausgabe-Objekte

Ein *Ausgabe-Objekt* erhält von seinem Darstellungsobjekt Aufträge zur Erzeugung von graphischen Darstellungen. Diese Aufträge bildet es auf die vom GKS angebotenen Operationen ab und gibt sie an das GKS-Objekt weiter. Zusätzlich ist ein Ausgabe-Objekt für die Erzeugung von Bewegungsgraphik wie Blinken und Objektverschiebung zuständig.

Die im Rahmen dieser Untersuchung verwendete Implementation von GKS bietet als logische Ausgabegeräte die Graphikspeicher und die Bildspeicher des Farbgraphik-Systems MBV der Fa. VTE (im weiteren als VTE bezeichnet). Die Graphikspeicher besitzen eine Auflösung von $512 \times 512 \times 1$ Bit und bieten eine Falschfarbendarstellung mit maximal 8 Farben. Die Bildspeicher dienen zur Darstellung von Farbbildern mit einer Auflösung von $512 \times 512 \times 8$ Bit je Farbauszug (RGB-Darstellung). Diese beiden Speicherarten sind im GKS als zwei logische Geräte verfügbar und werden vom Ausgabe-Objekt zur Graphikerzeugung angesprochen.

Den Farbbildspeicher verwendet das Ausgabe-Objekt zur permanenten Darstellung, den Graphikspeicher für temporäre Darstellungen und zur Erzeugung von bewegten Graphiken.

Das Ausgabe-Objekt ist nach dem logischen Schema aus Abbildung 6.1 der Darstellungskomponente zuzuordnen. Es bearbeitet in dem Sprachmodell die lexikalische Ebene.

6.2.5 Eingabe-Objekte

Ein *Eingabe-Objekt* überwacht ständig die Benutzereingaben. Diese Benutzereingaben können im Rahmen dieses Dialogsystems mithilfe von zwei Eingabegeräten erzeugt werden. Bei diesen Geräten handelt es sich um eine Maus als Zeigeinstrument, die eine Taste als Signalgeber besitzt, und eine alphanumerische Tastatur zur Eingabe textueller Information. Diese beiden Geräte werden durch zwei Geräte-Objekte (Maus-Objekt, Tastatur-Objekt) modelliert. Die Benutzereingaben werden somit von diesen Geräte-Objekten an alle im Dialogsystem bekannten Eingabe-Objekte gemeldet.

Die Hauptaufgabe eines Eingabe-Objektes besteht deshalb darin, den ständig eintreffenden Strom von Benutzereingaben zu überwachen. Es muß

Darstellungsobjekt gerichtet sind. Zur Durchführung dieser beiden Aufgaben benutzt jedes Darstellungsobjekt genau zwei ihm untergeordnete Objekte (siehe Abbildung 6.9). Sein Eingabe-Objekt überwacht die Benutzereingaben und meldet nach einer Vorverarbeitung die für sein Darstellungsobjekt bestimmten Eingaben weiter. Das Ausgabe-Objekt eines Darstellungsobjektes setzt dessen Aufträge zur graphischen Darstellung in die GKS-Notation um und führt sie anschließend aus.

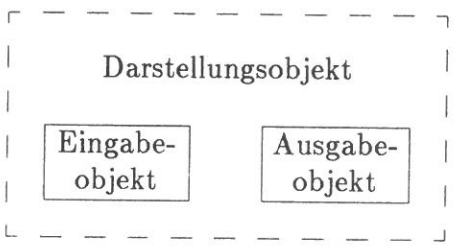

Abbildung 6.9: Struktur eines Darstellungsobjektes

Das Darstellungsobjekt behandelt die von dem Eingabe-Objekt weitergegebenen Benutzereingaben und erzeugt Ausgaben mithilfe des Ausgabe-Objektes. Diese Benutzereingaben werden gemäß der den Fensterelementen zugeordneten Strukturbeschreibung auf ihre korrekte Syntax und soweit möglich auf ihre korrekte Semantik kontrolliert.

Bei der Auswahl eines Fensterelementes durch den Benutzer hat das Darstellungsobjekt mehrere Aufgaben durchzuführen. Wurde ein neues Fensterelement ausgewählt, so müssen für das zuvor markierte Fenster eine Abschlußbehandlung und für das neu markierte eine Anfangsbehandlung erfolgen. Diese Behandlung wird dem Benutzer durch graphische Rückmeldungen signalisiert. Das Tastensignal der Maus ($1\times$ oder $2\times$ "Klick") wird entsprechend dem selektierten Fensterelement interpretiert. Die Zeicheneingabe wird auf Syntax und Semantik geprüft.

Das Darstellungsobjekt überwacht somit die Interaktionshandlungen (siehe Kapitel 5.6) des Benutzers. Bei offenen Interaktionshandlungen hält es einen temporären Objektzustand, der von ihm erst nach Abschluß der Interaktionshandlung übernommen wird. Abgeschlossene Interaktionshandlungen und die neuen Werte der Fensterelemente meldet es an sein Interaktionsobjekt.

Das Darstellungsobjekt ist der Dialogkontrolle (siehe Abbildung 6.1)

weiteren Attributen versehen werden kann.

6.2.2 Historie-Objekte

Jedes Interaktionsobjekt besitzt ein *Historie-Objekt*, das dessen objektbezogene Historie (siehe Kapitel 5.6) führt. Das Historie-Objekt erhält von seinem Interaktionsobjekt abgeschlossene Interaktionshandlungen und von der Interaktion mit dem Benutzer unabhängige Aktionen des Anwendungsobjektes gemeldet.

Bei den Interaktionshandlungen werden der Zeitpunkt (sowohl in der Objektzeit als auch in der Echtzeit), der veränderte Objektzustand und die Reaktion des Systems vermerkt. Die Folge von Interaktionshandlungen wird in einer permanenten Datenbasis gespeichert.

Alle Einträge in die Historie meldet das Historie-Objekt an ein spezielles Objekt, das die Systemhistorie verwaltet. Das Historie-Objekt bietet Operationen an, die eine schrittweise Inspektion und Modifikation einer Kopie der Historie gestatten.

6.2.3 Darstellungsobjekte

Ein *Darstellungsobjekt* ist für genau eine graphische Darstellung seines Interaktionsobjektes zuständig. Es erhält von seinem Interaktionsobjekt genau den Ausschnitt der Struktur- und Zustandsbeschreibung des Anwendungsobjektes, den das Darstellungsobjekt auf dem Bildschirm darstellen soll. Diese baumartige Objektbeschreibung wird von dem Darstellungsobjekt zur Erzeugung einer graphischen Darstellung bearbeitet und mit weiteren Informationen attributiert. In Abhängigkeit von der Art des Anwendungsobjektes wird die generelle Darstellungsform bestimmt (z.B. eine Linie bei einem Leitungsobjekt, ein Fenster bei einem Verarbeitungsobjekt). Bei der Darstellung mithilfe eines Fensters werden für die Fensterelemente u.a. ihre Position innerhalb des Fensters, der Platzbedarf in Form eines umfassenden Rechtecks und die Interaktionsklasse festgelegt. Die Interaktionsklasse gibt beispielsweise an, ob der Benutzer den Wert dieses Fensterelementes verändern darf.

Das Darstellungsobjekt ist nicht nur für die Erzeugung und Aktualisierung der graphischen Darstellung der Objektbeschreibung, sondern auch für die Verarbeitung der Benutzereingaben verantwortlich, die an dieses

Rahmen der Benutzerschnittstelle ein Verarbeitungs-, Leitungs-, Daten- oder Speicherobjekt darstellt. Weiterhin umfaßt die Beschreibung die Definition der Interaktionsschnittstelle des Anwendungsobjektes. Diese Interaktionsschnittstelle enthält eine Liste von Objektparametern und deren Typbeschreibung. Als Beispiele sind skalare, ganzzahlige, reellwertige Parameter und Zeichenketten zu nennen. Mit der Typbeschreibung der Parameter lassen sich die Benutzereingaben syntaktisch und semantisch überprüfen. Neben dieser einmaligen Objektdefinition kann das Interaktionsobjekt von seinem Anwendungsobjekt auch Änderungen über den Objektzustand mitgeteilt bekommen.

Zur Durchführung seiner Aufgaben verwendet das Interaktionsobjekt einige ihm untergeordnete Objekte (siehe Abbildung 6.8). Dazu gehören ein Historie-Objekt, das die Objekthistorie führt, und eine Menge von Darstellungsobjekten, die jeweils für eine graphische Darstellung ihres Interaktionsobjektes zuständig sind. Die Zuordnung eines Interaktionsobjektes zu N Darstellungsobjekten definiert somit einen Polymorphismus.

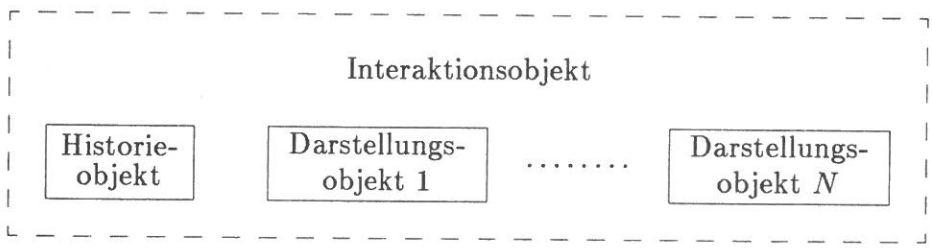

Abbildung 6.8: Struktur eines Interaktionsobjektes

Das Interaktionsobjekt ist für die Erzeugung und Koordination seiner Darstellungsobjekte zuständig. Weiterhin verwaltet es eine stets gültige Zustandsbeschreibung seines Anwendungsobjektes. Von seinen Darstellungsobjekten erhält es abgeschlossene Interaktionshandlungen gemeldet. Diese werden an das eigene Historie-Objekt weitergeleitet. Der Objektzustand wird entsprechend aktualisiert und bei Bedarf an das Anwendungsobjekt gesandt. Analog werden die Änderungsmeldungen des Anwendungsobjektes behandelt und an die betroffenen Darstellungsobjekte weitergeleitet. Eine weitere Aufgabe des Interaktionsobjektes besteht darin, die Meldungen seines Anwendungsobjektes in eine vom Dialogsystem benötigte interne Darstellung und umgekehrt zu transformieren. Als Interndarstellung wurde eine baumähnliche Struktur gewählt, die von den Darstellungsobjekten mit

als Anwendungsobjekt bezeichnet). Das Interaktionsobjekt führt stellvertretend für sein Anwendungsobjekt die Interaktion mit dem Benutzer durch. Die Interaktionsobjekte definieren somit zusammen mit ihren Anwendungsobjekten die Anwendungsschnittstelle im Bildfolgenauswertesystem (siehe Abbildung 6.6). Die Menge der Interaktions- und Anwendungsobjekte realisieren eine verteilte Kontrolle des Dialogs (siehe Abbildung 6.4).

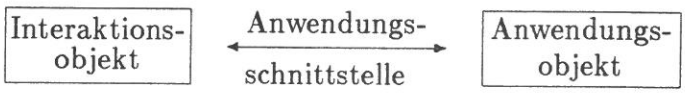

Abbildung 6.6: Kommunikation zwischen Dialog- und Anwendungssystem

Die Interaktionsobjekte sind in der logischen Struktur des Dialogsystems gemäß Abbildung 6.1 der Anwendungskomponente zuzuordnen. Ein weiterer Bestandteil der Anwendungskomponente dient der dynamischen Erzeugung von Interaktionsobjekten. Dafür existiert ein Objekt (Objektverwaltung), bei dem sich die Anwendungsobjekte nach ihrer Erzeugung anmelden müssen. Die Objektverwaltung erzeugt daraufhin ein Interaktionsobjekt, das sich nach einer Reihe von Initialisierungen mit seinem Anwendungsobjekt in Verbindung setzt (siehe Abbildung 6.7). Dieses Protokoll hat den Vorteil, daß das Dialogsystem immer nur aus der minimal benötigten Anzahl von Interaktionsobjekten besteht.

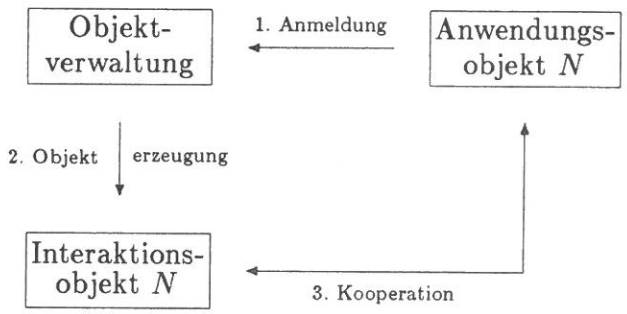

Abbildung 6.7: Protokoll der Objektanmeldung

Das Interaktionsobjekt erhält von seinem Anwendungsobjekt eine einmalige Objektdefinition in Form einer Struktur- und Zustandsbeschreibung. Aus der Strukturbeschreibung geht hervor, ob das Anwendungsobjekt in

6.2 Objektorientierter Systementwurf

Die *objektorientierte Entwicklung* [*Rentsch 82, Booch 86, Stefik & Bobrow 86*] ist ein Ansatz für den Entwurf von Programmsystemen, wobei ein System in eine Menge von Objekten zerlegt wird. Ein *Objekt* ist eine Einheit, dessen Verhalten durch die von ihm angebotenen Operationen und die von anderen Objekten benötigten Operationen bestimmt wird.

Ein derartiges System sollte nur aus Objekten bestehen, die untereinander über Nachrichten kommunizieren. Jedes Objekt ist ein eindeutig identifizierbarer Bestandteil eines solchen Systems. Bei umfangreichen objektorientierten Systemen hat es sich als nützlich erwiesen, diese aus einer *Hierarchie von Abstraktionsebenen* zu bilden [*Booch 86*]. Jede Ebene des Systems besteht aus Objekten und gestattet nur eine sehr *eingeschränkte Sichtbarkeit* von Objekten aus anderen Ebenen.

Die Anwendung einer derartigen objektorientierten Entwicklung für ein Dialogsystem wird in diesem Abschnitt vorgestellt. Dieses Dialogsystem realisiert die in Kapitel 5 konzipierte Benutzerschnittstelle und ist als ein Teil eines interaktiven Experimentalsystems zur Auswertung von Bildfolgen anzusehen.

Dieses Dialogsystem erhebt den Anspruch, nicht nur für ein spezielles Anwendungssystem, sondern für eine ganze Klasse von Anwendungssystemen einsetzbar zu sein. Diese Anwendungssysteme zur Bildfolgenauswertung werden mithilfe eines Werkzeuges zur Konfiguration von Experimentalsystemen für die Bildfolgenauswertung [*Faasch 86*] generiert. Das nachfolgend vorgestellte Dialogsystem muß somit in der Lage sein, für jede denkbare Konfiguration von Anwendungssystemen die geforderte Benutzerschnittstelle zu realisieren.

Die nachfolgenden Abschnitte beschreiben einen entsprechenden Systementwurf, der das Dialogsystem ausgehend von der konzipierten Benutzerschnittstelle in mehrere Abstraktionsebenen (top down) strukturiert. Diese Abstraktionshierarchie lehnt sich an die am Anfang dieses Kapitels erläuterte Sprachhierarchie an.

6.2.1 Interaktionsobjekte

Die Benutzerschnittstelle modelliert ein Bildfolgenauswertesystem mithilfe von Interaktionsobjekten. Ein *Interaktionsobjekt* repräsentiert im Dialogsystem genau eine Komponente des Anwendungssystems (nachfolgend

tur entspricht und zur Dialogkontrolle das Ereignismodell verwendet, wird in *Green 85* vorgestellt. Mithilfe einer speziellen Sprache können Prozesse zur Behandlung von asynchron ausgelösten Ereignissen formuliert werden. Eine solche Ereignisbehandlung besteht aus drei Teilen. Der erste Teil definiert die akzeptierten Ereignisse, dann folgt die Deklaration von lokalen Datenstrukturen, der dritte Teil enthält Anweisungen zur Bearbeitung von Ereignissen. Diese Sprache wird auf die Programmiersprache C abgebildet und zur Ausführung von einem Laufzeitsystem unterstützt.

Die Beschreibung eines Dialogsystems auf der Grundlage von Prozessen hat den Vorteil, daß eine flexible Behandlung von graphischen Geräten möglich ist. Von Nachteil ist dagegen, daß diese Notation weniger formal als die Grammatik- oder Übergangsnetznotation ist. Die Verwendung von Prozessen gestattet dafür eine angemessene Behandlung von Bildschirmfenstern und eine Strukturierung der Fenster und Geräte in Form von Hierarchien (siehe auch *Rosenthal 83*). Die Verwendung formaler Methoden erscheint nur dann sinnvoll, wenn Systeme zur Handhabung derselben verfügbar sind. Als Beispiele seien das System RAPID/USE [*Wasserman 85*] und die in *Jacob 85* vorgestellte Programmumgebung für Übergangsnetze genannt.

Zur Realisierung eines Dialogsystems für die in dieser Arbeit entwickelten Schnittstelle stehen jedoch derartige Werkzeuge nicht zur Verfügung. Weiterhin basiert die Benutzerschnittstelle auf der Gestaltung durch Objekte. Es erscheint deshalb sinnvoll, zur Realisierung einer objektorientierten Schnittstelle ebenfalls ein aus Objekten bestehendes Dialogsystem zu entwickeln. Aufgrund der Gestaltungskonzepte der Schnittstelle kann der Benutzer zu jedem Zeitpunkt mithilfe von Eingabegeräten Interaktionshandlungen mit den Objekten durchführen. Deshalb sollen die Benutzereingaben aus der Sicht der Objekte als asynchron ausgelöste Ereignisse modelliert werden. Diese Entscheidung wird durch die Erfahrung von mehreren Experten für die Erzeugung von Dialogsystemen unterstützt, die als Grundlage für ein Verwaltungssystem eine asynchrone Ereignisverarbeitung mithilfe eines Nachrichtensystems empfehlen [*Olsen et al. 84*]. Diese Konzepte sollten von der Implementationssprache des Verwaltungssystems voll unterstützt werden.

Der nachfolgende Abschnitt erläutert den Systementwurf für ein Dialogsystem, das die in dieser Arbeit entwickelte Benutzerschnittstelle realisiert.

Spracherweiterung realisierte Werkzeug ist als benanntes, typgebundenes Objekt definiert. Es wird durch ein vorgegebenes Eingabemuster (trigger) gestartet, führt eine interne Prozedur aus und produziert eine Ausgabe. Das Ein-Ausgabe-Werkzeug besteht prinzipiell aus einem Initialisierungsteil, einer Eingaberegel zur Definition des akzeptierten Musters, einem Abbruchabschnitt, einem Rumpf und einer Ausgaberegel, die die Ergebnisse übergeordneten Werkzeugen zur Verfügung stellt. Die Eingaberegeln ähneln den "Path Expressions" [*Campbell & Habermann 74*]. Die Werkzeuge werden nach PASCAL oder MODULA-2 übersetzt und zur Ausführung von einem Laufzeitsystem unterstützt. Ein Ein-Ausgabe-Werkzeug kann nur dann von einem Benutzer beeinflußt werden, wenn es eingabebereit ist. Weiterhin kann es keine Variablen besitzen, die eine über ihre Aktivierung hinausgehende Lebensdauer haben. Deshalb kann mit diesem Werkzeug beispielsweise kein Drehknopf nachgebildet werden, der jederzeit vom Benutzer veränderbar ist und seinen Wert zwischen der Benutzung beibehält.

SQUEAK [*Cardelli & Pike 85*] ist ebenfalls eine Sprache zur Realisierung von Dialogsystemen. Sie verwendet parallele Prozesse, die über Kanäle Nachrichten austauschen. Die Syntax orientiert sich an "Communicating Sequential Processes" [*Hoare 78*]. Die Geräte sind über vordefinierte Kanäle verfügbar. SQUEAK unterstützt die parallele Bearbeitung von mehreren Eingabegeräten und eignet sich besonders gut zur Geräte-Ansteuerung. SQUEAK wird in die Programmiersprache C transformiert.

GRINS [*Olsen et al. 85*] verwendet zur Strukturierung hierarchisch organisierte, mit Parametern und einem Resultatwert versehene Darstellungsobjekte. In *Carlson et al. 83* wird das Dialogsystem in Moduln organisiert, die über Nachrichten kommunizieren. Das ADM-System [*Schulert et al. 85*] zerlegt das Dialogsystem in Aufgaben und Zustände. Die Aufgaben können untereinander über Ereignisse kommunizieren.

Ein auf Lisp basierendes System [*Lieberman 85*] zur Erzeugung von graphischen Schnittstellen kennt drei Objektarten. Die Kommando-Objekte stellen die vom Benutzer durchführbaren Aktionen dar. Die Darstellungsobjekte dienen als graphische Darstellung von Daten des Anwendungssystems. Ein dritter Objekttyp realisiert die Anwendungsschnittstelle und umfaßt ein Bildschirmfenster, einen Prozeß, eine Interaktionsschleife und interne Daten. Die Objekte sind voneinander hierarchisch abhängig und kommunizieren über Nachrichten miteinander.

Ein Verwaltungssystem, das der in Abbildung 6.1 beschriebenen Struk-

die Lösung der oben angedeuteten Schwächen erlaubt, modelliert ein Dialogsystem mithilfe von *Prozessen* bzw. *Objekten* (auch als *abstrakte Geräte* bezeichnet). Dieses Konzept besitzt eine starke Verwandtschaft mit dem von Smalltalk [*Goldberg & Robson 83*] begründeten objektorientierten Ansatz. Die Kommunikation dieser Prozesse kann synchron oder asynchron über Nachrichten bzw. Prozeduraufrufe erfolgen. Die asynchrone Kommunikation mit Nachrichten wird in dem Kontext der Verwaltungssysteme auch als *Ereignismodell* bezeichnet (siehe auch *Green 84, Green 85*).

In *Edmonds 82* wird ein solches Konzept für die Modellierung der Mensch-Maschine-Schnittstelle vorgestellt. Dieses definiert Ein-Ausgabe-Prozessoren, die physische Aktionen des Benutzers in eine interne Darstellung umsetzen oder umgekehrt die interne Darstellung graphisch darstellen. Ein Dynamik-Prozessor kontrolliert den Dialogablauf. Ein Hintergrund-Prozessor ist für die Durchführung der vom Benutzer initiierten Aufgaben zuständig.

Anson 82 modelliert ein interaktives System mithilfe von abstrakten Geräten. Geräte können wieder aus einfacheren Geräten bestehen. Sie sind über Leitungen verbunden und können über Ereignisse miteinander kommunizieren. Die Beschreibung eines Gerätes enthält eine externe Sicht, die dessen Interaktion mit anderen Geräten spezifiziert, und eine interne Implementation. Die Geräte werden programmiersprachlich mit einer zu PASCAL ähnlichen Syntax als Prozeßtypen beschrieben. Die Prozesse bestehen aus lokalen Variablen, einem Initialisierungsteil, nach außen sichtbaren Operationen und enthalten einen Teil zur Bearbeitung von Ereignissen.

Das Konzept der Dialogzellen (dialogue cell) wird zur Definition von Interaktionen vorgeschlagen [*Borufka et al. 82*]. Diese Zellen bilden eine eigenständige Funktionseinheit. Sie umfassen vier Elemente, in die nach diesem Modell der Dialog zerlegt wird. Es sind eine Aufforderung an den Benutzer zur Eingabe (prompt), die Benutzereingabe, eine Rückmeldung und der aus der Eingabe resultierende Wert. Diese vier Elemente können zur Beschreibung der Aktionsfolge einer Zelle kombiniert werden. Eine Aktionsfolge darf weitere Dialogzellen über Prozeduren mit Parametern aufrufen. Die Organisation der Dialogzellen ist hierarchisch. Über einen Dialoggenerator sollen die aus Zellen bestehenden Spezifikationen in eine Programmiersprache übersetzt und auf die Fähigkeiten von GKS [*Enderle et al. 84*] abgebildet werden.

Ein zu den beiden vorherigen Ansätzen ähnlicher Vorschlag ist das Ein-Ausgabe-Werkzeug (input output tool) [*Bos et al. 83*]. Dieses als

taz 85a, Coutaz 85b] verwendet ebenfalls einen attributierten Baum als interne Repräsentation. Die Blätter des Baumes enthalten darzustellende Information mit einem fiktiven umfassenden Rechteck. Die Knoten dienen der Zusammensetzung von komplexeren Bildschirmelementen.

FORMAL [*Shu 85*] definiert eine Sprache, die die Beschreibung von Formularen erlaubt. Die Formulare können verknüpft und zu umfangreicheren Formularen zusammengesetzt werden. Dieses System fand zur Erzeugung von Dialogsystemen eine Anwendung, die für die Bearbeitung von relational gespeicherten Daten verwendet werden.

Das TIGER-System [*Kasik 82*] bietet eine spezielle, anwendungsunabhängige Dialogspezifikationssprache und einen Laufzeit-Interpretierer, der den Dialog steuert. Die Dialogspezifikationssprache ist blockorientiert, gestattet die Definition von Kommandos und wird zur Interpretation in einen Baum übersetzt. Als Interaktionsform wurden Menüs gewählt.

SCREEN RIGEL [*Rowe & Shoens 83*] ist ein Bestandteil der Programmiersprache RIGEL und gestattet die Behandlung von Bildschirm-Masken. RIGEL ist eine PASCAL-ähnliche Sprache zur Programmierung von Datenbanksystemen. Mithilfe von sog. Rahmen (frame) kann der Benutzer Masken beschreiben. Ein Rahmen enthält den Aufbau und die Elemente einer Maske sowie die Operationen zur Bearbeitung der Benutzereingaben.

Die Verwendung eines Prolog-Systems zur formalen Spezifikation einer Benutzerschnittstelle für ein Luftfrachtsystem wird in *Roach & Nickson 83* beschrieben. Durch den Prolog-Interpretierer konnte die Funktionalität der Schnittstelle sofort geprüft werden.

Die Definition von speziellen Dialogsprachen oder von Spracherweiterungen ist im Vergleich zu den Grammatiken weniger formal. Dafür bieten sie meistens eine größere Flexibilität und gestatten eine algorithmische Schnittstellenspezifikation.

Modellierung auf der Grundlage von Prozessen

Die meisten der bisher vorgestellten Methoden oder Systeme zur Verwaltung von interaktiven Systemen haben im Rahmen der in Abbildung 6.1 skizzierten Struktur den Schwerpunkt auf die Dialogkontrolle gelegt. Die gerätenahe Behandlung von graphischen Ein- und Ausgaben sowie die flexible Einbettung unterschiedlicher Geräte (siehe auch *Rosenthal 83*) wird in diesen Systemen meistens nur sehr eingeschränkt behandelt.

Ein Konzept, welches sowohl eine detaillierte Dialogkontrolle als auch

Elemente der Masken ist eine Angabe von semantischen Überprüfungen anhand von Typprüfungen, Bereichsprüfungen und logischen Ausdrücken möglich.

Zusammenfassend läßt sich sagen, daß eine interaktive Erstellung und Erprobung von graphischen Benutzerschnittstellen durchaus eine Alternative zur formalen Spezifikation darstellt. Diese Systeme haben jedoch meistens den Nachteil, daß sie nur wenige Interaktionsformen unterstützen (z.B. MENULAY [*Buxton et al. 83*] für Menüs, KSBASS [*Bass 85*] für formularähnliche Masken).

Spezielle Sprachen

Die in diesem Abschnitt beschriebenen Systeme zeichnen sich dadurch aus, daß sie zur Beschreibung der Benutzerschnittstelle eine eigene Sprache definieren oder vorhandene Sprachen erweitern.

COUSIN [*Hayes & Szekely 83*] gestattet die Beschreibung von an Kommandosprachen orientierten Schnittstellen. Deren graphische Darstellung ist an Masken oder Formulare gebunden. Die Zusammensetzung und das Aussehen der Formulare wird in einer speziellen Schnittstellensprache spezifiziert. Dieser Ansatz wird in *Hayes 84* erweitert, um aus einfachen Interaktionskonzepten abstraktere zu bilden.

In DESCARTES [*Shaw et al. 83*] wird ein Ansatz vorgestellt, der die Prinzipien der Programmiermethodik und Sprachgestaltung anwendet, um von Programmiersprachen aus ansprechbare interaktive Schnittstellen für beliebige Programmsysteme zu schaffen. Dabei wird auf eine Verknüpfung der graphischen Darstellung der Anwendungsdaten und ihrer internen Repräsentation im Anwendungssystem Wert gelegt. Es soll beispielsweise immer die graphische Darstellung einer Variablen des Anwendungssystems dem aktuellen Wert der Variablen entsprechen. Die Struktur der Schnittstellenbeschreibungssprache wurde von der in SCRIBE [*Reid 80*] verwendeten Spezifikationssprache abgeleitet. Es werden als Datenbasis Stildefinitionen eingeführt, die beispielsweise gewisse Entscheidungen über das Format der Darstellung von Daten und die Form des Dialogprotokolls umfassen. Zur Behandlung der Anwendungsdaten und ihrer graphischen Darstellung wird eine attributierte, baumartige Interndarstellung aufgebaut. Aus diesem Baum wird über vorgegebene Regeln zur Attributauswertung und -vererbung eine graphische Darstellung erzeugt.

Ein ähnlicher Ansatz zur Gestaltung einer Benutzerschnittstelle [*Cou-*

grammentwicklung besteht bei den Netzen die Tendenz, eine zu sehr am Detail und an Modi orientierte Schnittstelle zu entwerfen.

Ein weiteres wichtiges Problem bei beiden Ansätzen ist die Behandlung von gleichzeitigen Handlungen. Dies soll am Beispiel der in Kapitel 5 konzipierten Benutzerschnittstelle verdeutlicht werden. Der Benutzer kann dort mit einem Objekt kommunizieren (z.B. Parameter besetzen), während andere Objekte ihre Darstellung auf dem Bildschirm aktualisieren. Diese Aktionen erfolgen sowohl gleichzeitig als auch unabhängig voneinander. Ein entsprechendes Übergangsnetz müßte aus mehreren Teilen bestehen, die parallel zueinander arbeiten können.

Die nachfolgend vorgestellten Methoden oder Ansätze können im Vergleich mit den beiden ersten Methoden als weniger formal bezeichnet werden.

Interaktive Gestaltung

Es gibt einige Verwaltungssysteme, die dem Benutzer die interaktive Gestaltung einer Benutzerschnittstelle ermöglichen. Der Benutzer kann mithilfe eines speziellen Editors oder auch über ein Menüsystem die Bildschirmaufteilung und die Darstellung von Anwendungsdaten direkt in ihrer graphischen Repräsentation erstellen. Diese Verwaltungssysteme beschränken sich üblicherweise auf bestimmte Interaktionsformen und ihre Darstellung.

Das System FLAIR [*Wong & Reid 82*] erlaubt die Gestaltung eines Dialogs über Menüs. Der konzipierte Dialog wird automatisch gesichert und kann als Bildfolge abgespielt werden. Als Eingabegeräte werden eine Spracherkennung und ein Zeigegerät, als Ausgabegeräte ein Graphikgerät und ein Sprachgenerator unterstützt.

Ein anderes System [*Buxton et al. 83*] besteht aus zwei Komponenten. Mit MENULAY kann der Benutzer unter Verwendung von interaktiven graphischen Techniken den zukünftigen Dialog in Form von Menüs gestalten. Die Menüs werden durch MAKEMENU von einer internen tabellarischen Beschreibung in C-Programme übersetzt, die vordefinierte Graphikpakete verwenden. Ein gestalteter Dialog kann ebenfalls wie in FLAIR als Bildfolge abgespielt werden.

KSBASS [*Bass 85*] ist ein System zur Spezifikation von Benutzerschnittstellen für Datenbanken. Der zukünftige Dialog kann in Form von Bildschirm-Masken über einen Editor erstellt und modifiziert werden. Für die

können bis zum Zeilenende überlesen und das Zeichen für Zeilenende umdefiniert werden.

- *Strukturierung des Übergangsnetzes*
  Das Netz kann in Teilnetze zerlegt werden, die als Result einen Wert zurückliefern können. Es ist möglich, die Benutzereingabe für die Zustandsübergänge durch Teilnetze zu beschreiben.

- *Kommunikation mit semantischen Aktionen*
  Die mit den Übergängen ausgelösten semantischen Aktionen (meistens in Form von Prozeduren) sind in der Lage, einen Wert an das Netz zurückzugeben. Dieser erhaltene Wert kann verarbeitet werden, um beispielsweise eine Verzweigung durchzuführen. Weiterhin besteht die Möglichkeit, ein Kante mit einer Zeitschranke zu versehen. Wird diese Zeitschranke ohne eine zwischenzeitliche Benutzereingabe überschritten, so wird dieser Zustandsübergang durchgeführt.

Mit RAPID/USE wurde ein Dialogsystem für ein Datenverzeichnis-System realisiert, das der strukturierten Spezifikation von Systemen dient. RAPID/USE wurde in erster Linie für alphanumerische Bildschirmgeräte konzipiert, es sind jedoch Erweiterungen zur Behandlung von Graphik-Sichtgeräten geplant.

In *Jacob 85* wird eines der Werkzeuge für ein Verwaltungssystem vorgestellt. Dieses gestattet die interaktive Spezifikation einer Benutzerschnittstelle mit Übergangsnetzen. Die Netze können graphisch in mehreren Fenstern dargestellt und ihre Abarbeitung kann verfolgt werden.

Die Verwendung von Übergangsnetzen als formales Beschreibungsmittel bietet dieselben Vorteile wie die Verwendung kontextfreier Grammatiken. Gegenüber den Grammatiken definieren die Übergangsnetze eine explizite zeitliche Abfolge der Benutzeraktionen. Ebenso wie bei der vorherigen Methode dienen die Übergangsnetze einer externen Kontrolle des Dialogs. Sie sind besonders gut zur Beschreibung von linearen, alphanumerischen Schnittstellen geeignet. Die Erweiterungen zur Handhabung von graphischen Schnittstellen (siehe *Wasserman 85*) sind jedoch mühsam und wenig überzeugend. Für die Netze und Grammatiken gilt, daß bei Schnittstellen mit vielen vom Benutzer jederzeit durchführbaren Operationen (modusvermeidende Interaktion) die formale Beschreibung sehr umfangreich und mit vielen Details behaftet ist. Analog zu den Flußdiagrammen für die Pro-

Eine von fast allen Systemen vorgenommene Erweiterung [*Green 84*] erlaubt die Aufteilung des Übergangsnetzes in eigenständige Teilnetze. Ein *Teilnetz* ersetzt eine entsprechende Menge von Knoten und Kanten eines anderen Netzes. Sobald ein Teilnetz aktiviert wird, behält es die Kontrolle über den Dialog bis einer der Endzustände des Teilnetzes erreicht ist. Eine derartige Erweiterung wurde beispielsweise zur Realisierung eines militärischen Nachrichtensystems verwendet [*Jacob 83b*].

Ein (Teil-)Netz wird als rekursiv bezeichnet, wenn es sich selbst rekursiv aktivieren kann. In *Kieras & Polson 83* werden die rekursiven Übergangsnetze um globale Datenstrukturen erweitert. Der Inhalt dieser auch als Register bezeichneten Datenstrukturen wird von den mit den Kanten assoziierten Aktionen manipuliert. Ein Zustandsübergang und damit auch die ausführbaren Aktionen können von der Auswertung einer logischen Bedingung abhängen. Bei der Formulierung dieser logischen Bedingungen werden die Register mit einbezogen.

In *Green 83* wird eine Spezifikationssprache GUSL (Graphical User interface Specification Language) beschrieben, die auf der Verwendung von Übergangsnetzen basiert. Mithilfe von GUSL kann man eine graphische Benutzerschnittstelle in Form von Blöcken spezifizieren. Jeder dieser Blöcke realisiert einen Teil des Dialogsystems.

Für das Verwaltungssystem RAPID/USE [*Wasserman 85*] wurden vielfältige Erweiterungen an den Übergangsnetzen vorgenommen, um komplexe Benutzerschnittstellen spezifizieren zu können:

- *Spezifikation von Systemausgaben*
  Ein Knoten wird zur Darstellung einer Nachricht benutzt. Dabei können neben der Nachricht auch ihre Position auf dem Bildschirm sowie weitere Steuerzeichen zur Bildschirmkontrolle angegeben werden. Weiterhin ist die Definition von Nachrichtenvariablen und ihre Referenzierung, das Setzen von Tabulatoren und die Adressierung von verschiedenen Bildschirmfenstern möglich. Ähnlich zu *Kieras & Polson 83* werden globale Variablen eingeführt, die sowohl für Nachrichten als auch in Aktionen verwendet werden können.

- *Verarbeitung der Benutzereingaben*
  Von dem Schema der gepufferten Eingabe, die durch das Zeilenende abgeschlossen wird, kann auf eine ungepufferte, direkte Zeichenverarbeitung umgeschaltet werden. Dadurch ist eine sofortige Systemreaktion auf bestimmmte Eingabezeichen möglich. Eingabezeichen

Durch die Verwendung von kontextfreien Grammatiken realisieren die Verwaltungssysteme eine externe Kontrolle des Dialogs (siehe auch Abbildung 6.3). Damit sind die Möglichkeiten des Anwendungssystems, den Dialog zu beeinflussen, als äußerst gering anzusehen. Weiterhin ist von Nachteil, daß die zeitliche Abfolge der Benutzeraktionen implizit definiert wird. Bei genügend komplexen Schnittstellen wird eine BNF-Beschreibung sehr umfangreich und für den menschlichen Betrachter schwer zu handhaben. Eine verständliche Grammatik ist ferner von der guten Namensgebung der Meta-Symbole sowie von einer übersichtlichen Regelstruktur abhängig.

Die nachfolgend dargestellten Zustandsübergangsdiagramme besitzen eine starke Verwandtschaft mit den kontextfreien Grammatiken. Es ist möglich, die Zustandübergangsdiagrammme textuell als kontextfreie Grammatik darzustellen und umgekehrt aus einer kontextfreien Grammatik ein entsprechendes Zustandsübergangsdiagramm zu erzeugen [*Jacob 83a*].

Zustandsübergangsdiagramme

Die *Zustandsübergangsdiagramme* oder auch *Übergangsnetze* werden von Systemen verwendet, die die Benutzerschnittstelle als eine Sammlung von Zuständen modellieren. Die Aktionen des Benutzers bewirken Übergänge in dem Zustandsnetz.

Ein Übergangsnetz besteht aus einer Menge von gerichteten Graphen. Jeder gerichtete Graph enthält eine Menge von Knoten und Kanten. Die Knoten repräsentieren die Zustände der Benutzerschnittstelle, die Kanten beschreiben die vom Benutzer durchführbaren Aktionen. Befindet sich das System in einem dieser Zustände, so hat der Benutzer diejenigen Aktionen zur Auswahl, mit denen die von diesem Zustand ausgehenden Kanten markiert sind. Führt er eine dieser Aktionen durch, so vollzieht das System einen Übergang in den vom Kopf der Kanten gekennzeichneten Zustand und führt die mit diesem Übergang assoziierten Aktionen durch.

Ein reines Übergangsnetz, das nur aus einer Menge von Knoten und Kanten besteht, ist für die Beschreibung einer Vielzahl von Benutzerschnittstellen nicht mächtig genug. Weiterhin ist es schwierig zu handhaben, da bei hinreichend komplexen Benutzerschnittstellen die Tendenz zu sehr umfangreichen Übergangsnetzen besteht [*Jacob 83a*].

Es gibt trotzdem viele Systeme, die zur Dialogbeschreibung Übergangsnetze verwenden. Diese Netze wurden jedoch dafür vielfältig erweitert. Einige dieser Erweiterungen werden nachfolgend im Rahmen von Beispielsystemen vorgestellt.

Dies erfolgt als Abbildung auf Datenstrukturen und Prozeduren. Die *syntaktische Ebene* definiert die syntaktischen Elemente der Kommandosprache (z.B. Kommandos, Argumente). Die Bedeutung eines jeden Kommandos wird mithilfe der durch die semantische Ebene eingeführten Operationen festgelegt. Die *Interaktionsebene* beschreibt die physischen Aktionen des Benutzers, die mit jedem Element der syntaktischen Ebene verbunden sind, sowie die Regeln zur Dialogsteuerung. Die *Bildschirmaufteilungsebene* (spatial layout level) enthält die verwendete Gerätekonfiguration und die Graphikdarstellung. Alle verbleibenden physikalischen Merkmale des Dialogsystems werden in der *Geräteebene* behandelt.

Als Beispiel wird der Entwurf eines kleinen Teils eines Nachrichtensystems beschrieben.

Alle bisher vorgestellten Ansätze [*Reisner 81, Shneiderman 82, Moran 81*] sind als Vorschläge zur formalen Beschreibung von Dialogsystemen zu verstehen. Keine dieser Methoden wurde zur Realisierung von Verwaltungssystemen für interaktive Systeme verwendet.

Bei SYNGRAPH [*Olsen & Dempsey 83a, Olsen & Dempsey 83b*] handelt es sich um ein existierendes Verwaltungssystem, das aus der Beschreibung einer Benutzerschnittstelle mithilfe einer erweiterten BNF-Notation automatisch ein entsprechendes Dialogsystem erzeugt. Die BNF-Notation beschreibt eine erweiterte LL(1)-Grammatik, die auf der rechten Seite ihrer Regeln auch reguläre Ausdrücke zuläßt. Die mit den Regeln assoziierten Aktionen werden in Form von PASCAL-Prozeduren realisiert. Weiterhin können Operationen zur Graphikausgabe sowie zur Fehlerbehandlung und Ausgabe von Hilfestellungen angegeben werden. Aus dieser Beschreibung wird von dem Verwaltungssystem ein PASCAL-Programm erzeugt [*Olsen 83*]. Das Verwaltungssystem verwendet das am Anfang dieses Kapitels dargestellte Sprachmodell, um die Behandlung der Benutzereingaben zu strukturieren.

Die Verwendung von kontextfreien Grammatiken zur Beschreibung von Benutzerschnittstellen hat den Vorteil, daß eine formale Beschreibung der Schnittstelle existiert. Erfahrungen haben gezeigt, daß beim Entwurf eines Dialogsystems eine formale Spezifikation der Schnittstelle durch eine BNF-Notation sehr hilfreich sein kann [*Reisner 81, Moran 81*]. Eine formale Beschreibung ist gleichzeitig eine notwendige Basis, um eine systematische und vollständige Evaluierung und Ausprüfung von Benutzerschnittstellen zu ermöglichen oder gar zu automatisieren [*Draper & Norman 85*].

zifische Systemreaktionen möglich werden, gibt es die Möglichkeit, Meta-Symbolen einen Wert zuzuweisen und diesen im Rahmen einer Ausgabe zu verwenden. Für die Fehlerbehandlung existiert ein spezielles Meta-Symbol, das alle Zeichenketten akzeptiert, falls keine andere Regel anwendbar ist. Zur besseren Behandlung von Ausgaben werden Erweiterungen wie die Ausgabe von graphischen Attributen (z.B. Unterstreichung, Negativdarstellung, Farbe) und die Definition und Adressierung von Bildschirmfenstern eingeführt.

Ein anderer Vorschlag führt den Begriff der *Kommandosprachen-Grammatik* (*command language grammar*) ein [*Moran 81*]. Dieser Vorschlag wird von dem Autor sowohl als Hilfsmittel für die Konzeption als auch zur Beschreibung oder Analyse eines Dialogsystems verstanden. Die Kommandosprachen-Grammatik zerlegt ein Dialogsystem in drei Ebenen:

- Die *konzeptuelle* Ebene enthält die abstrakten Konzepte, auf denen das Dialogsystem basiert.
- Die *Kommunikationsebene* beschreibt die Kommandosprache und den Dialog.
- Die *physikalische* Ebene enthält die Behandlung der Ein- und Ausgabegeräte, mit denen der Benutzer arbeitet.

Diese Ebenenstruktur wird noch weiter aufgeteilt (siehe Abbildung 6.5).

Konzeptuelle Ebene:	Aufgabenebene
	Semantische Ebene
Kommunikationsebene:	Syntaktische Ebene
	Interaktionsebene
Physikalische Ebene:	Bildschirmaufteilungsebene
	Geräteebene

Abbildung 6.5: Ebenenstruktur der Kommandosprachen-Grammatik (nach *Moran 81*)

Die *Aufgabenebene* dient der Analyse der Benutzerwünsche und der Strukturierung seines Aufgabengebietes im Rahmen des interaktiven Systems. Die *semantische Ebene* beschreibt die Zerlegung des Systems in eine Menge von Einheiten und Operationen zur Manipulation dieser Einheiten.

folgen, das diese Systeme u.a. nach ihrer Realisierung der Dialogkontrolle (siehe Abbildung 6.1) einteilt. Das Spektrum der von bekannten Systemen verwendeten Methoden reicht von kontextfreien Grammatiken, Zustandsübergangsdiagrammen, speziellen Dialogbeschreibungssprachen, über eine interaktive Dialoggestaltung bis hin zur Modellierung auf der Grundlage von Prozessen.

Kontextfreie Grammatiken

Die *kontextfreien Grammatiken* werden meistens in einer erweiterten *Backus-Naur-Form* (BNF) beschrieben. Systeme, die eine BNF-Notation verwenden, definieren mithilfe einer Grammatik die gültigen Eingabeaktionen des Benutzers. Die Terminal-Symbole der Grammatik sind die von der Darstellungskomponente (siehe Abbildung 6.1) akzeptierten Wörter der Eingabesprache. Die Meta-Symbole und die Produktionsregeln der Grammatik beschreiben die Syntax.

Benutzerschnittstellen können durch eine reine BNF-Notation nur ungenügend beschrieben werden. Eine übliche Erweiterung besteht darin, die Regeln der Grammatik mit semantischen Aktionen zu assoziieren. Diese Aktionen (meistens als Prozeduren formuliert) rufen das Anwendungssystem auf, um die vom Benutzer gewünschten Operationen auszuführen. Ein Beispiel dafür ist die Erweiterung der BNF-Notation zu einer *Aktionsgrammatik* [*Reisner 81*]. Damit werden die möglichen Operationen des Farbgraphikgerätes "ROBART" spezifiziert. Die Grammatikregeln beschreiben komplexe Handlungen des Benutzers wie das Betätigen von Knöpfen, das Steuern eines Zeigeinstrumentes sowie die Eingabe mithilfe der Tastatur. Die Systemreaktionen sind in dieser Grammatik nicht spezifiziert.

Die Idee der Aktionsgrammatik wurde deshalb erweitert, um auch die Systemreaktionen beschreiben zu können. Dies kann mithilfe einer *Mehrfach-Komponenten-Grammatik* (*multiparty grammar*) erfolgen [*Shneiderman 82*]. Mit diesem Vorschlag können in einer Grammatik die Aktionen von mehreren Komponenten (z.B. Benutzer, Programmsysteme) eines interaktiven Systems spezifiziert werden. Die Meta-Symbole der Grammatik werden um Marken erweitert, die die agierende Komponente bezeichnen. In dem Spezialfall einer Zwei-Komponenten-Grammatik, die beispielsweise den Dialog zwischen einem Benutzer und einer Maschine beschreibt, können in einer Regel der Grammatik die Eingaben des Benutzers und die Reaktionen des Systems spezifiziert werden. Die Meta-Symbole werden mit den jeweiligen Aktionen (Benutzer oder Maschine) markiert. Damit eingabespe-

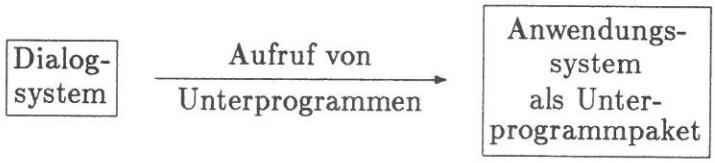

Abbildung 6.3: Externe Kontrolle

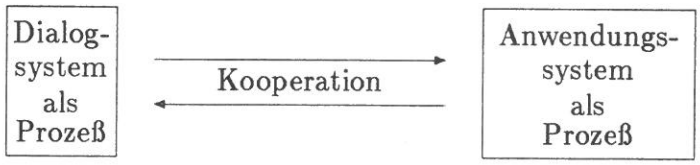

Abbildung 6.4: Verteilte Kontrolle

Zur Dialogkontrolle gehört ebenfalls eine Kontrolle über die Form der Darstellung der Anwendungsdaten und die Reihenfolge von gültigen Benutzeraktionen. Bei der *expliziten Darstellungskontrolle* spezifiziert das Anwendungssystem die Reihenfolge der Darstellung von Anwendungsdaten und die gültigen Folgen von Eingabeaktionen durch den Benutzer, während bei der *impliziten Darstellungskontrolle* von dem Anwendungssystem nur die Daten und gültigen Mengen von Benutzeraktionen angegeben werden. Jede Ordnung über die Reihenfolge der Aktionen und Datendarstellung wird durch das Dialogsystem induziert. Die implizite Kontrolle hat gegenüber der expliziten den Vorteil, daß das Anwendungssystem die Behandlung von Fehlern und Ausnahmebedingungen, die im Dialog mit dem Benutzer auftreten können, nicht vorsehen muß.

6.1.2 Klassifikation von Verwaltungssystemen

In diesem Abschnitt werden einige Verwaltungssysteme für interaktive Schnittstellen sowie Dialogsysteme vorgestellt, die entweder aufgrund ihrer Fähigkeiten im Leistungsumfang einem Verwaltungssystem gleichkommen oder als eines der Werkzeuge angesehen werden können, aus denen sich ein Verwaltungssystem zusammensetzt (s.o.).

Diese Systeme können anhand der im vorherigen Abschnitt aufgestellten Kriterien bewertet werden. Ihre Klassifikation wird nach einem Schema er-

ablauf, können drei Formen unterschieden werden.

In der ersten Form wird das Dialogsystem als Unterprogrammpaket angesehen (siehe Abbildung 6.2). Die Kontrolle über den Dialogablauf liegt damit ausschließlich bei dem Anwendungssystem. Diese aus der Sicht des Anwendungssystems *interne Kontrolle* hat mehrere Nachteile. Die Anwendungsschnittstelle ist nur in Form von Unterprogrammaufrufen spezifiziert. Dadurch wird die konzeptuelle Trennung zwischen Dialog- und Anwendungssystem nicht im Anwendungssystem sichtbar. Das Dialogsystem hat keinen Einfluß auf den Dialogablauf, es kann immer nur lokal in Form von Unterprogrammen arbeiten. Eine fehlerhafte Benutzung des Dialogsystems von der Anwendung kann nur schwer vermieden werden.

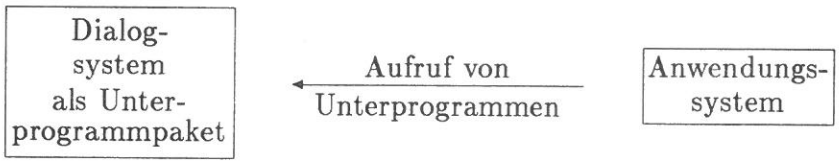

Abbildung 6.2: Interne Kontrolle

Die zweite Form der Dialogkontrolle ist eine völlige Umkehrung der ersten Form. Das Anwendungssystem wird als Unterprogrammpaket angesehen (siehe Abbildung 6.3). Dadurch wird die Kontrolle über den Dialogablauf nur durch das Dialogsystem ausgeübt. Diese aus der Sicht des Anwendungssystems *externe Kontrolle* besitzt die oben erwähnten Nachteile der internen Kontrolle nicht. Dafür ist bei der externen Kontrolle von Nachteil, daß das Anwendungssystem keinerlei Möglichkeit erhält, den Dialog zu beeinflussen. Weiterhin muß das Dialogsystem eine vollständige Spezifikation des Kommunikationsablaufs zwischen dem Benutzer und dem Anwendungssystem besitzen. Diese Spezifikation muß alle möglichen Benutzerinteraktionen vorsehen.

Die dritte Form gestattet sowohl dem Dialogsystem als auch dem Anwendungssystem einen Einfluß auf die Dialogkontrolle. Sie wird deshalb auch als *verteilte Kontrolle* bezeichnet. Dabei werden das Dialogsystem und das Anwendungssystem als kooperierende Prozesse angesehen, die über die Anwendungsschnittstelle eine verteilte Kontrolle des Dialogs ausüben (siehe Abbildung 6.4). Die Nachteile der internen und der externen Kontrolle werden von dieser Form der Dialogkontrolle aufgehoben. Weiterhin können mit ihr die beiden ersten Kontrollformen nachgebildet werden.

Die *Darstellungskomponente* ist für die externe oder graphische Darstellung gegenüber dem Benutzer verantwortlich. Sie erzeugt die graphische Ausgabe auf dem Bildschirm und liest die vom Benutzer bedienten Eingabegeräte. Im Rahmen des oben eingeführten Sprachmodells realisiert sie die lexikalische Ebene und bildet die Wörter der Eingabesprache (in Form von Eingabesignalen) in einer angemessenen internen Darstellung ab. Analog werden die Wörter der Ausgabesprache (in Form von Graphikelementen) in entsprechende graphische Darstellungen transformiert.

Die *Dialogkontrolle* definiert die Struktur des Dialoges zwischen dem Benutzer und dem Anwendungssystem. Sie empfängt eine lineare Folge von Eingabewörtern von der Darstellungskomponente und eine lineare Folge von Ausgabewörtern von der Anwendungskomponente. Mithilfe dieser Daten wird eine Dialogsteuerung vorgenommen. Diese Komponente repräsentiert im Sprachmodell die syntaktische Ebene.

Die *Anwendungskomponente* definiert die Schnittstelle zwischen dem Dialog- und dem Anwendungssystem. Sie kann direkt mit dem Anwendungssystem kommunizieren. Dabei werden die Daten des Anwendungssystems in eine dem Dialogsystem angemessene interne Darstellung überführt und umgekehrt die Aktionen des Benutzers auf die Anwendungsdaten abgebildet. Die Anwendungskomponente erzeugt aus der internen Darstellung der Anwendungsdaten Informationen, die die Dialogkontrolle bei der Durchführung ihrer Aufgaben unterstützen. Weiterhin führt sie eine semantische Überprüfung der Benutzereingaben durch. Sie kann im Sprachmodell als Realisierung der semantischen Ebene angesehen werden.

Neben der detaillierten Struktur des Dialogsystems ist ein weiterer wichtiger Punkt die Schnittstelle zwischen dem Dialogsystem und dem Anwendungssystem. Die Struktur der Anwendungsschnittstelle wird entscheidend durch die Form des Daten- und Kontrollflusses zwischen dem Anwendungs- und dem Dialogsystem bestimmt [*Draper & Norman 85, Green 84, Hayes et al. 85*]. Der Datenfluß und -austausch kann sich entweder an den Bedürfnissen des Dialogsystems oder an denen des Anwendungssystems orientieren. Dabei ist eine Abstraktion auf Anwendungsniveau vorzuziehen, da die Kommunikation in Form von Einheiten beschrieben wird, die das Anwendungssystem vom Benutzer benötigt oder ihm vermitteln will. Die Abstraktion auf dem Niveau des Dialogsystems spezifiziert die Daten in Form von Einheiten aus der Benutzerschnittstelle.

Für den Kontrollfluß zwischen dem Anwendungs- und dem Dialogsystem, und damit für den Einfluß des Anwendungssystems auf den Dialog-