



KfK 5001
April 1992

Visualisierung in komplexen Systemen und deren Anwendung im Umweltschutz

R. Denzer
Institut für Datenverarbeitung in der Technik
Projekt Schadstoff- und Abfallarme Verfahren

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe

**Institut für Datenverarbeitung in der Technik
Projekt Schadstoff- und Abfallarme Verfahren**

KfK 5001

**Visualisierung in komplexen Systemen und
deren Anwendung im Umweltschutz**

Ralf Denzer

**vom Fachbereich Informatik der Universität Kaiserslautern
genehmigte Dissertation**

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Als Manuskript gedruckt
Für diesen Bericht behalten wir uns alle Rechte vor

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

Zusammenfassung

Visualisierung in komplexen Systemen und deren Anwendung im Umweltschutz

Bei der Führung technischer Anlagen sowie bei Überwachung und Schutz der natürlichen Umwelt fallen große Mengen örtlich verteilter Daten an. Diese Daten sind von unterschiedlichster Art, liegen in der Regel in unterschiedlicher Darstellung auf beliebigen Systemen vor und haben die unterschiedlichste Bedeutung. Die verschiedenen auf diese Daten angewandten Methoden sind oft ebenso von komplexer Natur.

Insbesondere bei Informationssystemen für den Umweltschutz entsteht dabei eine neue Qualität in der Gestaltung von Benutzerdialogen. Die Daten, die nämlich an einem Umweltarbeitsplatz zur Verfügung gestellt werden müssen, können über ein großes, beliebig verteiltes Rechnernetz verstreut sein, wobei nicht immer von vorne herein klar sein muß, wo ein bestimmtes Datum zu finden ist. Das Rechnersystem selbst stellt also ein komplexes verteiltes System dar, welches auf dem komplexen verteilten System Umwelt operiert. Alleine das Auffinden und der geordnete Zugriff auf bestimmte Daten ist - so einfach das klingen mag - keine triviale Angelegenheit, wenn man sich ein bundesweit operierendes Informationssystem für den Umweltschutz vorstellt.

Die Visualisierung *des* komplexen Systems Umwelt wird durch die verwendeten komplexen Rechnersysteme zur Visualisierung *in* einem komplexen System.

Zwei grundsätzliche Fragestellungen sind hier offen. Erstens liegt nach wie vor eines der Hauptprobleme in der technischen Schnittstelle zwischen dem User-Interface-System und der Anwendung. Obwohl auf diesem Gebiet in den letzten Jahren viel geschrieben wurde und in letzter Zeit einige begrüßenswerte Ansätze existieren, täuscht dies nicht darüber hinweg, daß dies nach wie vor ein konzeptionell nicht befriedigend gelöstes Problem ist. Zweitens ergibt sich die Frage, wie Benutzerschnittstellen für solche komplexen Systeme derart aufgebaut werden können, daß sie ein quasi-intelligentes Verhalten besitzen.

In der vorliegenden Arbeit wird eine neuartige Architektur für graphische Benutzeroberflächen entwickelt. Sie bietet einen konzeptionell-theoretischen Rahmen zum effizienten und flexiblen Aufbau von Endbenutzerschnittstellen in den genannten Systemen. Dabei ist die Integration eines der Hauptmerkmale.

Ein Kernpunkt dieser Architektur ist ein *neuartiges Kommunikationskonzept* für offene, verteilte Systeme, welches dazu dient, im Netz verstreute Daten ohne Aufwand on-line in eine lokale Benutzerschnittstelle zu integrieren. Dieses Kommunikationskonzept ist nicht auf graphisch-interaktive Anwendungen beschränkt und kann allgemein eingesetzt werden. Ein weiteres Charakteristikum ist die Zusammenfassung von Darstellungs- und Anwendungskomponenten entgegen dem Seeheim-Modell. Dies wird als *integrierte Modellierung* bezeichnet. Der Aufbau der Benutzerschnittstelle erfolgt unter Verwendung objekt-orientierter und regel-basierter Methoden.

Abstract

Visualization in Complex Systems and its Application in Environmental Protection

Both in the domains of guidance and control of technical systems and the protection of our natural environment, there are large sets of distributed data. These data are of many different kinds and may have different meanings. The methods operating on the data are often also of complex nature.

Especially for environmental information systems there is a new quality in the design of end user dialogues. The data needed at a workplace may be distributed over a large network, where it is not always obvious where the data may be found. The computer network itself is a complex system which operates on the data of the complex system "environment". Even finding the right data may be difficult.

The visualization of the complex system "environment" therefore is some kind of visualization *within* a complex system.

There are two questions open today. Firstly, there is still the difficulty of the interface between the user interface system and the application. Although there have been many approaches in the past few years, this problem is far from being solved. Secondly, there is the problem how to design user interfaces for such very complex systems.

With this work, we present a new architecture for graphical user interfaces. The architecture shows a conceptual framework for the design of flexible end user interfaces for complex systems.

The architecture is based on a *new communication concept* for open distributed systems which is capable to integrate remote data very easily into the user interface. The communication concept is a general one and therefore not limited to user interface implementation. The second main concept is the integration of representation and application components in opposition to the Seeheim model. This is called *integrated modeling*. The implementation of the user interface is done using a combination of object orientation and rule based methods.

Inhaltsverzeichnis

- 1. Einleitung 1**
 - 1.1. Transformation in den graphisch-optischen Kanal 1
 - 1.2. Verdeutlichung von Aspekten 2
 - 1.3. Gewinnung von Information 3
 - 1.4. Interaktivität 3
 - 1.5. Komplexe Systeme 5
 - 1.6. Ziele und Konzepte 7
 - 1.7. Technischer Stand 10
- 2. Konzepte 12**
 - 2.1. Übergreifende Konzepte 12
 - 2.2. Spezifische Gesichtspunkte bei der Führung technischer Anlagen . . . 18
 - 2.3. Spezifische Gesichtspunkte bei Umweltinformationssystemen 26
- 3. Grundlagen 39**
 - 3.1. Offene Systeme 39
 - 3.1.1 Unbegrenztheit 39
 - 3.1.2 Zugang 40
 - 3.1.3 Heterogenität 40
 - 3.1.4 Autonomie 41
 - 3.1.5 Dezentralität 41
 - 3.1.6 Zusammenarbeit in offenen Systemen 42
 - 3.1.7 Ansätze zur Kooperation in offenen Systemen 43
 - 3.2. Verteilte Systeme 44

3.2.1	Konzepte	45
3.2.2	Objekt-orientierte Vorgehensweise	47
3.2.3	Anwendbarkeit der Konzepte	48
3.3.	Computergraphik und User Interface Management	49
3.3.1	Objektorientierte Graphik	50
3.3.2	User Interface Management	53
3.4.	Wissensbasierte Anwendungen	54
3.4.1	Wissensbasierte Methoden in der Prozeßtechnik	55
3.4.2	Wissensbasierte Methoden in der Umweltinformatik	58
4.	Eine Architektur für Benutzerschnittstellen komplexer Systeme	60
4.1.	Überblick	60
4.2.	Kommunikationsmodell	67
4.3.	Integrierte Modellierung	74
4.4.	Implementierung	76
5.	Filterungstechniken und Dialoggestaltung	80
5.1.	Eine Benutzerschnittstelle für eine verfahrenstechnische Anlage	80
5.1.1	Prozeß-Adaptiver Dialog	81
5.1.2	Dialogmanagement auf komplexen Prozessen	93
5.1.3	Modellierung des Feuerungsteils	104
5.2.	Eine Benutzerschnittstelle für das Luftgütemeßnetz Kärnten	111
5.2.1	Beispiel zur Oberflächengestaltung	112
5.2.2	Modellierung	120
6.	Diskussion	127
	Literaturverzeichnis	131

1. Einleitung

Visualisieren im puren Sinn des Wortes bedeutet *optisch darstellen*. Nun ist es schon seit Bestehen der Computergraphik deren Sinn, optische Darstellungen DV-gestützt zu erzeugen und nutzbar zu machen. Es wird allerdings erst seit einigen Jahren verstärkt von *Visualisierung* bzw. von *Visualisierungstechniken* gesprochen. Woher kommt dieser neuere Sprachgebrauch?

Unter Visualisierungstechniken werden heute noch primär eine begrenzte Klasse von Verfahren verstanden. Sieht man sich die erste IEEE-Konferenz zu diesem Thema an [1.1], so fällt auf, daß es in einer Vielzahl von Beiträgen um die Darstellung von Flächen oder Volumina bzw. um deren mathematische Grundlagen geht. Nun gehören diese Verfahren mit Sicherheit zu den attraktivsten innerhalb der Computergraphik bzw. der Informatik im allgemeinen und es ist daher kein Wunder, daß sie innerhalb dieses Gebiets derzeit eine maßgebliche Rolle spielen. Dies ist allerdings nur die eine Seite der Medaille.

Es fällt allerdings auch auf, daß zunehmend Visualisierungsverfahren für eher "nicht-klassische" Gebiete der Computergraphik-Anwendungen entwickelt werden. Als Beispiele seien hier genannt: die dynamische Visualisierung eines Telefonnetzwerks [1.2], die graphische Visualisierung des letzten Theorems von Fermat [1.3] oder die Verwendung paralleler Koordinaten zur Visualisierung multi-dimensionaler Geometrien [1.4]. Gerade das letzte Beispiel zeigt, wie weit entfernt heute verwendete Visualisierungsverfahren von klassischen Gebieten der Computergraphik liegen können.

1.1 Transformation in den graphisch-optischen Kanal

Bei genauerem Hinsehen fällt der erste Zweck der Verwendung von Visualisierungsverfahren auf: Visualisierung ist die *Transformation von Information* aus ihrer ursprünglichen Form in eine für den Menschen leichter erkennbare Form.

Eine der ersten und zugleich für das Verständnis der Naturwissenschaften und der Mathematik wichtigsten visuellen Transformationen ist die Darstellung von Zeitverläufen über einer Zeitachse. Hiermit ist heute nahezu jeder vertraut. Diese Transformation spiegelt auf einfache Weise ihren Nutzen wider. Durch sie wird es einfach, eine abklingende Sinusschwingung "auf einen Blick" zu erkennen, was jedermann anhand der ursprünglichen Daten nahezu unmöglich ist. Den analogen Sinn verfolgen auch komplexere Visualisierungsverfahren.

Es ist allgemein bekannt, daß der graphisch-optische Kanal in vielen Fällen eine weitaus höhere Kapazität besitzt als die Kanäle Text oder Sprache. Visualisierung dient also zunächst der Steigerung des Informationsflusses zwischen Rechner und Mensch.

1.2 Verdeutlichung von Aspekten

Ein weiteres Merkmal der Visualisierung liegt in der Art und Weise, wie die Ursprungsinformation in graphische Information transformiert wird. Oft dient diese Transformation der Verdeutlichung bestimmter Aspekte der Ursprungsinformation: aus einer komplexeren Informationsmenge werden bestimmte interessierende Einzelaspekte unter Verwendung graphischer Methoden herausgeschnitten.

Die ursprüngliche Information wird also zu *einem gegebenen Zweck* manipuliert, um bestimmte Charakteristika innerhalb der Information hervorzuheben (Abb. 1.1). Die verwendeten Visualisierungsverfahren werden zu diesem Zweck optimiert.

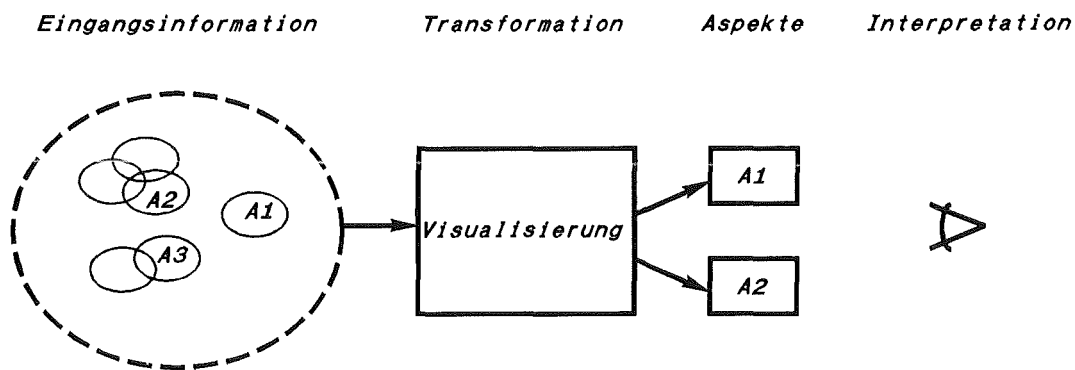


Abb. 1.1: Hervorhebung von Aspekten

Nun kann man einwerfen, daß es eine Vielzahl von Verfahren gibt, um solche Informationsfilterungen durchzuführen, so z.B. aus der Systemtheorie (Korrelationen) oder aus der künstlichen Intelligenz (Diagnose). Wozu soll man also für diesen Zweck Visualisierungsverfahren einsetzen? Hierzu zwei Antworten.

1.) Es gibt eine Reihe von Anwendungen, bei denen der Zugang zur Information auf graphische Art der natürlichste ist. Als Beispiel seien thematische Karten genannt [1.5]. Hier entspricht ein graphisches Verfahren der Arbeitsweise, die auch ohne Datenverarbeitung angewandt wird.

2.) Mathematische oder wissensbasierte Verfahren haben einen entscheidenden Nachteil: sie benötigen ein wie auch immer geartetes Modell der Realität, damit sie überhaupt eingesetzt werden können. Darüber hinaus muß das Modell von entsprechender Güte sein, um valide Aussagen machen zu können.

Die vorliegende Arbeit beschäftigt sich mit komplexen Systemen wie verfahrenstechnischen Anlagen und Umweltprozessen. In diesen Fällen ist es derzeit oft noch nicht möglich oder zu aufwendig bzw. zu teuer, valide Modelle anzugeben, anhand derer eine automatische Interpretation eines Systemzustands erfolgen kann. In vielen Bereichen sind allenfalls Teilmodelle vorhanden.

Bei komplexen Systemen können Visualisierungsverfahren dazu beitragen, gewisse Teilaspekte anhand von Teilwissen herauszuarbeiten und dem Menschen zugänglich zu machen. Hierdurch wird die Interpretation komplexer Zusammenhänge durch den menschlichen Experten unterstützt.

Demnach wird unter Visualisierung in der Folge eine Menge von Verfahren und Methoden verstanden, die eines gemeinsam haben: sie gehen von einer Menge von Eingangsinformation beliebiger Art aus und transformieren diese Information in optisch-graphische Information, wobei der Zweck der Transformation ein möglichst rasches, effizientes und einfaches Erkennen bestimmter Aspekte der Ursprungsinformation ist.

Die Stärke von Visualisierungsverfahren für diesen Zweck liegt darin, daß sie - im Vergleich zu modell-basierten Verfahren in obigem Sinn - einfacher zu verwenden, zu implementieren und zu handhaben sind. Das heißt nicht, daß sie von einfacher Natur sind; aber sie sind mit vertretbarem Aufwand in die Praxis umsetzbar, was bei modell-basierten Verfahren oft nicht der Fall ist.

1.3 Gewinnung von Information

Visualisierung kann auch dazu dienen, aus einer Menge von Informationen neue Informationen abzuleiten bzw. - genauer gesagt - auf mögliche Zusammenhänge innerhalb der Informationen Hinweise zu geben. Hierzu werden Daten auf unterschiedlichste Art verschnitten, korreliert und dargestellt.

Visualisierungssysteme selbst erzeugen dabei im allgemeinen keine neue Information. Sie helfen aber dabei, Induktionsprozesse anzustoßen, wenn der Benutzer des Systems die präsentierte Information mit seinem Hintergrundwissen in Verbindung bringt.

Eine sinnvolle Anwendung dieser Form der Visualisierung ist die Verschneidung multidimensionaler Umweltdaten. Leider werden die Möglichkeiten, die die Visualisierung multidimensionaler Daten für das Erkennen von Korrelationen bietet, in der Umweltinformatik noch nicht konsequent untersucht geschweige denn genutzt.

1.4 Interaktivität

Die vorangegangenen Abschnitte haben verdeutlicht, daß Visualisierungsverfahren als Hilfsmittel bei der Interpretation von Informationen durch den Menschen dienen.

Visualisierung bezieht explizit den Menschen in ein verwendetes Verfahren ein und es soll nicht die Aufgabe des Verfahrens sein, die Interpretation der Information vorzunehmen.

Dies ist ein kritischer Punkt bei jeglicher Visualisierung, denn durch die Transformation der Ursprungsinformation in die graphische Darstellung geschieht schon mehr oder minder eine Interpretation. Ein typisches Beispiel hierfür ist die Verwendung der Signalfarbe Rot zur Hervorhebung von Störfällen bei technischen Systemen. Im genannten Fall ist diese Interpretation durch die Farbgebung erwünscht. Allerdings mag in anderen Fällen Art und Weise der Farbgebung, Schraffur usw. zu einer unerwünschten Vorinterpretation von Information durch das Visualisierungssystem führen. Hierüber muß sich sowohl der Entwickler als auch der Anwender eines Visualisierungssystems im klaren sein.

Dieses Problem kann dadurch gemildert werden, daß man es soweit als möglich dem Bediener überläßt, wie er bestimmte Informationen betrachten möchte. Es soll also weitestgehend die Entscheidung des Benutzers und nicht die Entscheidung des Systems sein, welche Teilaspekte der Information auf welche Art präsentiert werden. Visualisierungssysteme sind daher in der Regel nur dann sinnvolle Werkzeuge für Anwender, wenn sie hochgradig interaktiv sind.

Interaktivität bedingt zunächst gute Mensch-Maschine-Schnittstellen. Es ist allgemein bekannt, daß schlechte Benutzerschnittstellen die Akzeptanz von Software auf ein Minimum reduzieren. Benutzerschnittstellen benötigen ein Maximum an Komfort, Transparenz, Übersichtlichkeit und Flexibilität.

Bei komplexen Systemen hat eine Benutzerschnittstelle darüberhinaus die schwierige Aufgabe zu erfüllen, die Komplexität des Systems beherrschbar zu machen. Das bedeutet, daß besondere Sorgfalt darauf gerichtet werden muß, wie der Benutzer an die Information herangeht und wie er in der Benutzerschnittstelle navigiert.

Ein Kernpunkt und ein Ziel der vorliegenden Arbeit ist die Beantwortung der Frage, wie man die Navigation in der Benutzerschnittstelle - in der Folge *Dialog* genannt - unter Verwendung von Hintergrundinformation optimieren kann und welche informationstechnischen Konzepte hierzu sinnvoll eingesetzt werden können. Die Optimierungsziele für die betrachteten Systemklassen werden in Kap. 2 näher erläutert. In der Hauptsache handelt es sich bei diesen Zielen um

- die Geschwindigkeit des Verstehens,
- die Geschwindigkeit des Zugriffs und
- die Auffindbarkeit von Information.

Die Geschwindigkeit des Verstehens ist insbesondere bei technischen Systemen von Bedeutung. Sie kann z.B. durch Informationsfilterung und Hervorhebung wichtiger Zustände oder Ereignisse erhöht werden. Hierzu wird Hintergrundwissen über den technischen Prozeß in die Dialogsteuerung aufgenommen. Die Geschwindigkeit des

Zugriffs auf Information soll dadurch verbessert werden, daß unnötige Interaktionen unterdrückt werden. Dies führt zur Modifikation des Dialogs zur Laufzeit durch Adaption an einen gegebenen Systemzustand. Auch hierzu wird Hintergrundinformation benötigt. Zur einfachen Auffindbarkeit von Information wird Wissen über das informationsverarbeitende System in den Dialog integriert. Darüberhinaus wird die Information über ein Objekt soweit als irgend möglich direkt bei dem Objekt selbst zur Verfügung gestellt.

Nach Einführung der betrachteten Systemklassen in Kap. 1.5 und einer knappen Definition der Ziele der vorliegenden Arbeit werden die Konzepte für den Dialog und die Dialog-Software in Kap. 2 detailliert abgeleitet.

1.5 Komplexe Systeme

Zwei Klassen von komplexen Systemen werden betrachtet: chemisch-verfahrenstechnische Anlagen und Informationssysteme für den Umweltschutz.

Während die Visualisierungsverfahren für die beiden Systemklassen zum Teil erheblich differieren, existieren große Ähnlichkeiten bei den Anforderungen an Dialoggestaltung und Dialogsoftware. Dies ist auch der Schwerpunkt der folgenden Betrachtungen.

Es soll nicht verschwiegen werden, daß die in Kap. 4 vorgestellte Architektur XAVIA (*Expert Advanced Visualization Architecture*) ursprünglich für Benutzerschnittstellen technischer Anlagen entworfen wurde. Das Ziel war dabei, durch Verwendung "intelligenter Dialoge" Störfälle und Fehlfunktionen technischer Anlagen schneller erkennen und beseitigen zu helfen.

Störungen in solchen Anlagen führen in vielen Fällen zu Umweltbeeinträchtigungen. Man muß nicht einmal die hinlänglich bekannten Chemieunfälle der letzten Jahre betrachten, auch im kleinen kann man sich dies leicht klar machen. Verbrennungsprozesse (z.B. Heizungen) besitzen i.d.R. einen Betriebspunkt, der sowohl energetisch als auch emissionsmäßig optimal ist. Jede Abweichung vom optimalen Punkt wirkt sich negativ auf das Emissionsverhalten aus. Daher sollen Störungen möglichst vermieden bzw. schnell beseitigt werden. Technische Anlagen besitzen zu diesem Zweck eine Vielzahl von automatischen Steuerungen und Regelungen [1.6].

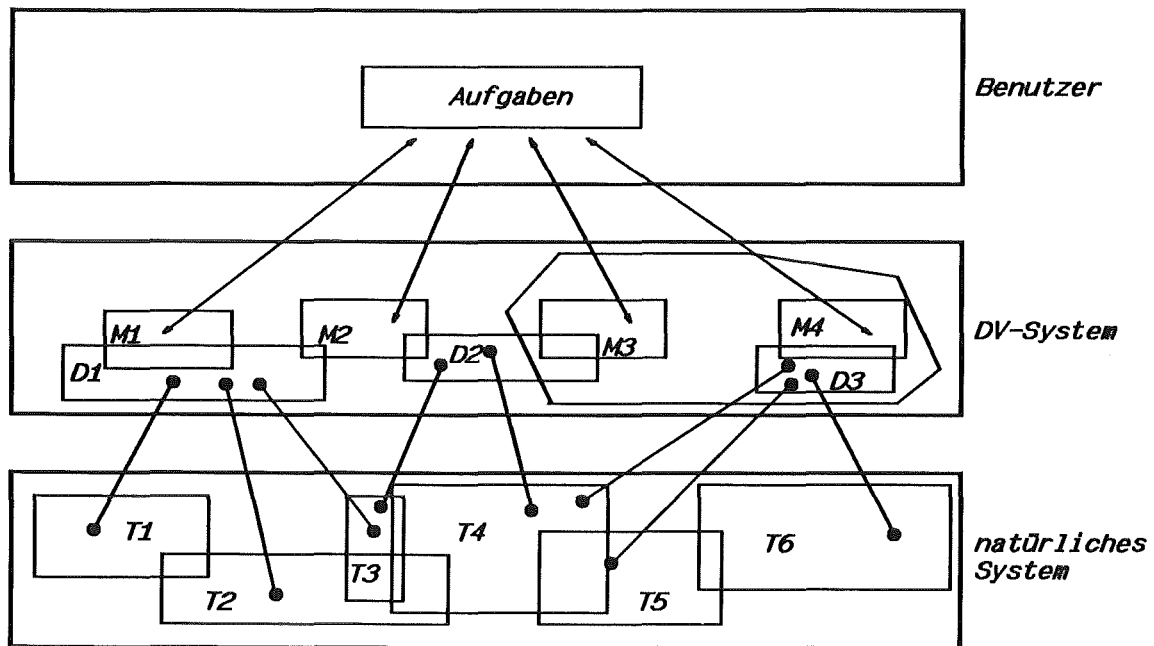
Allerdings ist es mit rein systemtheoretischen Mitteln heute noch in den seltensten Fällen möglich, die Zustände einer verfahrenstechnischen Anlage vollständig zu erfassen und zu kontrollieren. Die Menge der möglichen Störursachen solcher Anlagen ist einfach zu groß und viele der Verkopplungen sind äußerst schwierig zu beschreiben. Aus diesem Grund ist der Operateur nach wie vor das bestimmende Element in der Schaltwarte, wenn es darum geht, Störfälle zu beheben oder Strategien zu entwickeln. Daher kommt der Mensch-Maschine-Schnittstelle in diesem Fall eine so große Bedeutung zu.

Im Lauf der Entwicklung der Architektur stellte sich heraus, daß einige der betrachteten Konzepte auch auf Umweltinformationssysteme anwendbar sind. Am einfachsten erscheint dabei eine Umsetzung bei stark technisch orientierten Teilen von Umweltinformationssystemen wie z.B. Luftgütemeßnetzen. (Ein Luftgütemeßnetz besitzt im Prinzip alle Eigenheiten eines verteilten technischen Prozesses).

Unter einem *komplexen System* wird im Rahmen der folgenden Ausführungen ein System verstanden, das

- aus einem *komplexen natürlichen System*,
- einem *komplexen DV-System* und
- dem *Benutzer* des DV-Systems

besteht. Dabei werden als Klassen natürlicher Systeme die Systeme "Umwelt" und "technische Prozesse" angenommen. Die DV-Systeme können aus beliebigen informationsverarbeitenden Komponenten bestehen.



Legende:

T_i : Teilsysteme

D_i : Datenmengen

M_i : Methoden

Abb. 1.2: Komplexes System

Das natürliche System zeichnet sich durch folgende Eigenschaften aus:

- Es ist örtlich verteilt.
- Es beinhaltet zeitlich und u.U. örtlich variable Information.
- Die Informationsmengen sind groß und stark verkoppelt.
- Das System besitzt eine Eigendynamik.
- Es können spontane Ereignisse auftreten.
- Nur in Teilen sind valide Modelle bekannt.

Das DV-System wird durch folgende Eigenschaften beschrieben:

- Es ist örtlich und funktional verteilt.
- Es existieren große, zeitlich variable Datenmengen variabler Größe.
- Die eingesetzten Methoden sind örtlich verteilt.
- Es können spontane Ereignisse auftreten.
- Das DV-System besteht aus autonomen Subsystemen.

Entscheidend für die Komplexität des Gesamtsystems sind eine Vielzahl von Verkopplungen und gegenseitigen Abhängigkeiten der einzelnen Datenmengen und Teilsystemen untereinander (Abb. 1.2) sowie die zum Teil nicht vorhersehbare Eigendynamik.

Innerhalb des Systems sollen Aufgaben durchgeführt werden, die Daten und Methoden aus beliebigen Teilsystemen erforderlich machen. Dieser Tatsache muß ein Visualisierungssystem bzw. ein Dialogsystem Rechnung tragen.

Abb. 1.2 zeigt ein Modell eines solchen Systems. Dieses Modell gibt allerdings nur einen Teil der heutigen Realität wieder. Während bei technischen Prozessen heute durchweg die verteilte Natur der Systeme berücksichtigt wird, ist dies bei Informationssystemen im Umweltschutz noch nicht der Fall.

1.6 Ziele und Konzepte

Die Hauptziele der vorliegenden Arbeit liegen in der Gestaltung von Benutzeroberflächen für komplexe, zeitlich variable Informationsmengen. Die Geschwindigkeit des Verstehens und der Interaktion und die Auffindbarkeit von Daten in der komplexen Informationsmenge spielen, wie schon erwähnt, die Hauptrolle.

Diese Ziele sollen dadurch erreicht werden, daß Wissen über das zu visualisierende System in die Benutzerschnittstelle eingebettet wird. Das Wissen dient dazu, die Benutzerschnittstelle zur Laufzeit an bestimmte Situationen zu adaptieren.

Die primären Ziele implizieren eine Reihe von sekundären aber wichtigen Entwicklungszielen, ohne die die Aufgabe nicht gelöst werden konnte. Diese werden in der Folge kurz erläutert und in Kap. 2 begründet.

Eine wichtige Eigenschaft der betrachteten Systeme ist die Tatsache, daß sie sowohl örtlich als auch funktional verteilt sind. Sowohl bei der Prozeßdatenverarbeitung als auch bei Umweltinformationssystemen sind die zu visualisierenden Informationen über eine Reihe von Subsystemen verstreut. Auch die eingesetzten Methoden sind i.d.R. auf mehreren Rechnerknoten verteilt. Dabei ist davon auszugehen, daß jedes der autonomen Subsysteme eine spezifische Aufgabe durchführt.

Auf den an sich schon als komplex zu betrachtenden Systemen "technischer Prozeß" bzw. "Umwelt" operieren also komplexe, verteilte DV-Systeme. Läßt man diese Tatsache außer acht, so wird man der Problemstellung nicht umfassend gerecht. Aus der Visualisierung komplexer Systeme wird dadurch die *Visualisierung in einem komplexen System*.

Daher wird in der Folge davon ausgegangen, daß die darzustellenden Informationen über ein beliebiges Netzwerk in beliebigen, unabhängigen Teilsystemen vorhanden sind. Ein Grundmerkmal der in Kap. 4 vorgestellten Architektur ist daher ihr *verteilter Ansatz*.

Das wichtigste Merkmal der einzelnen Softwaresysteme im verteilten System ist deren *Autonomie*. Jedes Teilsystem löst eigene Aufgaben unter eigenen Randbedingungen. Dennoch sollen im Gesamtsystem Aufgaben bearbeitet werden, die die Zusammenarbeit der Teilsysteme erfordern. Das zweite Grundmerkmal der Architektur ist daher die *Offenheit*. Beliebige Teilsysteme arbeiten autonom und die Zusammenarbeit erfolgt aufgrund einer gemeinsamen Übereinkunft über die *Kommunikationsmechanismen*.

Um die beiden vorstehenden Randbedingungen zu befriedigen, wurde ein neuartiges Kommunikationskonzept entwickelt. Dieses sollte einen möglichst einfachen und transparenten Mechanismus zur online-Kopplung von Anwendungen bieten.

Das Kommunikationskonzept stellt sog. *virtual links* (virtuelle Verbindungen) zwischen Objekten unterschiedlicher Anwendungen zur Verfügung. Dabei wird nicht festgelegt, welche Objektbeschreibung eine einzelne Applikation verwenden muß. Jede Applikation ist hier vollständig frei in der Wahl ihrer Mittel, d.h. Programmiersprachen und Werkzeuge.

Eine weitere Fragestellung liegt darin begründet, wie man ein zugleich effektives, transparentes und einfaches *Kommunikationsmanagement* innerhalb des Gesamtsy-

stems durchführen kann. Diese Fragestellung führte zu der Einführung sog. *Signifikanzkriterien*, welche den Austausch von Information steuern.

Aufbauend auf lokale und externe Information, die über den Kommunikationsmechanismus importiert wird, muß die Anwendung und die Benutzerschnittstelle modelliert werden. In die Benutzerschnittstelle soll Wissen über das zu visualisierende System aufgenommen und zur Dialogsteuerung verwendet werden. Hier wirkte sich die Komplexität der betrachteten Systemklassen auf den Entwurf des Dialogsystems aus. Die zu beantwortende Fragestellung lautet: wenn die zu visualisierenden Systeme derart komplex sind, wie kann man dann eine Visualisierungssoftware derart gestalten, daß ihre eigene innere Komplexität nicht überproportional ansteigt und sie zudem für den Designer der Schnittstelle überschaubar bleibt?

In diesem Zusammenhang ist außerdem die Frage von Bedeutung, wie man das verwendete Wissen sinnvoll modellieren kann.

Die beiden vorangegangenen Fragen führten zu einem Ansatz für die Visualisierungsarchitektur, die sich durch eine Kombination zweier Methoden auszeichnet: der objekt-orientierten Programmierung und der regel-basierten Programmierung. Genauer gesagt werden Objekte und Objektklassen verwendet, deren Methoden Regeln sind.

Die Objekte bzw. Objektklassen sollen auf möglichst natürliche Weise die Realität wiedergeben, damit die interne Softwarestruktur transparent wird. Dies wird durch ein Konzept erreicht, das *integrierte Modellierung* genannt wird. Im Gegensatz zur üblichen Vorgehensweise bei User Interface Management Systemen werden Darstellungskomponenten und Applikationskomponenten nicht voneinander getrennt. Objekte, die auf diese Art und Weise modelliert werden, sind vollständig wiederverwertbar, da sie von der Prozeßschnittstelle aufwärts über eventuell vorhandene Zwischenstufen bis hin zu ihrer Darstellung und Interaktion *als Ganzes* betrachtet werden. Es entsteht somit eine hochwertige Objektbeschreibung.

Objektorientierung an sich bietet für die Programmierung schon bedeutende Vorteile wie z.B. Datenkapselung. Der objekt-orientierte Ansatz wird umso mächtiger, wenn man ihn zu einem Ansatz einer *Objektdatenbank* erweitert. Es wird dadurch möglich, Objekte und Objektklassen zu beschreiben und sie bei einer neuen Anwendung zu inkludieren. Die Entwurfsumgebung für Dialoge wird dadurch mit jeder neuen Objektklasse mächtiger.

Durch die abschließende Bemerkung schließt sich der Kreis zu den verteilten Systemen. Will man einen komfortablen und effizienten Entwurf innerhalb eines verteilten Systems und für ein verteiltes System gewährleisten, so kommt man nicht umhin, dies bei der Objektbeschreibung des Dialogsystems zu berücksichtigen. Die hochwertige Objektbeschreibung verliert ihre Mächtigkeit, wenn man weiterhin sämtliche Kommunikation explizit programmieren muß.

Daher wurden sogenannte *externe Attribute* (externe Instanzvariablen) in die Objektbeschreibung des Dialogsystems eingeführt. Das sind Eigenschaften von Objekten, die nicht innerhalb der eigenen Applikation erzeugt und modifiziert werden, sondern von einer im Netz an einer beliebigen Stelle vorhandenen Anwendung aktualisiert werden. Die Initialisierung und Aktualisierung dieser Instanzvariablen geschieht asynchron vollständig automatisch und wird einzig und allein dadurch angestoßen, daß ein externes Attribut in das Laufzeitsystem geladen wird. Zum Datenaustausch zwischen Anwendungen ist daher keinerlei Programmierung mehr vonnöten.

Dieser Abschnitt gab einen kurzen Überblick über die Ziele der XAVIA-Architektur und stellte schon einige der verwendeten Konzepte vor. Eine sinnvolle Umsetzung im Hinblick auf die Offenheit des Gesamtsystems und die Anwendbarkeit der Architektur führte zur Einführung mehrerer Schichten. Die Basisschicht ist eine netzwerkunabhängige Kommunikationsschicht, die eine Verwaltung der Meldungen und Kopplungen zu anderen Applikationen vornimmt. Oberhalb dieser Schicht existiert eine Sprachschicht, die die Ankopplung an unterschiedliche Programmiersprachen vornimmt. Eine Hauptschwierigkeit liegt hier in der asynchronen Natur des Kommunikationsmodells, d.h. an der Grenze dieser Schicht zur Anwendung muß eine Synchronisation erfolgen. In der Anwendungsschicht ist jede Applikation frei, ihre eigene Beschreibung der Daten und des Wissens vorzunehmen, d.h. hier kann jede Programmiersprache eingesetzt werden, für die eine Sprachschicht implementiert wird.

In der Anwendungsschicht wurde eine objekt-orientierte Umgebung entworfen und implementiert, mit der adaptive Benutzerdialoge aufgebaut werden können. Hierfür wird derzeit eine Entwicklungsumgebung erstellt, welche allerdings in der vorliegenden Arbeit nicht berücksichtigt werden wird.

Im folgenden Kapitel werden die spezifischen Anforderungen und Randbedingungen betrachtet, die in ihrer Summe zu der beschriebenen Vorgehensweise und zu der Architektur des Kap. 4 führten. Das dabei entstandene Modell der Informationsverarbeitung und Visualisierung in komplexen, verteilten, offenen Systemen ist von theoretischer Natur, wurde aber anhand praktischer Randbedingungen spezifiziert.

1.7 Technischer Stand

Der Stand der Prototyp-Implementierung der XAVIA-Architektur im April 1991 stellt sich folgendermaßen dar: die Realisierung der Graphik konzentrierte sich zunächst auf symbolische Darstellungen. Vektorielle Information wurde nicht betrachtet. Dies war im ersten Ansatz nicht nötig, da zur Visualisierung verfahrenstechnischer Prozesse (sieht man von Kurvendarstellungen ab) eine symbolische Darstellung in Form von Fließbildschemata ausreicht. Außerdem liegt wie schon erwähnt der Schwerpunkt der Arbeit auf dem Gebiet der Dialoggestaltung. Der Prototyp ist dazu geeignet, Benutzerschnittstellen und Visualisierungssysteme für technische Anlagen aufzubauen, wobei der Entwurf in einem verteilten System geschieht.

Bei Umweltinformationssystemen kann der Prototyp als Benutzerschnittstelle eingesetzt werden, speziell bei Luftmeßnetzen existieren große Analogien zur Visualisierung verfahrenstechnischer Systeme, so wie sie in dieser Arbeit angegangen wird.

2. Konzepte

Die folgenden Abschnitte sollen detailliert in die spezifischen Probleme der Dialoggestaltung und der Dialogsoftware bei den betrachteten Systemen einführen. Dabei soll in der Einführung versucht werden, gemeinsame Charakteristika herauszustellen, die für technische Benutzerschnittstellen bzw. Benutzerschnittstellen von Umweltinformationssystemen gelten. Begriffe wie "verteilt System", "Offenheit" und ähnliches werden eingeführt, so wie sie in der vorliegenden Arbeit zu verstehen sind. Sie sind für das Verständnis der Zusammenhänge grundlegend. In den beiden folgenden Abschnitten werden für die beiden Systemklassen spezifische Aspekte erörtert. In Kap. 3 wird dann auf die theoretischen Grundlagen der verwendeten Begriffe eingegangen.

Dabei muß davon ausgegangen werden, daß derzeit real lediglich die Anforderungen bei der Führung technischer Prozesse abgeleitet werden können. Die Überlegungen, die bezüglich Umweltinformationssystemen angestellt werden, führen über die derzeitige Realität hinaus, da es eigentlich in diesem Bereich noch keine offenen, verteilten Ansätze gibt.

Betrachtet man die Literatur [2.1-2.6], so werden im Bereich der Umweltinformatik zwar Worte wie "verteilt" gelegentlich benutzt. Bezüglich wirklicher verteilter Konzepte, die Umweltdaten an jedem beliebigen Arbeitsplatz einfach verfügbar machen (auch über Grenzen von Bundesländern hinweg), herrscht aber beharrliches Schweigen.

2.1 Übergreifende Konzepte

Die Anforderungen an Benutzerschnittstellen für komplexe Systeme und der zugrundeliegenden Dialogsoftware wurden erstmals in [2.7] und [2.8] angegeben. Es sind dies im einzelnen:

- Unterstützung des Endbenutzers bei der Navigation durch das System,
- Unterstützung des User Interface Designers bzw. Software-Ingenieurs beim Entwurf,
- Integration von Daten und Methoden in eine einheitliche Benutzerschnittstelle, insbesondere auch Integration entfernter Daten,
- eine verteilte Systemarchitektur der Dialogsoftware,
- eine offene Systemarchitektur der Dialogsoftware,
- Abbildung des Softwaresystems an sich in die Benutzerschnittstelle und
- Einbettung von Entwurfsmethoden in das User Interface System.

All diesen Gesichtspunkten muß ein System genügen, welches zum Aufbau von Benutzerschnittstellen für komplexe Systeme geeignet sein soll. Von den genannten Punkten wurden im Rahmen der vorliegenden Arbeit der zweite Punkt nur teilweise und der letzte Punkt nicht betrachtet. Der Schwerpunkt der Arbeiten lag auf der Architektur des Systems.

Das primäre Ziel ist die Unterstützung des Endbenutzers bei der Handhabung der im System vorhandenen Daten und Methoden. Die Systeme haben gemeinsam, daß sie

- mit großen Datenmengen arbeiten, die sich über die Zeit ändern (auch in ihrem Umfang),
- die Daten von unterschiedlichster Art, Qualität und Bedeutung sein können und
- die Zusammenhänge der Daten untereinander von komplexer Natur sind.

Da die Daten und Zugriffsmethoden innerhalb verteilter Rechnersysteme vorliegen, entsteht eine vielfach verkoppelte Sicht des Benutzers auf das System. Abb. 2.1 stellt eine Konkretisierung von Abb. 1.2 des letzten Kapitels dar.

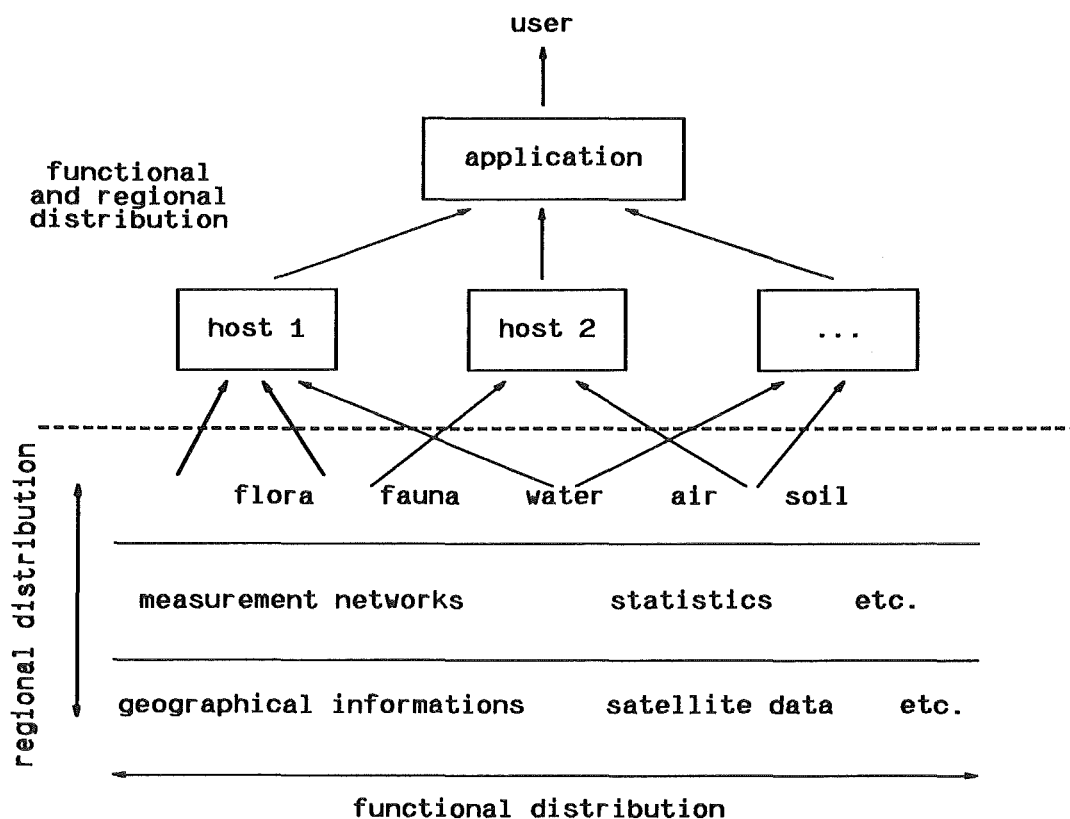


Abb. 2.1: Benutzersicht eines komplexen Systems (aus [2.6])

Die Hauptaufgaben der Benutzerschnittstelle sind dabei

- die Bereitstellung einer komfortablen Navigation durch das System,
- die Unterstützung durch eine intelligente Hilfe,
- Hilfe bei der Handhabung komplexer Problemräume und
- die Verdeutlichung der Beziehungen untereinander, also kausaler Abhängigkeiten, aufgrund des Wissens des Systems.

Der Hauptaugenmerk der Untersuchungen ist dabei die Navigation im System. Es soll an jedem Interaktionspunkt die Möglichkeit gegeben werden, in möglichst optimaler Weise zu demjenigen Interaktionspunkt zu gelangen, der wahrscheinlich als nächster benötigt wird. Es soll also eine Anpassung des Dialogs an spezifische Situationen und Aufgaben zur Laufzeit erfolgen (Abb. 2.2). Zur Lösung einer Aufgabe A_i soll der Dialog bezüglich eines Systemzustands Z_j optimiert werden. Hierzu ist Wissen über den Problemraum und über die Problemlösung durch den Benutzer vonnöten. Diese Vorgehensweise wurde bei technischen Prozessen detailliert untersucht.

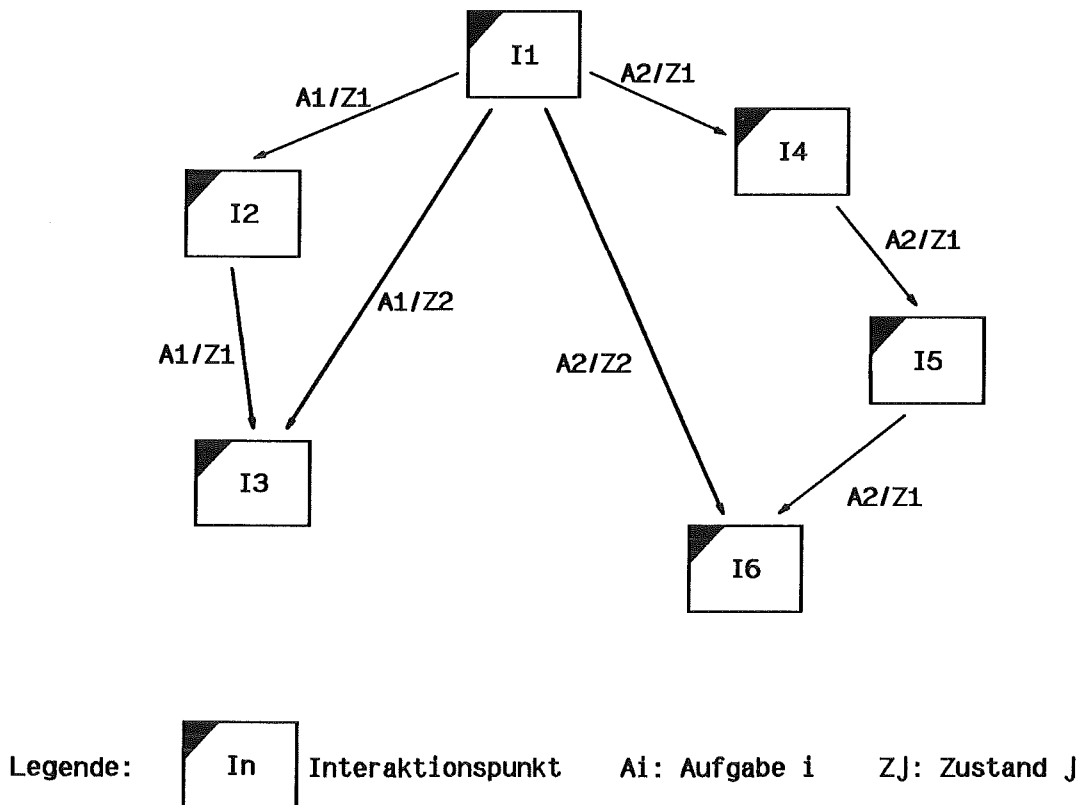


Abb. 2.2: Modifikation der Navigation zur Laufzeit

Gleichzeitig soll aber jederzeit ein Maximum an Information möglichst schnell zur Verfügung stehen, ohne daß jedoch diese Information den Benutzer überschwemmt. Dies führt sinnvollerweise dazu, Informationen direkt bei den Objekten anzusiedeln, zu denen die Informationen gehören und die Information durch Anwahl der Objekte hervorzuheben. Dieses Konzept wurde bei technischen Prozessen sowie bei einem Immisionsmeßnetz betrachtet.

Jede Benutzerschnittstelle besitzt zumindest zwei Gesichter: für den Endbenutzer und für den Dialog-Designer. Die Programmierung von Benutzerschnittstellen nimmt nach wie vor einen großen Raum der gesamten Implementierung eines Softwaresystems ein. In vielen praktischen Fällen ist es allerdings nicht so, daß die Benutzerschnittstelle und der Algorithmus von unterschiedlichen Menschen implementiert werden (Diese Trennung wird oft von Protagonisten des Seeheim-Modells [2.9] als dessen Vorteil hervorgehoben).

Daher wird es immer notwendiger, brauchbare Werkzeuge zur einfachen und effizienten Gestaltung von Benutzeroberflächen zur Verfügung zu stellen. Die Frage der Werkzeuge war für die vorliegende Architektur-orientierte Arbeit von geringerer Bedeutung, auch wenn die derzeit noch laufenden Arbeiten sich dieses Gebiets widmen. Allerdings wurde bei der in Kap. 4 vorgestellten Systemarchitektur auf folgende Randbedingungen geachtet:

- Die Architektur sollte derart aufgebaut sein, daß sie die Handhabung komplexer Problemräume unterstützt.
- Es sollte in einfacher Weise möglich sein, neue (auch entfernte) Daten und u.U. in der Zukunft auch Methoden in eine lokale Benutzerschnittstelle zu integrieren.
- Durch einen objekt-orientierten Ansatz sollte vor allem die Wiederverwendbarkeit einmal implementierter Objekte und Objektklassen gewährleistet werden.

Der besondere Augenmerk wurde dabei auf den zweiten Punkt gelegt. Ein Kernpunkt der Architektur soll ihre Integrationsfähigkeit sein. Integration bedeutet in diesem Zusammenhang die Fähigkeit, Daten und Methoden in ein System einzubetten. Während die Nahtstelle zwischen Anwendung und Dialogsoftware - folgt man der Literatur - in der Regel durch das User Interface Management System (UIMS) gestaltet wird (welches selbst wieder ein geschlossenes System sein kann), wurde hier ein Objekt- und Kommunikationsmodell entwickelt, welches beliebige Daten direkt in User-Interface-Objekte integriert. Dabei können die Daten über beliebige Anwendungen quer über ein Rechnernetzwerk verstreut sein und sie dürfen sich spontan ändern.

Diese Anforderung geht direkt aus den betrachteten komplexen Systemen hervor. Würde man den verteilten Aspekt der Systeme nicht berücksichtigen, so wäre ein System zur Dialoggestaltung für diese Systeme schwerlich einsetzbar. Insbesondere wird dies bei Störfällen in technischen Anlagen oder bei Grenzwertüberschreitungen

in Meßnetzen klar. Kann die Dialogsoftware in diesen Fällen nicht direkt und spontan auf nicht vorhersehbare Ereignisse reagieren, so wird sie ihrer Aufgabe nicht gerecht.

Der Entwurf einer Benutzerschnittstelle wird daher zum *Entwurf in einem verteilten System*. Unter einem solchen werden in der Folge eine Menge von Datenverarbeitungsanlagen verstanden, die über Kommunikationsnetze lose miteinander gekoppelt sind und unterschiedliche Hardware, Betriebssysteme und Softwarekomponenten betreiben.

Zum Betrieb eines verteilten DV-Systems für die betrachteten Systemklassen sind natürlich unterschiedliche Ansätze denkbar, so z.B. für den Umweltbereich ein Ansatz einer zentralen oder verteilten Datenbank, die die Schnittstelle zu den einzelnen Anwendungen bildet. Allerdings ist ein solcher Ansatz nicht unproblematisch, wenn es z.B. um die Einhaltung von zeitlichen Randbedingungen geht. Außerdem prägt er dem Gesamtsystem eine Philosophie auf, die für bestimmte Teile gut, für andere Teile weniger gut geeignet sein mag.

Die Konzeption eines *offenen Systems* hingegen besitzt diesen möglichen Nachteil nicht. In einem offenen System sind alle Teilsysteme autonom, in der Wahl ihrer Mittel frei und verständigen sich einzig und alleine durch Kommunikation. Es werden keinerlei Konventionen über Datenformate oder ähnliches in einzelnen Teilsystemen vereinbart, denn es mag gute Gründe dafür geben, warum in einem Teilsystem die gleichen Daten anders gespeichert werden als in einem anderen. Die einzige global existierende Konvention ist jene, die die Art und Weise des Austauschs von Daten regelt.

Offene Systeme haben den Vorteil, daß jede einzelne Applikation das Werkzeug, die Programmiersprache, den Rechner u.s.w. benutzen kann, die für die spezifischen Aufgaben des Teilsystems bestens geeignet sind. Es existieren keine konzeptionellen Beschränkungen bezüglich des Aufbaus eines Teilsystems, wie es z.B. schon gegeben ist, wenn man einen Datenbankansatz wählt.

Dialogwerkzeuge, die in komplexen Systemen eingesetzt werden, sollten demgemäß sinnvollerweise ebenso auf einem offenen Konzept beruhen. Beliebige Teilsysteme im Netz können unabhängig voneinander Daten liefern, die im Dialogsystem dargestellt werden und mit denen der Benutzer interagiert. *Der Austausch der Information geschieht durch Kommunikation innerhalb eines verteilten, offenen Konzepts.*

Die Komplexität des Gesamtsystems hat ihrerseits Rückwirkungen auf die Gestaltung der Benutzerschnittstelle. Je heterogener, verteilter und offener die Umgebung wird, umso mehr wird es notwendig sein, *das Rechnersystem selbst zu visualisieren*. Das heißt, das Rechnersystem ist auf die Benutzerschnittstelle mit abzubilden.

Die abschließende Anforderung liegt darin begründet, daß in der Zukunft verstärkt neue Entwurfsmethoden für Benutzerschnittstellen in der Praxis zu erwarten sind. Es hat sich inzwischen die Einsicht durchgesetzt, daß man intensiver den Endbenutzer in den Entwurfsprozeß einbeziehen muß, um Fehler beim Softwarede-

sign zu vermeiden. Die Dialogsoftware sollte also einen iterativen Entwurfsprozeß unterstützen, wie er z.B. beim *user centered design* angestrebt wird [2.10]. Ein objekt-orientierter Ansatz der Dialogsoftware bietet hier gewiß Vorteile. Dies wurde allerdings nicht untersucht.

Aus der Menge der Anforderungen wurden schwerpunktmäßig diejenigen untersucht, bei denen ein erhöhter Forschungs- und Entwicklungsbedarf erschien. Diese lassen sich grundsätzlich in folgende zwei Gruppen gliedern:

- **Gruppe 1: Methodik der Darstellung**
 - Navigation in komplexen Systemen
 - Filterung komplexer Informationsmengen und
 - Visualisierung des komplexen DV-Systems selbst

- **Gruppe 2: Methodik der Implementierung**
 - Integration
 - Verteiltheit und
 - Offenheit

Diese beiden Gruppen bilden den Ausgangspunkt für die Entwicklung einer Architektur zur Visualisierung in komplexen Systemen.

In den beiden folgenden Abschnitten soll nun noch auf spezifische Gesichtspunkte bei technischen Systemen und bei Umweltinformationssystemen eingegangen werden, soweit sie in besonderer Weise für die oben stehenden Gesichtspunkte von Bedeutung sind.

2.2 Spezifische Gesichtspunkte bei der Führung technischer Anlagen

In diesem Abschnitt werden die spezifischen Anforderungen an den Dialog und an die Dialogsoftware bei technischen Anlagen besprochen. Die im vorangegangenen Abschnitt beschriebenen Anforderungen an die Architektur werden begründet und - soweit hier weitergehende Anforderungen auftreten - erweitert.

Die Schwierigkeiten, einen technischen Prozeß sinnvoll zu visualisieren, sind außerordentlich vielschichtig. Die Aufgabe der Operateure ist es, einen komplexen technischen Prozeß in einer optimalen Art und Weise zu steuern. Dabei sind

- die Mengen an relevanten Daten groß
- die kausalen und temporalen Abhängigkeiten komplex und
- die zeitliche Informationsdichte stark schwankend und (besonders in Störfällen) oft hoch.

Es findet also ein Entscheidungsprozeß auf einer komplexen Informationsmenge statt. Die derzeit verfügbaren Leitsysteme haben vor allem den Nachteil, daß keine Filterung der präsentierten Information möglich ist und daß sie sich insgesamt zu passiv verhalten: es ist keine irgendwie geartete Führung des Operateurs möglich. Es fehlt an einer geeigneten Adaption des Dialogs an den Prozeßzustand. Diese Tatsache wirkt sich im stationären Betrieb einer Anlage nicht negativ aus, wohl aber bei Störfällen.

In solchen Störfällen kann es dazu kommen, daß eine große Menge von Alarmen in kurzer Zeit entstehen, nämlich der Alarm der primären Ursache (Primärstörung) sowie die Alarme sämtlicher Folgefehler (Sekundärstörungen). Es ist dann die Aufgabe des Operateurs, alle diese Störungen zur Kenntnis zu nehmen, die primäre Ursache(n) zu erkennen und entsprechende Maßnahmen zu ergreifen.

Wenn die Anzahl Alarme pro Zeiteinheit zu groß wird, kann dies zu einer mentalen Überlastung führen. Sachs bezeichnete diesen Zustand als *cognitive overload* [2.11]. Abb. 2.3 zeigt ein Beispiel einer solchen Kette von Alarmen. Gelingt es nicht schnell genug, den Prozeß wieder an seinen gewünschten Betriebszustand heranzuführen, so kann es sein, daß die Anzahl von Teilprozessen, die fehlerhaft arbeiten, nach und nach ansteigt, bis schließlich eine Abschaltung der Anlage aus Sicherheitsgründen erfolgt.

Man kann ohne Übertreibung sagen, daß es mit der heute eingesetzten Technologie weder möglich ist, die Informationsauswahl (sprich: Konzentration auf die primären Ursachen aufgrund von Hintergrundwissen über den Prozeß) noch die Entscheidungsfindung (sprich: was ist zu tun?) zu unterstützen.

Ein dem *cognitive overload* ähnlicher Fall kann entstehen, wenn eine Störung auftritt, welche äußerst selten vorkommt. Dem Anlagenfahrer ist u.U. in diesem Moment die Konsequenz des Alarms und seiner Entscheidung nicht so bewußt wie bei einer häufiger auftretenden Störung.

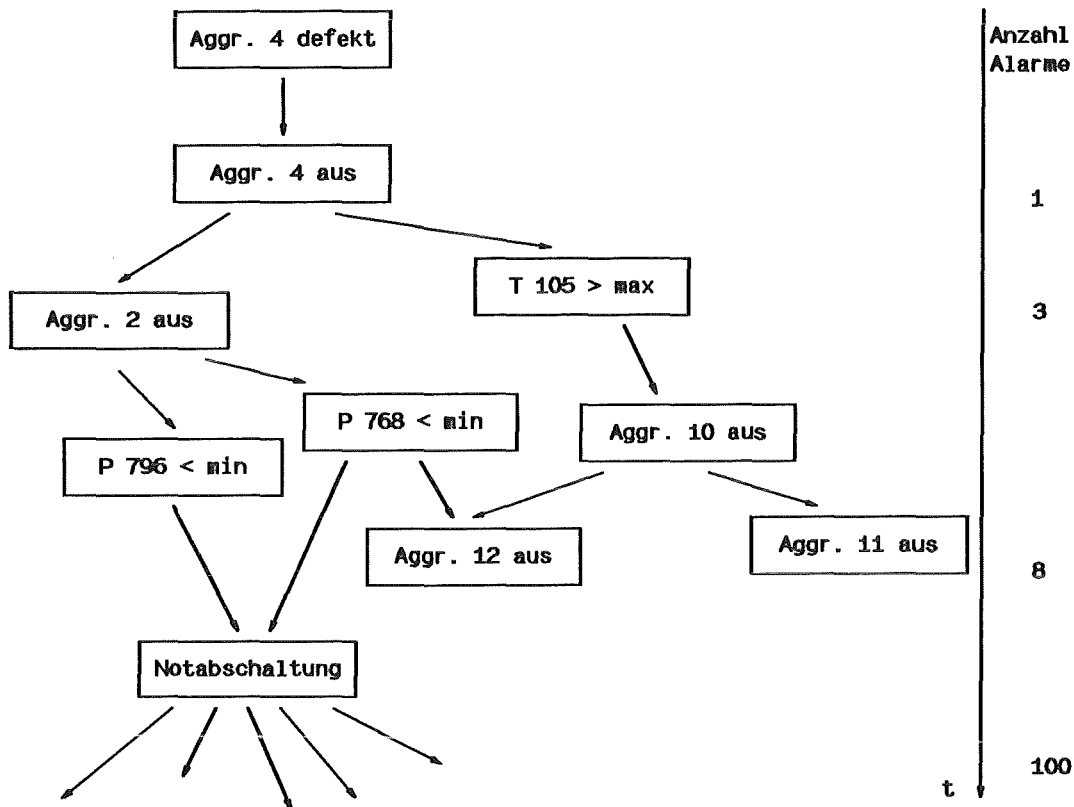


Abb. 2.3: Cognitive Overload

Ebenfalls kritisch sind Verriegelungen in der Automatisierungsstruktur, also Bedingungen der Form:

- Wenn die Temperatur $T_{308} > 500\text{ °C}$, dann Aggregat Pumpe-5 aus bzw.
- Pumpe-5 darf nur eingeschaltet werden, wenn $T_{308} < 500\text{ °C}$.

Solche Verriegelungen sind bei technischen Anlagen allgegenwärtig. Muß zur Behebung einer anderen Störung das Aggregat eingeschaltet werden, so führt eine solche Verriegelung zur Abweisung des Befehls. Um also eine Störung zu beheben, muß zunächst eine andere Störung behoben werden, nämlich die von Pumpe-5. Jene hängt aber von der Temperatur T_{308} ab.

Das Beispiel zeigt, daß oft Störursachen bis zu deren primären Ursachen zurückgeführt werden müssen, was seinerseits kostbare Zeit benötigt. Hier kann ein Visualisierungssystem dazu beitragen, durch entsprechende Filterung und dadurch entsprechende Konzentration auf das Wesentliche den Prozeß der Störungsbehebung zu unterstützen. Wie schon erwähnt sind heute gängige Leitsysteme hierzu nicht zu gebrauchen.

Da gravierende Störungen im Betrieb immer wirtschaftliche und oft auch ökologische Folgen haben, ist es nötig, Mechanismen zur Informationsfilterung und Bedienerführung in die Mensch-Maschine-Schnittstelle zu integrieren. Dies kann nur unter Zuhilfenahme des technologischen und heuristischen Wissens über den Prozeß - insbesondere die steuerungstechnischen Zusammenhänge - geschehen.

Aus der Notwendigkeit heraus, technische Anlagen besser zu betreiben, werden seit einigen Jahren Systeme zur Entscheidungsunterstützung für Operateure entwickelt. Auf einige wenige Beispiele wird in Kap. 3 eingegangen werden.

Die Gestaltung der Benutzeroberfläche, die für den Umgang des Operateurs mit dem Prozeß von zentraler Bedeutung ist, wurde bei vielen Arbeiten weitestgehend ausgelassen und es ist nur ein relevanten Projekt zu diesem Thema bekannt (GRADIENT, [2.12]). Dies mag daran liegen, daß hier eine Schnittstellenproblematik vorliegt: für KI-Forscher mögen die eingesetzten Wissensbasen zu dürftig sein, für Computergraphiker mögen die Anforderungen an die Graphik zu schmal sein und die Arbeiten auf dem Gebiet der User Interface Management Systeme (UIMS) scheinen traditionell eher im nicht-technischen Bereich angesiedelt zu sein.

Eine 1989 durchgeführte umfangreiche Recherche führte zu der Schlußfolgerung, daß auf diesem Gebiet der Prozeßführung bislang wenig methodische Ansätze sichtbar sind. Gleichzeitig zeigten die Arbeiten an dem TAMARA-Projekt [2.13], daß genau dieses Gebiet einer näheren Untersuchung würdig erscheint. Es wurde das Ziel abgesteckt, Erfahrung und Wissen der Operateure mit den informationsverarbeitenden Kapazitäten der Maschine zu einem gemeinsamen Schlußfolgerungsprozeß zu verbinden. Dabei sollten folgende zwei Kernfragen beantwortet werden:

- Wie wählt man die Information und deren Darstellung unter Berücksichtigung des Zustands des Prozesses aus und wie kann man den Operateur führen und bei der Entscheidungsfindung unterstützen?
- Wie muß ein graphisches System konzipiert werden, welches dies ermöglicht?

Um an die Beantwortung dieser Fragen zu gehen, muß man zunächst die vorhandenen Informationsmengen und Informationsquellen näher betrachten. Tab. 2.1 gibt hierüber einen Überblick.

Von den genannten Informationen werden derzeit lediglich die online anfallenden Meßwerte zur Prozeßvisualisierung verwendet. Insbesondere das Fehlen der technischen Dokumentation in der Benutzerschnittstelle ist äußerst nachteilhaft, da bei der Fehlersuche ein ständiger Wechsel des Mediums (Bildschirm - Papier) und eine Suche in dem Medium Papier auftritt. Die erste triviale Forderung ist daher die Einbettung der technischen Dokumente in den Dialog.

Information	Quelle	System bzw. Medium
Meßwerte, Zustände	Meßeinrichtungen	Automatisierungssystem
technische Information über einzelne Aggregate	Dokumentation	Papier
steuerungstechnische Zusammenhänge	Schaltungen, Programme	Papier
physikalisch-chemische Zusammenhänge	Forschung, Erfahrung während des Betriebs	Diagnosesysteme u.ä.
heuristisches Wissen	Erfahrung der Bediener	Expertensysteme u.ä.

Tab. 2.1: Informationen über technische Prozesse

Steuerungstechnische Informationen können dazu verwendet werden, den Dialog an den Prozeßzustand zu adaptieren. Dies war ein Kernpunkt bei dem Entwurf von Dialogen, wie er in Kap. 5 aufgezeigt wird.

Die Informationen aus den letzten beiden Punkten werden in der Zukunft verstärkt durch Systeme wie z.B. Expertensysteme aufgeschlossen werden. Neben dem Visualisierungssystem werden also Systeme zur Diagnose, Prognose, Qualitätssicherung, Statistik, numerische Beobachter etc. eingesetzt werden. Hier entstehen für die Benutzeroberfläche potentielle Probleme:

- Es besteht die Gefahr, daß die verwendeten Systeme separat und unkoordiniert entwickelt werden und dadurch eine uneinheitliche Benutzerschnittstelle entsteht, die darüberhinaus noch über mehrere Ein-/Ausgabegeräte verteilt ist (Abb. 2.4).
- Die Informationen der unterschiedlichen Systeme sind von unterschiedlicher Bedeutung: ein diagnostizierter Wert hat eine andere Qualität als ein realer Meßwert.
- Die ohnehin große Menge an Information wächst erheblich an.

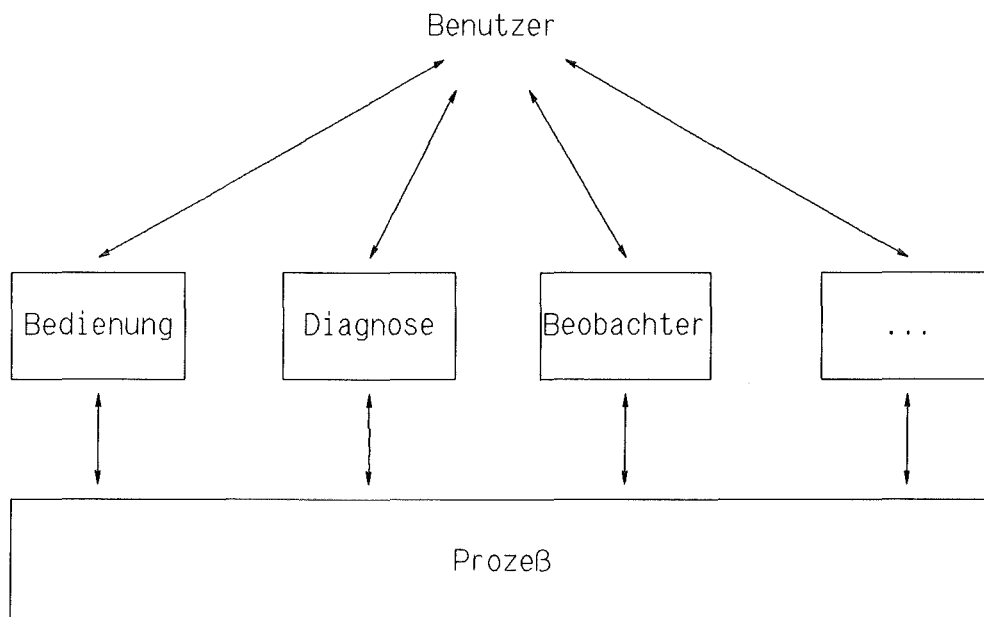


Abb. 2.4: Uneinheitliche Benutzerschnittstelle

Diesen Schwierigkeiten soll dadurch begegnet werden, daß eine uniforme Benutzerschnittstelle verwendet wird, die allerdings über einen entsprechenden Auswahl- und Führungsmechanismus verfügt. In diese Benutzerschnittstelle sind die Informationen aus den einzelnen Teilsystemen zu integrieren (Abb. 2.5).

Aufgrund der funktionalen Verteilung der Teilsysteme und aufgrund der räumlichen Verteilung von Automatisierungssystemen ist klar, daß dem Visualisierungssystem eine verteilte Systemarchitektur zugrunde liegen muß. Da die einzusetzenden Methoden von unterschiedlicher Art sind, so z.B. PROLOG-Systeme für Diagnosezwecke, C-Programme für numerische Beobachter, muß die Systemarchitektur zugleich offen gestaltet werden.

Offenheit wird in diesem Zusammenhang folgendermaßen angesehen: jedes beliebige Teilsystem ist in einer beliebigen Umgebung implementierbar, die für das jeweilige Teilproblem optimal geeignet ist. Die Schnittstelle der Systeme untereinander ist weder eine Datenbankschnittstelle noch eine Dateischnittstelle sondern eine *Netzwerkschnittstelle*, die auf einer standardisierten Task-zu-Task-Kommunikation beruht. Einzig eine solche Netzwerkschnittstelle ist beim derzeitigen Stand der Technik geeignet, ein offenes System zu gewährleisten.

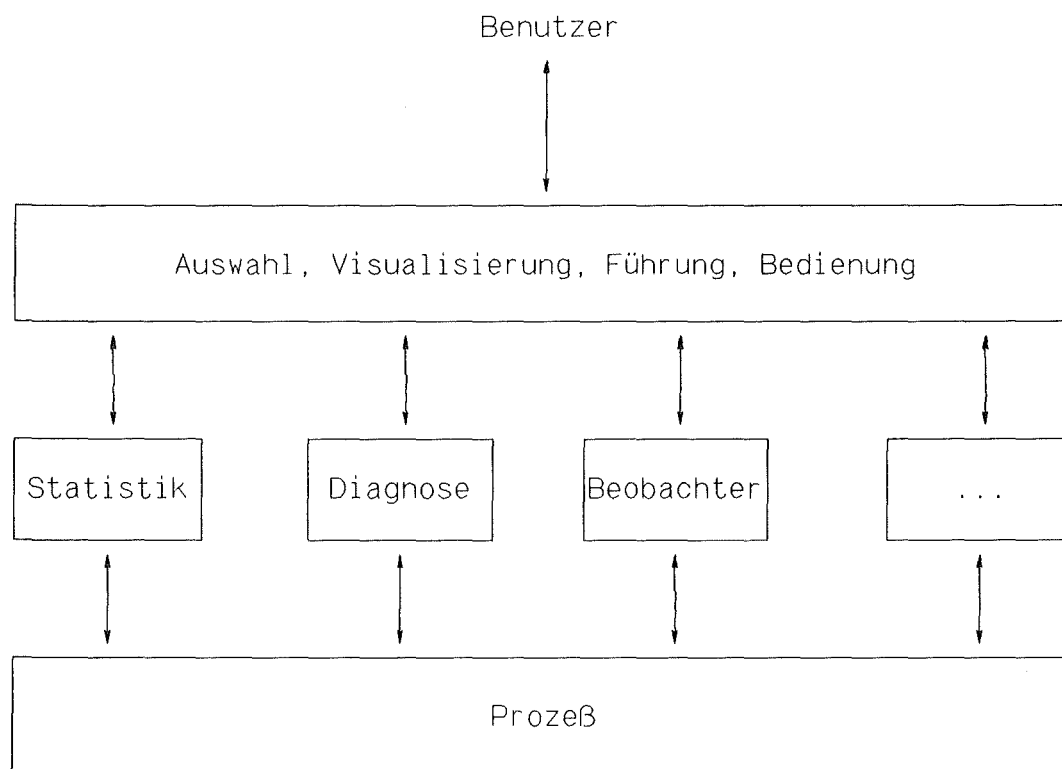


Abb. 2.5: Einheitliche Benutzerschnittstelle

Offenheit hat aber auch noch eine andere Bedeutung: durch die prinzipielle Autonomie der einzelnen Teilsysteme ist von vorne herein gewährleistet, daß dem Gesamtsystem keine Struktur eingeprägt wird. Das offene Systemkonzept erzwingt keine organisatorische Struktur. Damit ist es möglich, sowohl vollkommen dezentrale Systeme aufzubauen, d.h. jeder kommuniziert mit jedem, als auch hierarchisch gegliederte Systeme zu realisieren, wie sie in der Automatisierungstechnik gängig sind. Das offene Konzept bietet ein Höchstmaß an konstruktiver Freiheit beim Aufbau einer konkreten Anwendung.

Eine bei Automatisierungssystemen erhebliche Randbedingung ist deren Effizienz bzw. Echtzeitfähigkeit, d.h. die Einhaltung zeitlicher Randbedingungen. Nun ist es i.d.R. nicht nötig, an ein Prozeßvisualisierungssystem scharfe zeitliche Randbedingungen zu stellen, da der Prozeß der Fehlersuche und Störungsbehebung ohnehin erheblich länger andauert als der Prozeß zur Übertragung und Darstellung von Information. Dennoch soll hier eine kurze Betrachtung der Effizienz des Informationsaustauschs vorgenommen werden.

Heutige Automatisierungssysteme sind stark an regelmäßigen zeitlichen Zyklen orientiert. Insbesondere analoge Meßwerte werden zyklisch erfaßt, übertragen, vi-

sualisiert und archiviert. Dies hat auch seine Berechtigung, da man bei der zyklischen Bearbeitung klare zeitliche Grenzen einhalten kann und damit Echtzeitfähigkeit garantiert. Die Summe der Zeitdauern der Algorithmen pro Zeiteinheit (z.B. Ein-/Ausgabetreiber, Regler u.Ä.), in den entsprechenden Zyklen verteilt, gibt einen genauen Aufschluß über das Echtzeitverhalten des Systems.

Auch die Abarbeitung von Regelungsalgorithmen ist an festen Zyklen orientiert, da in die Berechnung eines digitalen Reglers aus dem kontinuierlichen Regelungsverhalten heraus die Abtastzeit eingeht. (Ein Parameter der z-Transformation ist die Abtastzeit [2.14]). Ein auf diese Weise optimierter Regler muß auch in dem für ihn berechneten Zyklus ablaufen, sonst kann sein Regelungsverhalten vom berechneten Verhalten abweichen, er kann sogar instabil werden.

Für Zwecke der Prozeßvisualisierung (und auch für die Archivierung) ist der Zyklus hingegen vollkommen überflüssig. Oft werden Meßwerte zum Leitsystem hin übertragen, dort visualisiert und archiviert, obwohl diese sich geringfügig wenn nicht sogar gar nicht geändert haben. Jedenfalls liegt die Änderung oft unterhalb der Meßgenauigkeit der Betriebsmeßgeräte. Will man z.B. KI-Werkzeuge zur Diagnose einsetzen, so muß man ohnehin die zyklische Denkweise überwinden, da diese Werkzeuge i.d.R. mit den Zeitanforderungen von Automatisierungssystemen nicht Schritt halten können.

Diesem Problem wird z.B. in der KI dadurch begegnet, daß man eine Filterung des Informationsstroms am Eingang des Teilsystems vornimmt. Diese Vorgehensweise wird z.B. in [2.15] und [2.16] beschrieben. Diese Vorgehensweise kann schwerlich als intelligent bezeichnet werden, da hierbei im zumeist eingenommenen stationären Betriebspunkt der Anlage Unmengen von Daten übertragen werden, die nirgends real Verwendung finden können, da sie sich gegenüber der letzten Übertragung kaum geändert haben. Sie werden schlichtweg am Eingang des Teilsystems weg gefiltert.

Die vorangegangenen Bemerkungen führen zu folgender Konsequenz: es ist eine Art von *Informationsmanagement* nötig, welches schon am Ausgang des Teilsystems, das ein Datum generiert, eine potentielle Information einer Filterung unterwirft. Diese Aussage hat die XAVIA-Architektur, insbesondere das Kommunikationsmodell, erheblich geprägt.

Ein wichtiges Ziel des Informationsmanagements im verteilten System ist daher, *nur signifikante Änderungen von Prozeßvariablen zwischen Teilsystemen und Visualisierungssystem auszutauschen*. Jeder Teilnehmer im verteilten System kann für eine bestimmte Information eines anderen Teilnehmers ein Signifikanzkriterium angeben, welches bestimmt, bei welcher Änderung des Wertes der Wert zu übertragen ist. Auf diese Weise kann z.B. ein Visualisierungssystem einen Analogwert bei 5-prozentiger Änderung anfordern, ein System zur Alarmfilterung fordert hingegen den gleichen Wert nur bei Überschreitung eines Grenzwertes an.

Um eine gewisse Rangfolge von Informationen möglich zu machen, wird (für technische Prozesse) außerdem noch gefordert, daß Informationen mit einer Priorität

behaftet sind, welche die Reihenfolge beim Versenden, Empfangen und Bearbeiten von Nachrichten bestimmt.

Faßt man die Gedanken dieses Abschnitts zusammen, so kommt man zu einer verteilten, offenen Systemarchitektur, deren Kern ein möglichst effizientes und in gewisser Weise "intelligentes" Informationsmanagement ist. Die Managementerschicht verbindet einzelne Teilsysteme untereinander und alle Teilsysteme mit einer einheitlichen Benutzerschnittstelle (Abb. 2.6).

Die Benutzerschnittstelle bündelt die Informationen aus den einzelnen Teilsystemen in eine uniforme Umgebung ein, sie muß allerdings auch Unterschiede in der Bedeutung der Daten hervorheben. Sie soll über eine gewisse "Intelligenz" verfügen: aufgrund von Prozeßwissen wird zur Laufzeit ein Auswahlmechanismus durchgeführt, der aus den großen, komplexen Datenmengen die wichtigsten Informationen bestimmt und die Aufmerksamkeit des Operateurs auf diese lenkt.

Geht man auf diese Weise vor, so kann man die Mensch-Maschine-Schnittstelle für alle genannten Teilsysteme in einem Ein-/Ausgabegerät vereinigen. Dies ist auch

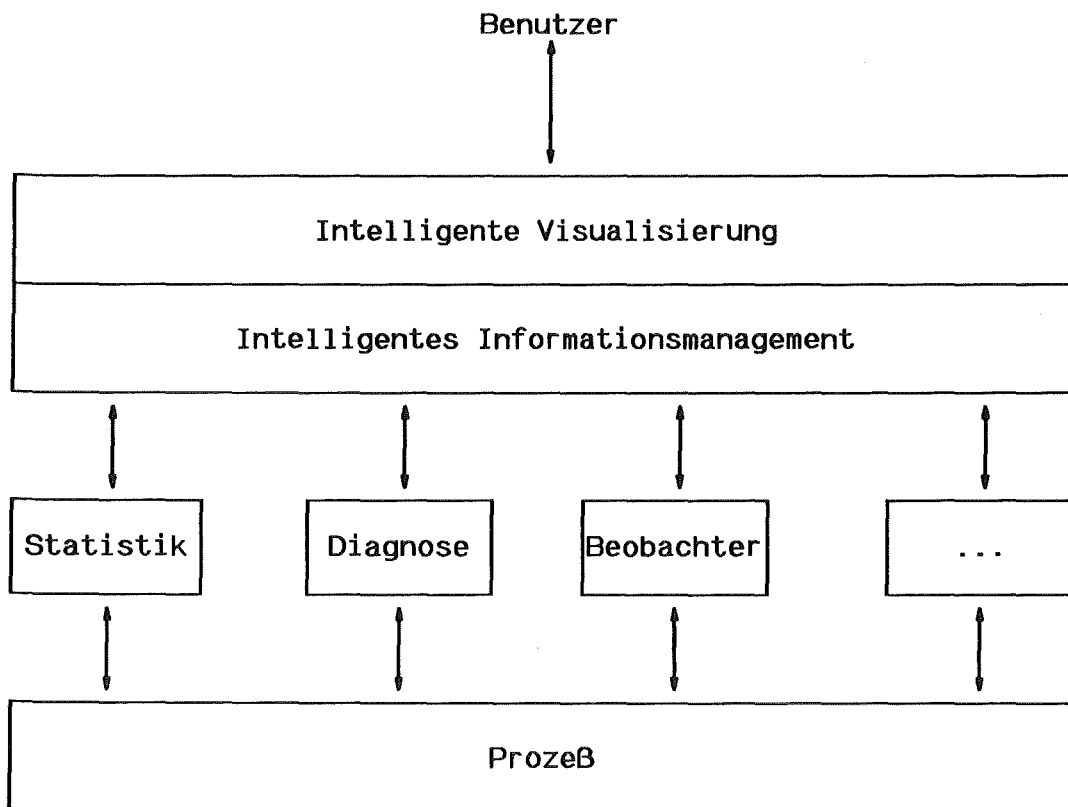


Abb. 2.6: Intelligentes Prozeßvisualisierungssystem

zwingend geboten, denn es ist nicht akzeptabel, daß sich Operateure in der (im Störfall) von Streß und Hektik geprägten Atmosphäre einer Leitwarte noch mit einer Reihe uneinheitlicher Dialogsysteme auseinandersetzen müssen.

Als Fazit dieses Abschnitts fassen folgende drei Aussagen das Gesagte möglichst knapp zusammen:

- Das Systemkonzept muß zwingend verteilt und offen sein.
- Es muß ein Konzept zum Informationsmanagement existieren.
- Alle Informationen über den Prozeß müssen in einer einheitlichen Mensch-Maschine-Schnittstelle zusammen fließen.

Vernachlässigt man einen dieser drei Punkte, so wird man heutigen Anforderungen und zukünftigen Entwicklungen im Bereich der Prozeßvisualisierung mit Sicherheit nicht gerecht werden.

2.3 Spezifische Gesichtspunkte bei Umweltinformationssystemen

Der Schutz der natürlichen Umwelt ist in der Industriegesellschaft nur noch mit validen Informationen über den Ist-Zustand der Umwelt und über mögliche Entwicklungen in der Zukunft möglich. Die zu diesem Zweck erhobenen, gemessenen, gespeicherten und ausgewerteten Daten wachsen von Tag zu Tag. DV-Systeme, die sich mit der Erhebung, Speicherung und Bewertung dieser umweltrelevanten Daten befassen und die versuchen, dies von einem übergeordneten Standpunkt aus zu tun, werden i.d.R. Umweltinformationssysteme (in der Folge kurz UIS) genannt.

UIS sind an der Praxis orientierte Systeme. Da die Hoheiten im Vollzug über unterschiedliche Verwaltungsebenen und in unterschiedlichen Regionen liegen und diese Systeme sich jeweils an lokalen Gegebenheiten orientieren, ist eine genaue Definition der Inhalte eines UIS schwierig.

Die Aufgaben und Inhalte von UIS werden z.B. in [2.17], [2.18], [2.19] und [2.20] beschrieben. Die folgenden Ausführungen lehnen sich an die von Schimak durchgeführte Studie [2.17] und die Überlegungen bezüglich offener Systeme im Umweltschutz in [2.18] an.

Die Aufgaben eines UIS liegen unter anderem in der Dokumentation, Präsentation und Visualisierung von Umweltdaten, in der Informationsverarbeitung im Verwaltungsvollzug, in der kontinuierlichen Erfassung und Archivierung von Daten, der Planung, Modellbildung und Simulation und der Prognose. Dies mag ein kleiner und nicht vollständiger Ausschnitt aus allen möglichen Aufgaben eines UIS sein.

Dabei haben die unterschiedlichen Benutzergruppen (Behörden, Forschung, Öffentlichkeit,...) stark variierende Anforderungen an die Verfügbarkeit und Auswertung der gemessenen und erhobenen Informationen.

Die Inhalte von UIS werden heute von den Inhalten von sog. Fachinformationssystemen bestimmt, deren Grundlagen die eingebrachten Daten und Methoden sind. Unter Fachinformationssystemen versteht man Informationssysteme für einen thematischen Umweltbereich, also z.B. Wasser, Boden, Luft oder Artenschutz. Die einzelnen Fachinformationssysteme besitzen ähnliche Verkopplungen wie die natürlichen Systeme.

In Abb. 2.7 (entnommen aus [2.18] bzw. [2.17]) wird der denkbare Aufbau eines UIS und eine Gliederung in Fachinformationssysteme gezeigt. Die Fachinformationssysteme werden unter anderem folgende Daten beinhalten:

- *Topographische Basisdaten*

Hierunter fallen

- digitale topographische Karten
- digitale Geländemodelle
- digitalisierte Luftbilder
- Satellitendaten

Diese Daten befinden sich in einem Topographisch-Kartographischen Informationssystem (TOKAPIS).

- *Daten zur Raumstruktur*

Hierunter fallen die Daten, die in Abteilungen für Raumordnung und Statistik produziert und in ein Raumordnungsinformationssystem (RIS) vorort eingebracht werden. Dies sind im wesentlichen

- statistische Daten für Länder, Bezirke und Gemeinden über Bevölkerung, Wirtschaft, Siedlungsstruktur und Flächennutzung
- Digitalkartierung der Siedlungs- und Infrastruktur, Flächennutzung, Verwaltungsgrenzen, Raumeinheiten der Raumordnung und des Umweltschutzes

- *Daten für Naturschutz, Artenschutz und Landschaftsschutz*

Hierzu gehören:

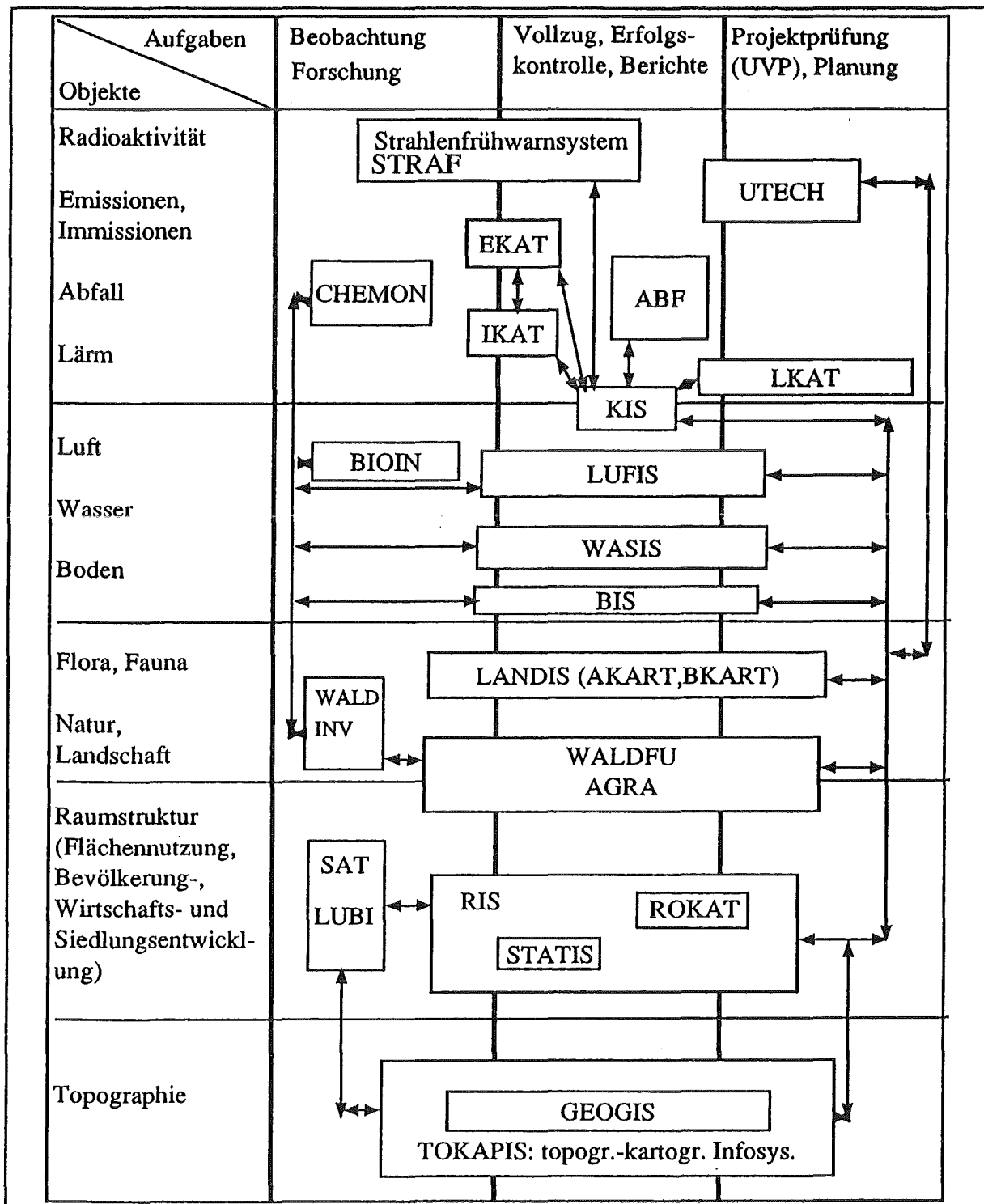


Abb. 2.7: Komponenten eines UIS

- digitale Biotopkartierung und digitale Artenschutzkartierung
- Digitalkartierung der Land- und Forstwirtschaft
- digitale Waldschadenskartierung
- rasterbezogene Sammlung ökologischer Daten

Diese Daten finden sich in Fachinformationssystemen wie z.B. in einem Agrarinformationssystem (AGRA) und in einem Landschaftsinformationssystem (LANDIS) usw.

- *Daten zu Umweltbelastung und Umweltgefahren*

Es handelt sich hierbei um Daten, die aus Fachinformationssystemen wie aus Monitoring-Systemen stammen können wie z.B. aus

- Meßnetzen zur Überwachung der Luftbelastung (LUFIS), Bodenbelastung (BIS) und der Belastung von Grund- und Oberflächengewässern (WASIS)
- Frühwarnsystemen für Lawinen, Hochwasser und Smog
- Datenbanken umweltrelevanter Anlagen (UTECH)
- Bioindikatormeßnetzen (BIOIN)
- Systemen zur Unterstützung des Verwaltungsvollzugs in den Bereichen Abfall und Sondermüll (ABF), Lärmschutz (LKAT) und Strahlenschutz
- Strahlenfrühwarnsystemen (STRAF)

Diese Aufzählung kann mit Sicherheit nicht als vollständig betrachtet werden, zeigt aber schon, daß wir uns mit einer großen, komplexen, verteilten Informationsmenge befassen müssen.

In jedem der Fachinformationssysteme werden Instrumente DV-technischer Art eingesetzt, die die tägliche Arbeit im Umgang mit den Informationen erleichtern sollen. Diese Instrumente basieren auf informationstechnischen Grundlagen wie

- Datenbanken, verteilte Datenbanken [2.21]
- interaktive graphische Instrumente [2.22,2.23]
- User-Interface-Systeme [2.7]
- Methoden der KI, Expertensysteme [2.24]

- Geographische Informationssysteme [2.25]
- Monitoring [2.26]
- Modellbildung und Simulation [2.27, 2.28]
- Bildverarbeitung und Fernerkundung [2.29]

Die in die Fachinformationssysteme eingebrachten Daten und Methoden bilden also die Grundbausteine eines UIS. Das UIS setzt sich aus Fachinformationssystemen zusammen.

Der derzeitige Stand der Integration einzelner Fachsysteme zu übergeordneten Systemen muß aufgrund der Literatur und der Diskussionen als dürftig bezeichnet werden. Insbesondere scheint die Integration an Landesgrenzen oftmals halt zu machen. Dabei ist bezeichnend, daß in der Literatur zwar von Zeit zu Zeit Begriffe wie "verteilt System" und "offenes System" verwendet werden, jedoch werden nirgends Konzepte aus den Gebieten "verteilte Systeme" und "offene Systeme" angewandt oder dargestellt. Die derzeitigen Systeme erscheinen daher als Insellösungen, und zwar

- als *örtliche Insellösungen* (Kommunen, Land, Bund,...)
- und als *thematische Insellösungen* (Wasser, Abfall, Boden,...).

Dieser Zustand wird eigentlich allseits bemängelt, oft hinter vorgehaltener Hand, aber letztlich selten offen ausgesprochen.

Im Bereich der Grundlagenforschung arbeiten Spezialisten i.d.R. innerhalb der Grenzen ihrer eigenen Gebiete. Auch hier findet wenig thematische Integration statt. Gesamtkonzepte sind nur in einzelnen Fällen sinnvoll umgesetzt wie z.B. in [2.28].

UIS und damit die Anforderungen an die Visualisierung innerhalb eines komplexen UIS können daher derzeit noch nicht so definitiv angegeben werden wie dies im vorangegangenen Abschnitt bei technischen Prozessen der Fall war. Sie sind erst auf dem Wege hin zu verteilten und vernetzten Systemen. Dies dürfte einer der wichtigsten Aspekte der Arbeit in den nächsten Jahren auf diesem Gebiet sein.

Selbst einzelne Fachinformationssysteme können schon eine *hohe inhaltliche Komplexität* aufweisen. Anhand eines Bodeninformationssystems soll dies kurz verdeutlicht werden.

Vorsorgemaßnahmen zum Schutz der Böden beruhen auf der Vorhersage von Risiken, die durch bestimmte Nutzungen und Belastungen der Böden entstehen. So werden z.B. auf den Gebieten Schadstoffanreicherung und -verlagerung, Bodenversauerung, Nitratauswaschung, Erosion, Humusschwund, u.s.w. Risikobeurteilungen und Vorhersagen benötigt. Hierzu sind Informationen über die jeweiligen Bodenei-

genschaften, bereits vorhandene Schäden und das Ausmaß der derzeitigen Belastung zu erheben und verarbeiten.

Für eine dem Zweck des Bodenschutzes geeignete Datenbasis werden Bodeninformationssysteme benötigt, die unter anderem folgende Informationen beinhalten

- Informationen über Aufbau und Zustand der Böden, ihre Standort- und Umwelteigenschaften und ihre Belastbarkeit
- geologische und hydrologische Daten
- Informationen über Bodenbelastungen wie Stoffeinträge
- Daten zu Naturschutz und Landschaftspflege

Die einzelnen Informationen können dabei durchaus aus unterschiedlichen Behörden und Geschäftsbereichen kommen oder auch in anderen Fachinformationssystemen gespeichert sein.

Die Architektur eines bundesweiten UIS wird demgemäß sinnvoller Weise nur mit einem *verteilten Ansatz* realisierbar sein. Ein einzelner Arbeitsplatz muß Zugang zu beliebigen Daten und Methoden erhalten, ohne daß dies mit einem größeren Aufwand verbunden ist. Die *Integration bestehender DV-Anwendungen* in ein neues verteiltes, offenes Konzept wird dabei ein Hauptproblem sein, da die einzelnen Umgebungen von heterogener Natur sind. Damit wird die UIS-Systemarchitektur wie in Abb. 2.8 gezeigt von mehreren Einflußgrößen bestimmt, deren wichtigsten aus technischer Sicht die der Kommunikationsfähigkeit und Offenheit sind.

Einzelne Entwicklungen sind aufgrund der unterschiedlichen Vollzugshoheiten kaum zu beeinflussen, d.h. es kann kein global definiertes Konzept für einzelne Teilsysteme geben. Synchronisation und Abstimmung ist daher nur durch Zusammenwirken, d.h. durch *Kommunikation* und *Koordination* im Sinne *offener Systeme* möglich.

In [2.18] wurde daher ein sog. *Kommunikations-Informationssystem* (KIS) vorgeschlagen, welches die Inkompatibilitäten der einzelnen Teilsysteme untereinander abfängt und zugleich für den Zugang zu anderen Systemen und für den Zugriff auf externe Daten und Methoden die Grundlage bildet. Das KIS kann als Schaltstelle auf jeder Ebene eingesetzt werden, d.h. als Knoten im System kann ein ganzes UIS eines Landes aber auch ein einzelnes Fachinformationssystem gelten - auch in gemischter Weise.

Es existiert weder eine Datenbankschnittstelle noch eine Dateischnittstelle noch eine Datenträgerschnittstelle, da jene jeweils bestimmte Voraussetzungen an Hardware oder installierter Software verlangen, was wegen der Autonomie der Teilsysteme nicht verlangt werden kann. Das KIS bietet eine nach OSI standardisierte *Kommunikationsschnittstelle* an, welche zunächst den puren Austausch der Daten über unterschiedliche Hard- und Softwareplattformen gewährleisten kann.

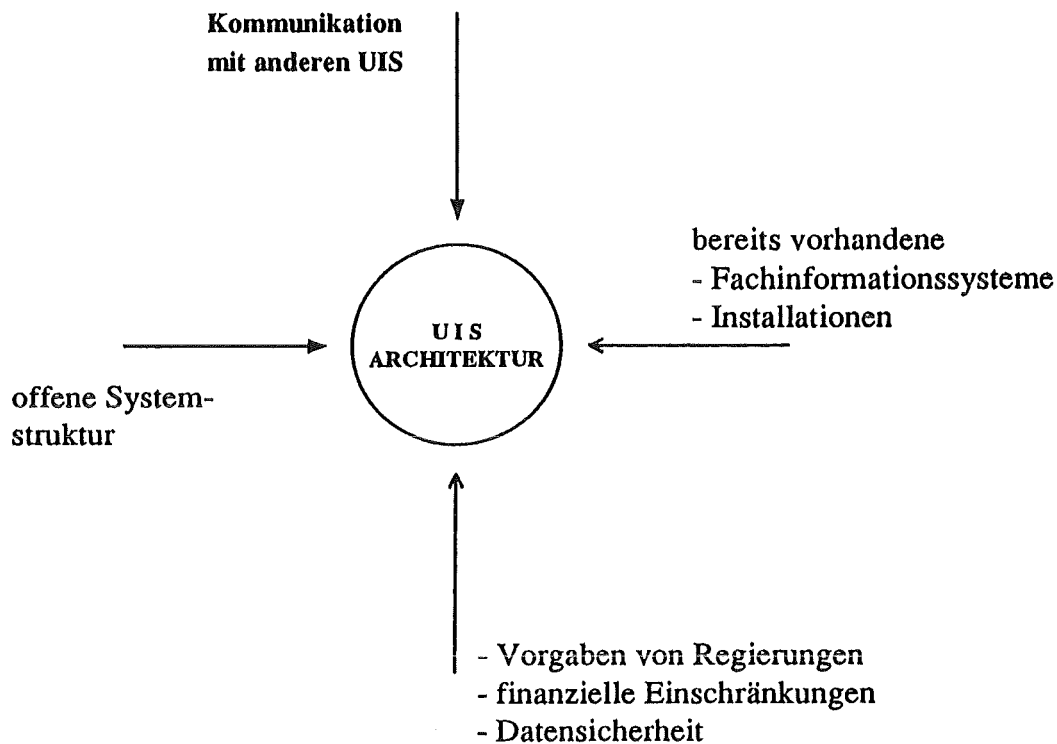


Abb. 2.8: Einflußgrößen auf die Systemarchitektur

Darüberhinaus wird das KIS als Informationsbörse fungieren. Oftmals wird es schlichtweg nicht jedem Informationskonsumenten bekannt sein, wo welche Daten zu einem bestimmten Thema existieren. Ebenso ist es heute nicht möglich zu sagen, ob es z.B. über ein bestimmtes Gebiet Daten gibt und wenn ja welche und wo. Es existiert ein großes Informationsdefizit über die Datenbestände an sich.

Hierzu wird über das KIS Zugang zu einer Reihe von Katalogen gewährleistet, die auf jeder beliebigen Ebene eines UIS aufgebaut werden. Abb. 2.10 zeigt die "erste Komplexitätsebene" eines nationalen UIS, eine von drei in [2.18] angedeuteten Ebenen. Die einzelnen Kataloge haben folgende Funktionen:

- Der *Umweltdatenkatalog* ist ein Übersichtsverzeichnis über die Gesamtheit der bei den Fachstellen erhobenen und gespeicherten Umweltdaten. Durch ihn wird die Transparenz der Datenbestände für fachübergreifende Aufgaben erhöht. Mehrfaches Erfassen und Speichern von Daten soll dadurch vermieden werden, daß einmal erfaßte Daten auffindbar und verfügbar werden.

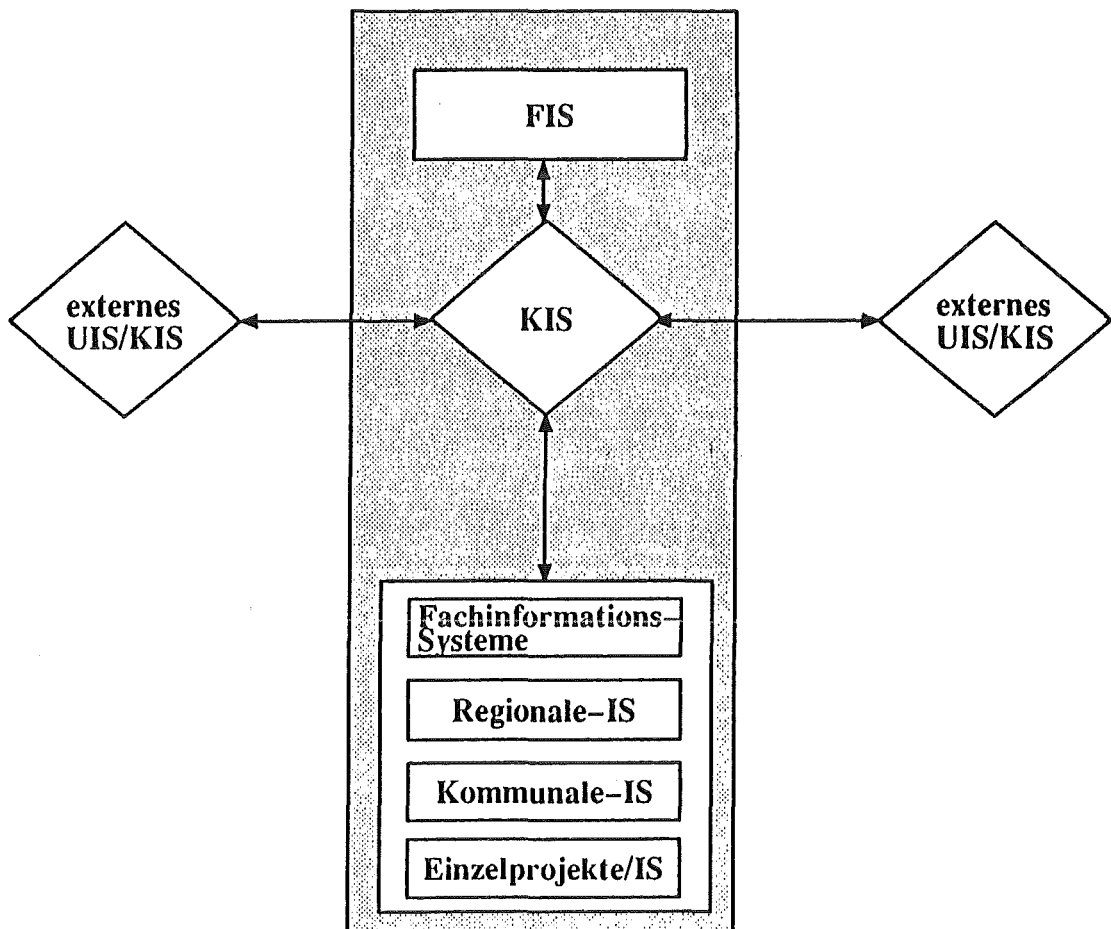


Abb. 2.9: UIS mit integriertem KIS

- Der *Methodenkatalog* stellt ein Verzeichnis dar, in dem die fachbezogenen und fachübergreifenden DV-Methoden dokumentiert werden. Durch den Methodenkatalog soll die Übersicht über die vorhandenen mathematischen, statistischen, graphischen und anderen Verfahren und Modelle erhöht und ihre gemeinsame Nutzung erleichtert werden. Redundante Programmentwicklung soll dadurch vermieden werden.
- Der *Installationskatalog* ist ein Verzeichnis der Hardware- und Softwareplattformen eines einzelnen Teilnehmers im System.
- *Umweltdatenbanken* speichern die bei den fachbezogenen Umweltaufgaben erhobenen Umweltdaten und stellen die Datengrundlage für die Bewertung komplexer Umweltprobleme dar.

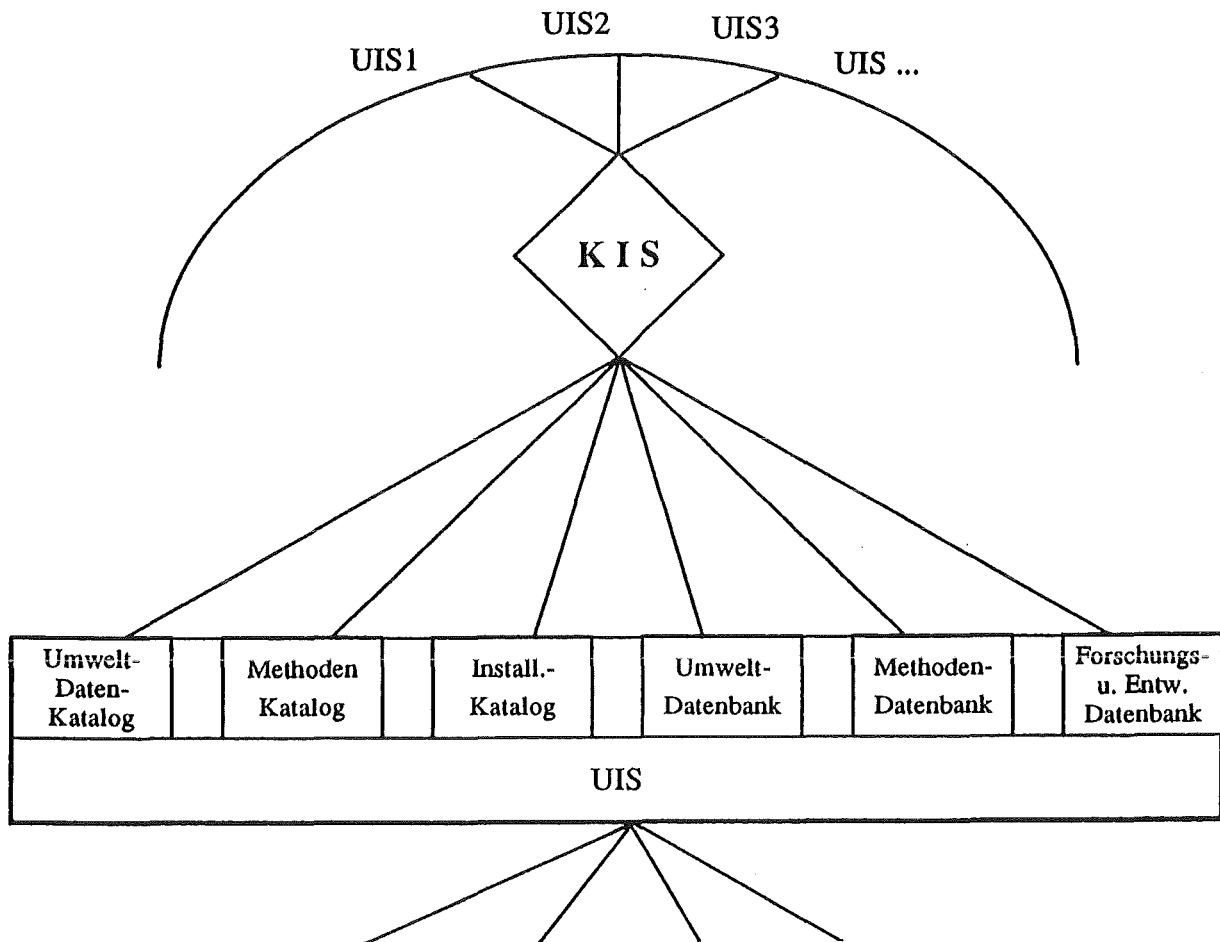


Abb. 2.10: UIS-Komplexitätsebene 1

- *Methodendatenbanken* sind Sammlungen von Methoden zur Lösung fachübergreifender Aufgaben.
- Schließlich können noch *Forschungs- und Entwicklungsdatenbanken* existieren, welche laufende Projekte dokumentieren.

Diese Komponenten stellen einen ersten Diskussionsvorschlag zur Organisation eines KIS dar. Die Aufzählung ist mit Sicherheit noch von unvollständiger Natur.

Betrachtet man Abb. 2.10 und 2.11, so sieht man, daß auf jeder Ebene des Systems die gleiche Struktur existiert. Verbindet also das KIS eines nationalen UIS dieses mit anderen nationalen UIS, so kann ein KIS genauso dazu dienen, Fachinformationssysteme zu verbinden und damit ein nationales UIS aufzubauen. Die Gliederung des

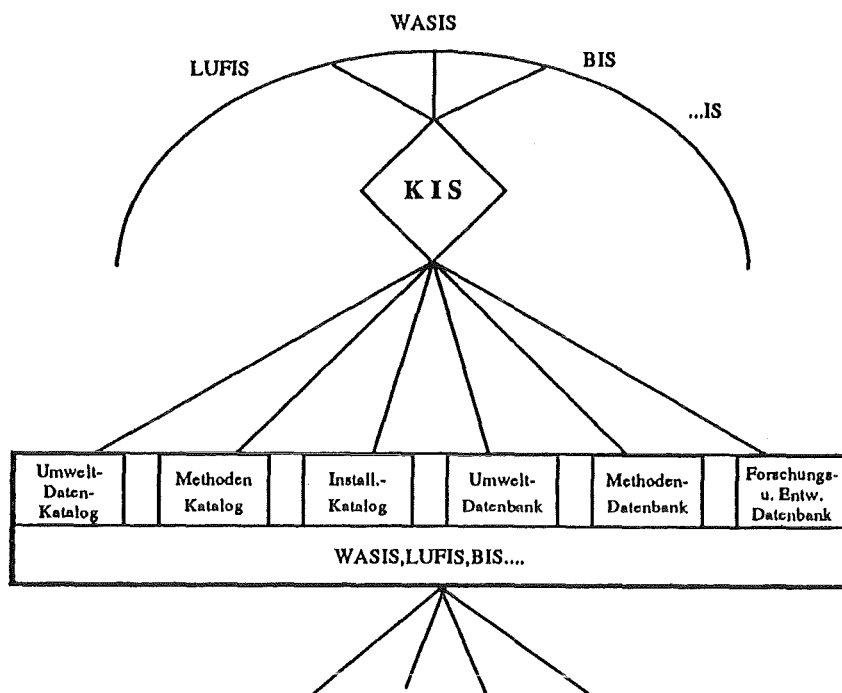


Abb. 2.11: UIS-Komplexitätsebene 2

Gesamtsystems ist bei diesem Konzept vollkommen frei: es wäre genauso möglich, Hierarchieebenen analog den Verwaltungsgrenzen und Ländergrenzen aufzubauen wie es möglich wäre, die Gliederung des Gesamtsystems thematisch aufzubauen (Wasser, Luft, Boden,...). Ein gemischter Aufbau ist denkbar wie es auch sinnvoll und möglich ist, bestimmte Informationen an bestimmten Orten zu konzentrieren (z.B. beim Deutschen Wetterdienst).

Aus technischer Sicht sieht ein solches großes verteiltes UIS wie in Abb. 2.12 aus: Eine Vielzahl von unterschiedlichen Hardware-Plattformen nimmt an einem gemeinsamen Netzwerk teil. Das KIS ist die softwaremäßige Anpassung all dieser Systeme aneinander. Alle KIS gemeinsam bilden die Informationsplattform für die Verwendung der Daten und Methoden innerhalb des Gesamtsystems.

Die Unterschiede zu den Gegebenheiten bei technischen Prozessen, wie sie im letzten Abschnitt geschildert wurde, liegen in folgenden Punkten:

- Während bei technischen Prozessen von vorne herein definiert ist, wo welche Daten und Informationen im verteilten System angesiedelt sind,

muß dies bei UIS nicht der Fall sein. Es ist durchaus wahrscheinlich, daß nicht exakt bekannt ist, wo welche Informationen auffindbar sind.

- Bei technischen Prozessen ändert sich das Softwaresystem während des Betriebs i.d.R. nicht oder nur wenig, weder die Datenmengen noch die Struktur des Gesamtsystems. Wenn es sich doch ändern sollte, so ist das allen Beteiligten bekannt. Bei UIS hingegen werden sich sowohl die Datenmengen, die Methoden, als auch die Strukturen (d.h. auch die zugrunde liegende Hardware und Software) permanent ändern, insbesondere deswegen, weil ein solches System über lange Zeit in Betrieb sein muß und ständig neue Anforderungen zu erwarten sind.

Für die Visualisierung hat das folgende Konsequenzen: es wird bei UIS notwendig sein, *das Rechnersystem an sich und dessen Inhalte graphisch zu visualisieren*. Bei technischen Prozessen geschieht das auch, allerdings zu einem anderen Zweck: dort geht es hauptsächlich darum, daß auch die Zustände der Rechnersysteme zum

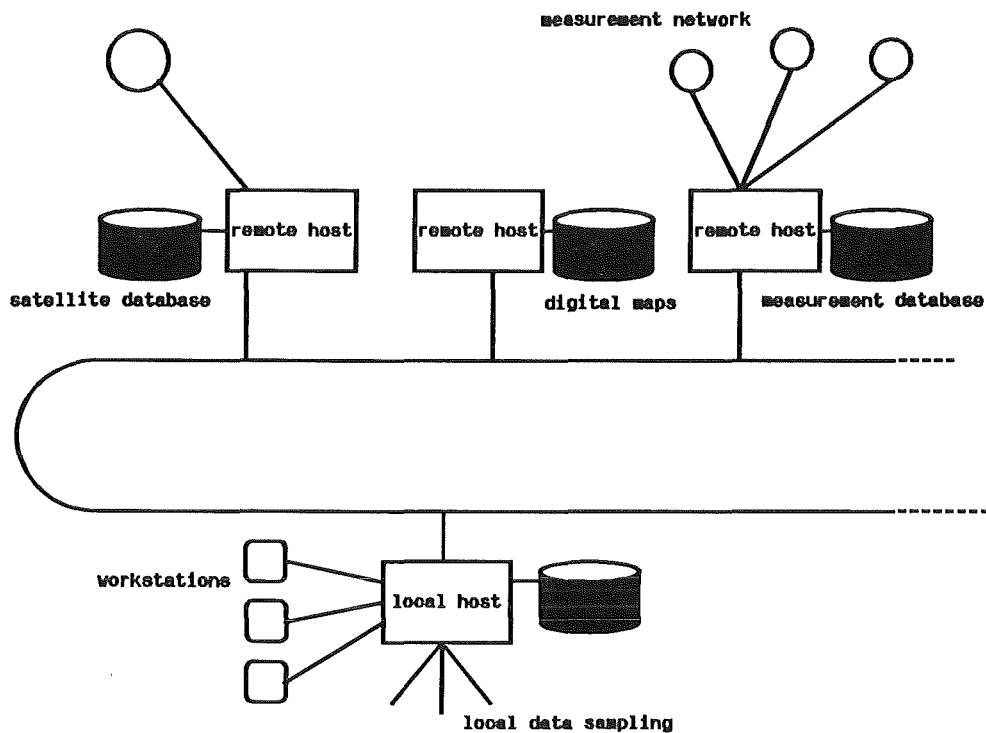


Abb. 2.12: Technische Sicht eines UIS (aus [2.8])

globalen Anlagenzustand beitragen und daher für den Betrieb einer Anlage wichtig sind.

Bei UIS hingegen gibt es für die Visualisierung des Informationssystems zwei überwiegende Gründe:

- Die Inhalte des UIS und der jeweilige Ort einer gewünschten Information müssen visualisiert werden, damit gewünschte Daten überhaupt aufgefunden werden können.
- Gerade weil sich das verteilte System permanent verändert, ist eine solche Visualisierung notwendig.

Die Visualisierung dient in diesem Fall der eigentlichen Darstellung von Daten, der graphischen Suche nach Daten, der graphischen Abfrage von Daten und sollte auch noch den Datenaustausch unterstützen. *Die Visualisierung wird zur Visualisierung in einem komplexen System.*

Aus der Notwendigkeit einer schnellen Anpassung an neue Anforderungen aus der Praxis heraus müssen die Endbenutzerschnittstellen effizient entworfen werden. Sonst ist eine geeignete Dynamik des Softwaresystems schwerlich zu gewährleisten. Hierzu werden Werkzeuge benötigt, mit denen es möglich wird, Dialogsysteme schnell und komfortabel zu erstellen.

Ein Blick in die bislang vorliegende Literatur zeigt, daß in der Umweltinformatik noch kaum mit graphischen Werkzeugen für die User-Interface-Gestaltung gearbeitet wird. Vermutlich ist die Zeit für den Einsatz von solchen Werkzeugen wie User Interface Management Systems [2.9] (in der Folge UIMS genannt) auch noch nicht reif. Hier ist der state-of-the-art in den letzten Jahren sehr in Bewegung.

Bei UIS sind allerdings besondere Anforderungen gegeben. Es reicht nicht aus, brauchbare UIMS einzusetzen. Vielmehr sind die Anforderungen an vektor-graphische Darstellungen (Karten,3D,...) im Umweltbereich recht hoch. Hier scheint es noch an der Integration der Techniken aus dem Bereich der UIMS und aus der übrigen Computergraphik zu mangeln, was nicht verwundert, da dies kein triviales Problem ist.

Es wird daher in der Zukunft auch notwendig sein, über die Integration von Visualisierungstechniken [2.6,2.22,2.23,2.28] in Werkzeuge für Benutzerschnittstellen nachzudenken.

Dieses Kapitel führte in die Konzepte ein, die der vorliegenden Arbeit zugrunde liegen. Die Anforderungen bei der *Visualisierung in komplexen Systemen* sind derart, daß Werkzeuge für graphische Benutzerschnittstellen von vorne herein auf einem verteilten und offenen Informationskonzept beruhen sollten.

Sie sollten in der Lage sein, externe Informationen direkt und automatisch in ein lokales Endbenutzersystem zu integrieren und dem Benutzer unter Verwendung von Wissen über das natürliche System und über das DV-System wertvolle Hilfestellungen bei seiner Arbeit zu geben. Im folgenden Kapitel werden die informationstechnischen Grundlagen für mögliche Ansätze vorgestellt.

3. Grundlagen

3.1 Offene Systeme

Der Begriff des offenen Systems ist vor allem seit der Normungsbemühung der ISO hinsichtlich offener Netzwerke ein Begriff geworden. Das OSI-Schichten-Modell [3.1,3.5] definiert eine Menge von Protokollen und Diensten, die die Informationsverarbeitung in heterogenen Rechnernetzen standardisieren. Ebenso ist der Begriff des offenen Systems von den Bemühungen gekennzeichnet, einheitliche Benutzerschnittstellen für eine große Klasse von Rechnern zur Verfügung zu stellen [3.52].

Der in dieser Arbeit verwendete Begriff des offenen Systems geht über dies weit hinaus. Er ist hauptsächlich gekennzeichnet durch die Fragestellung, wie in großen, verteilten, heterogenen Strukturen, die autonom operieren, eine funktionierende Kooperation einzelner Teilsysteme aufgebaut werden kann.

Der in dieser Arbeit verwendete Begriff ist aus den praktischen Anforderungen bei verteilten technischen Prozessen und verteilten Informationssystemen für den Einsatz im Umweltschutz abgeleitet. Er deckt sich weitestgehend mit den Vorstellungen, die in [3.20] dargestellt werden. Hierauf soll in der Folge näher eingegangen werden.

Unter einem offenen System wird hier ein System verstanden, das

- unbegrenzt ist,
- (mehr oder minder) frei zugänglich ist,
- aus heterogenen und autonomen Teilsystemen besteht,
- und in jeglicher Hinsicht dezentral aufgebaut ist und operiert.

Das Zusammenspiel innerhalb eines solchen offenen Systems wird weder durch globale Entwurfsentscheidungen für einzelnen Teilsysteme noch durch die Verwendung einheitlicher Rechnerarchitekturen oder Betriebssysteme erzwungen, sondern *einzig und alleine durch Kommunikation*.

Der wichtigste Vorteil dieser Vorgehensweise liegt darin, daß jede Komponente (Teilsystem) die für sie günstigste Umgebung zur Implementierung wählen kann.

3.1.1 Unbegrenztheit

Unbegrenztheit heißt: frei von Restriktionen jeglicher Art. Einzelne Komponenten können beliebig geographisch verteilt sein. Sie können beliebigen organisatorischen Konzepten unterworfen werden und gemäß der lokal sinnvollen Bedingungen betrie-

ben werden. Es existieren keine globalen Konventionen über die Art und Weise, wie einzelne Komponenten implementiert und betrieben werden, mit einer Ausnahme: sie müssen kommunikationsfähig sein und sich in der Art und Weise der Kommunikation an das globale Konzept anpassen.

3.1.2 Zugang

In offenen Systemen soll theoretisch freier Zugang zu den einzelnen Komponenten existieren. Offene Systeme haben gemäß [3.20] eine wechselnde Benutzergemeinschaft, die beliebigen Zugang zu jeglichen Ressourcen hat und beliebige Dienste benutzen darf.

Natürlich muß der "freie Zugang" in der Praxis durch eine Menge von Zugriffsrechten und Zugriffskontrollen gesichert werden, um absichtlichen oder unfreiwilligen Mißbrauch zu verhindern. Dies ist nötig bei Umweltinformationssystemen und zwingend bei technischen Prozessen, wo strenge Sicherheitskriterien erfüllt sein müssen.

Die Philosophie des offenen Systems ist allerdings grundsätzlich *nicht* die Abschottung einzelner Komponenten, sondern das zur Verfügung stellen von Informationen und Diensten innerhalb eines geordneten Rahmens.

3.1.3 Heterogenität

Jedes Teilsystem bzw. jede Organisation innerhalb des offenen Systems hat die Freiheit, beliebige Subsysteme so einzurichten, daß sie den spezifischen lokalen Gegebenheiten, Aufgaben und Anforderungen optimal angepaßt sind. In [3.20] wird zwischen syntaktischer und semantischer Heterogenität unterschieden.

Unter *syntaktischer Heterogenität* wird die Verwendung unterschiedlichster Hardware, Betriebssysteme und Softwarekomponenten verstanden. Hierunter fallen z.B. auch die Form der Datenrepräsentation, Dateistrukturen und die Verwendung von unterschiedlichen Programmiersprachen. Unter *semantischer Heterogenität* werden unterschiedliche Betrachtungsweisen und Interpretationen von Informationen und Ereignissen verstanden.

Heterogenität ist eine zwingend notwendige Eigenschaft von großen Informationsverbänden, wenn einzelne Komponenten spezifische Aufgaben (auch für andere Komponenten mit) übernehmen sollen, weil sie für diese Aufgaben besonders geeignet sind. Die derzeit teilweise zu erkennenden Trends, UNIX (oder ein beliebiges anders Betriebssystem) zu einem Allheilmittel zu erklären, halten vor einer kritischen Diskussion nicht Stand, wenn man offene Systeme anstrebt. Es ist so und wird wahrscheinlich für immer so bleiben, daß zu bestimmten Zeiten für spezielle Aufgaben spezielle Umgebungen besonders geeignet sind, wie es z.B. derzeit der Vektorrechner für die Lösung großer numerischer Probleme ist (z.B. bei der Lösung großer

partieller Differentialgleichungssysteme). Auch in [3.22] wird bemerkt, daß die Forschung über Betriebssysteme derzeit zu sehr durch die Bemühungen gekennzeichnet ist, kompatibel zu UNIX zu bleiben und daß zukünftige Entwicklungen davon gekennzeichnet sein sollten, über Ideen und Konzepte der UNIX-Philosophie hinauszugehen.

3.1.4 Autonomie

Autonomie heißt, daß Teilsysteme in einem offenen System unabhängig und selbständig an unterschiedlichen Orten unter unterschiedlicher Autorität operieren. Jedes Teilsystem besitzt eigene Verhaltensweisen und bestimmt diese einzig und allein selbst. Es ist unabhängig von den Bedingungen seiner Umwelt. Daher kann ein Beobachter von außen weder interne Zustände noch interne Verhaltensweisen beeinflussen.

Im offenen System gibt es gemäß [3.20] keine globale Information und keine global bestimmte Verhaltensweise. Lediglich die Fähigkeit zur Kommunikation wird allgemein angenommen.

Dieser Umstand verdient eine tiefere Diskussion. Ist z.B. ein an einer bestimmten Stelle geführter Katalog über verfügbare Information innerhalb eines bundesweiten Umweltinformationssystems globale Information? Bestimmt dieser Katalog globale Verhaltensweisen im Gesamtsystem, d.h. Verhaltensweise einzelner Teilsysteme? Letztere Frage kann klar mit nein beantwortet werden; er darf dies nicht tun. Aber beinhaltet er globale Information und wenn, wie wird sie auf dem Laufenden gehalten? Im Hinblick auf hunderte oder tausende von Teil-Informationssystemen, die sich laufend ändern, ist dies ein schwieriges Unterfangen. Die Frage nach globaler Information in Umweltinformationssystemen ist wohl derzeit noch nicht zu beantworten. Aber vielleicht wird es ein Ansatz der Zukunft sein, daß der "globale" Katalog sich aus der Menge der Teilkataloge zusammensetzt, die über entsprechende Abfragemechanismen erreichbar sind. Damit besteht der "globale" Katalog nicht mehr aus globaler Information sondern aus lokaler Information in obigem Sinn.

3.1.5 Dezentralität

Dezentralität bedeutet: alle Komponenten werden als gleichberechtigt betrachtet. Es gibt keine Autorität, die andere organisiert und globale Konzepte, Regeln oder Prinzipien erzwingt. Ausgenommen sind diejenigen Mechanismen, die der Kommunikation und Kooperation dienen und damit ein Zusammenspiel im offenen System erst möglich machen.

Eine wichtige Tatsache ist, daß keine Komponente je über alle Zustände aller anderen Komponenten informiert sein kann. Daher ist eine globale Konsistenz nicht erreichbar. In offenen Systemen gibt es keinen globalen Zustand, kein globales

Objekt und (genau genommen) keine kooperierende Aktion, die zu einem gemeinsamen Ziel führt.

Der Begriff der Dezentralität kann auch folgendermaßen aufgefasst werden: mehrere Teilsysteme arbeiten örtlich verteilt und führen unterschiedliche Aufgaben aus. Dies sei als *örtliche Dezentralität* und *funktionale Dezentralität* bezeichnet. Diese Begriffe überschneiden sich mit den oben erwähnten Begriffen der Unbegrenztheit und der Autonomie. In der Prozeßtechnik wird der Begriff der Dezentralität aber eher in diesem Sinne verwendet. Dort sind die Vorzüge gegenüber zentralen Systemen hinlänglich bekannt. Sie reichen von geringerem Aufwand bei der Verkabelung bis hin zur verbesserten Sicherheit des Gesamtsystems bei lokalen Ausfällen. Bei der Überwachung, Führung und Steuerung technischer Anlagen bieten sich aufgrund der örtlichen und funktionalen Verteilung der Prozesse ohnehin dezentrale Konzepte an.

Ob man nun die Begriffe in ersterem oder in letzterem Sinne definiert, ist letztlich für das Verständnis großer, verteilter, offener Systeme unerheblich und dürfte eine Angelegenheit des persönlichen Geschmacks sein.

3.1.6 Zusammenarbeit in offenen Systemen

Zusammenarbeit in offenen Systemen kann nach den Bemerkungen des letzten Abschnitts einzig und alleine durch Kommunikation geschehen. Die wichtigste Voraussetzung, die Komponenten eines offenen Systems erfüllen müssen, ist daher die Fähigkeit zur Kommunikation.

Durch Kommunikation können Komponenten des offenen Systems Informationen über Zustände und Ereignisse in anderen Komponenten gewinnen. Die Komponente erweitert dadurch ihre eigene Sichtweise über die eigene Umgebung hinaus. In [3.20] wird argumentiert, daß aufgrund der Tatsache, daß keine globalen Objekte existieren, der Informationsaustausch nur durch message-passing, also durch Übermittlung von Botschaften geschehen kann. Damit geschieht eine teilweise Duplikation der Information.

Für den Informationsaustausch muß im Gegensatz zu den oben erwähnten Randbedingungen für die Komponenten *sowohl syntaktische als auch semantische Homogenität* gewährleistet sein.

Syntaktische Homogenität wird durch die Einführung allgemein akzeptierter Standards und Protokolle (hoffentlich) in naher Zukunft erreicht werden. Die OSI-Normen spielen hierbei derzeit mit Sicherheit die maßgebliche Rolle. Analog hierzu ist zu wünschen, daß sich das Xwindows-Konzept [3.52], welches derzeit als Industriestandard angesehen wird, auf dem Gebiet der Benutzerschnittstellen durchsetzen wird. Auch die Einbettung von Graphik-Standards wie z.B. PHIGS [3.53] ist hier mit Interesse zu verfolgen. Die Sicherstellung syntaktischer Homogenität für den Informationsaustausch ist schon ein gewaltiges Problem an sich und betrifft in der Hauptsache Normungsprozesse.

Semantische Homogenität hingegen wird durch globale Vereinbarungen festgelegt, die im offenen System getroffen werden. Das bedeutet, daß für jede Information bei deren Austausch von einer Komponente zur anderen deren Bedeutung eindeutig sein muß. Auch darf die Bedeutung der Information nicht unterschiedlich sein, wenn die Information von einer beliebigen Komponente zu einer weiteren beliebigen Komponente ausgetauscht wird. Die Definition, Sicherung und Überwachung syntaktischer Homogenität innerhalb eines großen offenen Systems dürfte ein noch schwerwiegenderes Problem sein. Halten sich die damit verbundenen Schwierigkeiten bei einer Vielzahl technischer Prozesse noch in Grenzen, so sind sie bei globalen Umweltinformationssystemen derzeit mit Sicherheit nicht voraussehbar.

3.1.7 Ansätze zur Kooperation in offenen Systemen

Der in [3.20] beschriebene Ansatz OAI (Open application and interaction systems) definiert den Begriff der *Rolle* und des *Dienstes* für einzelne Komponenten des offenen Systems. Unter der Rolle wird der Anteil der Komponente an der Gesamtaktivität bezeichnet. Unter einem Dienst wird in OAI der abstrakte Begriff der Sichtweise einer Komponente auf eine Funktion einer anderen Komponente verstanden.

Es wird desweiteren zwischen *applikationsspezifischen* und *systemspezifischen* Vereinbarungen unterschieden. Applikationsspezifische Vereinbarungen haben eine lokal und zeitlich begrenzte Bedeutung. Sie werden von jeder Komponente eigenständig definiert. Systemspezifische Vereinbarungen hingegen müssen eine globale und langfristige Bedeutung besitzen, um den dauerhaften funktionierenden Informationsaustausch zu gewährleisten. Sie müssen durch globale Entwurfsentscheidungen und Standards eingeführt werden.

Ein wichtiger Begriff ist desweiteren die *Qualität eines Dienstes* (service quality). In vielen Fällen wird ein Dienst eine Anfrage nach einer spezifischen Information nicht mit der gewünschten Rate oder in der gewünschten Qualität befriedigen können, so z.B. wenn eine gewünschte Genauigkeit nicht vorhanden ist oder eine gewünschte Rate eine zu große lokale Rechenleistung erfordern würde.

Daher werden globale Strategien benötigt, die die Qualität von Diensten und deren Handhabung definieren. Außerdem müssen Mechanismen eingeführt werden, die mögliche Dienstqualitäten transparent und zugänglich machen (in einer Art Informationsbörse). Eine Schwierigkeit besteht darin, daß man sinnvollerweise eine begrenzte aber dennoch ausreichende Menge von Qualitätskriterien global vereinbaren muß.

In der Standardisierungsgruppe für ODP (open distributed processing) wird als Anhaltspunkt zum Entwurf von verteilten Systemen eine Menge von *viewpoints* angegeben. Es folgt eine kurze Beschreibung.

- Der *enterprise viewpoint* ist die Beschreibung des technischen Systems einer Komponente bzw. eines Unternehmens. Er beschreibt explizit Gebiete, Ressourcen und beteiligte Personen.
- Der *technology viewpoint* ist die Beschreibung der realisierten und betriebenen Hardware, also der Maschinen, Peripheriegeräte, Betriebssysteme, Softwarekomponenten etc.
- Der *information viewpoint* definiert die Informationsstrukturen und Informationsflüsse und modelliert die Regeln, wie Information verwendet und manipuliert werden kann.
- Der *computational viewpoint* bezieht sich auf die Operationen und Prozesse, die Information generieren und manipulieren. Ziel ist hierbei, Applikationen so aufzubauen, daß sie unabhängig von Netzwerk und Maschine werden.
- Der *engineering viewpoint* schließlich beschreibt die Aspekte, die notwendig sind, um das verteilte System effizient, zuverlässig und transparent zu betreiben.

Die vorstehend genannten Betrachtungsweisen sollen Hinweise geben, wie man an die Konzeption offener verteilter Systeme herangehen kann. Sie sind derzeit Gegenstand der Forschung und sollen an dieser Stelle nicht weiter vertieft werden.

3.2 Verteilte Systeme

Unter einem verteilten System wird ein DV-System verstanden, bei dem mehrere Daten- und/oder Funktionskomponenten eines Programmsystems auf mehrere zu einem Netz zusammengeschlossenen Rechner verteilt sind [3.1]. Jeder der Knoten besitzt, wie schon im vorangegangenen Abschnitt gesagt, Autonomie, d.h. konkret auch autonom arbeitende CPU-, Speicher- und Peripherieressourcen.

In verteilten Systemen erfolgt also eine Dezentralisierung der Kontrollstruktur. Es ist keiner der Komponenten möglich, einen globalen, konsistenten Systemzustand in Erfahrung zu bringen.

Als Vorteile von verteilten Systemen können unter anderem genannt werden:

- der Leistungsgewinn (durch parallele Arbeit)
- die Fehlertoleranz (durch Redundanz)
- die Verfügbarkeit des Gesamtsystems (durch Umschaltung).

Unter einem Prozeßsystem versteht man eine Menge von kooperierenden Prozessen, die auf gemeinsame Informationen zugreifen müssen, um gemeinsam eine oder

mehrere übergeordnete Aufgaben erledigen zu können. Aus Kap. 2 wird ersichtlich, daß wir es im Kontext der vorliegenden Arbeit grundsätzlich immer mit Prozeßsystemen zu tun haben.

In der Prozeßtechnik werden seit langem zur Automatisierung technischer Anlagen verteilte Konzepte eingesetzt. Die oben genannten Vorteile haben dabei in der Prozeßtechnik ein besonderes Gewicht.

Der Zugriff auf verteilte Informationen bedingt Synchronisationskonzepte, um Fehler beim Zugriff auf verteilte Daten zu vermeiden. Konzepte wie Semaphore und Monitore [3.1] sind hier hinlänglich bekannt und finden verbreitet Anwendung.

Durch die zunehmende Vernetzung der Rechnerumgebungen und auch durch die Möglichkeit des schnellen Datentransfers haben verteilte Systeme in den letzten Jahren verstärkt auch Bedeutung bei der Entwicklung von Betriebssystemen und Anwendungssoftware erlangt [3.6-3.13,3.38-3.44].

Die in Kap. 4 vorgestellte Architektur setzt bei der Verteilung von Information auf der Anwendungsebene und nicht auf der Betriebssystemebene auf. Im folgenden Abschnitt sollen Begriffe verteilter Konzepte vorbereitet werden. Eine Einordnung des in Kap. 4.2 vorgestellten Kommunikationsmodells in eines dieser Konzepte ist nicht vollständig möglich. Am ehesten mag das Kommunikationsmodell als eine asynchrone Erweiterung von *remote procedure calls* angesehen werden.

3.2.1 Konzepte

Die Grundlage vieler verteilter Konzepte ist der *remote procedure call* (RPC) [3.45]. Beim RPC handelt es sich um einen Mechanismus, bei dem einem lokalen Prozeß ein lokaler Funktionsaufruf vorgespiegelt wird, wobei die Ausführung der Funktion auf einem *remote host*, also einem anderen Knoten im Rechnernetz, geschieht. Der RPC dient dem Zugriff auf externe Information und der Ausführung externer Operationen.

Die Vorgehensweise ist folgende: führt ein lokaler Prozeß eine Funktion

LocalFunction (input-parameters, output-parameters)

aus, so werden innerhalb dieser Funktion die Funktionsargumente in eine Meldung verpackt, die an den *remote process* geschickt wird. Dieser Vorgang wird *marshaling* genannt [3.18]. Der *remote process* entpackt die Parameter (*unmarshaling*) und führt die eigentliche Funktion

ImplementationFunction(input-parameters, output-parameters)

aus. Während der Ausführung dieser Funktion geht der lokale Prozeß durch ein blockierendes Lesen (der Ergebnisse) in einen Wartezustand über. (Auf die Blockie-

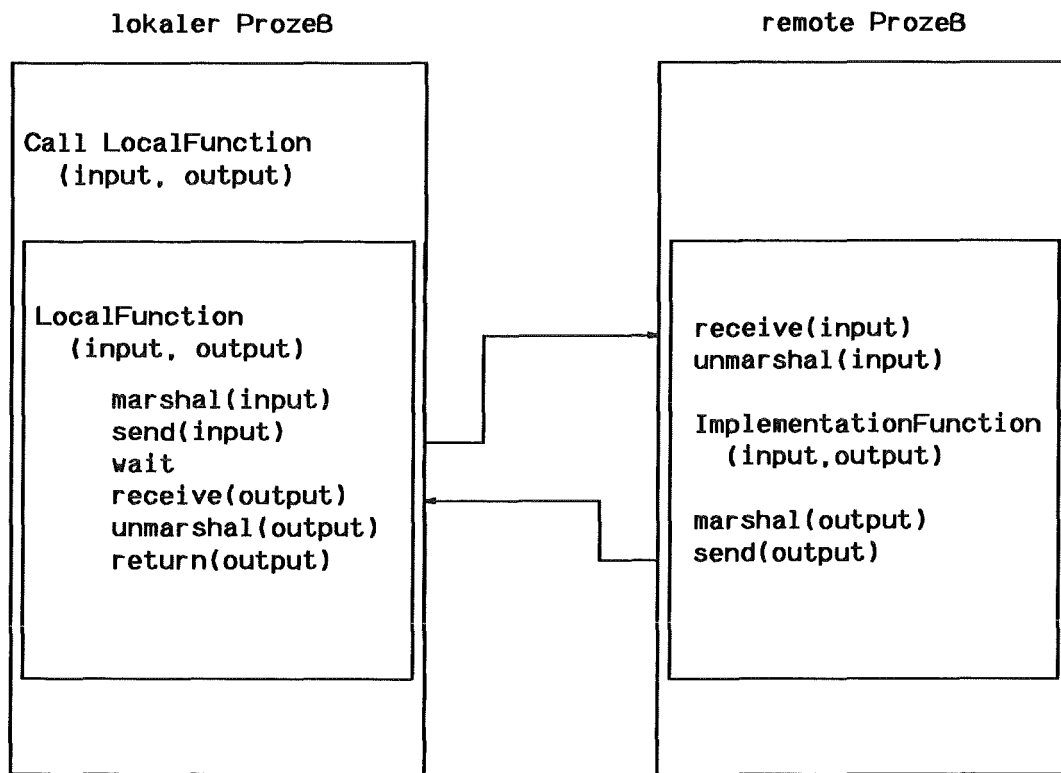


Abb. 3.1: Remote Procedure Call (RPC)

zung des Prozesses sei in der Folge besonderes Augenmerk gerichtet, da bei dem in Kap. 4.2 vorgestellten Kommunikationskonzept eine blockierende Vorgehensweise nicht möglich ist). Nachdem die *ImplementationFunction* ausgeführt worden ist, werden die Ergebnisse wieder verpackt und versendet. Der lokale Prozeß entpackt die Ergebnisse und liefert sie als Ergebnisse der Funktion an das aufrufende Programm zurück (Abb. 3.1). Der RPC basiert auf dem Rendezvous-Konzept, d.h. auf dem synchronen, blockierenden Senden und Empfangen von Nachrichten [3.30].

Da der Vorgang des marshalings bei jedem RPC analog vor sich geht, werden sog. *stub-Generatoren* verwendet. Ein stub-Generator erzeugt aus dem Quellcode die nötige funktionale Schicht zwischen dem lokalen und externen Prozeß. Die Funktion, die das aufrufende Programm benutzt, ist die *stub-routine* [3.18].

Ein typisches Konzept, das den RPC benutzt, ist die *Client-Server-Strategie* [3.19,3.24]. Bei der Client-Server-Strategie hält der Server-Prozeß stellvertretend für einen oder mehrere Client-Prozesse Daten und stellt Funktionen auf diesen Daten zur Verfügung. Diese Strategie wird in der Zwischenzeit vielfach, z.B. bei Datenbankanwendungen benutzt.

In [3.46] wird ein Beispiel gegeben, wie man RPCs direkt als Spracherweiterung in die Programmiersprache C integrieren kann. Hier werden Funktionen angeboten wie INITIALIZE_SERVER und START_SERVER, letztere führt dazu, daß ein initialisierter Server damit beginnt, RPCs anzunehmen und abzuarbeiten. Eine Erweiterung von C wird z.B. durch Konstrukte wie BEGIN_TRANSACTION und END_TRANSACTION vorgenommen. Der Nachteil dieser Vorgehensweise liegt darin, daß eine Beschränkung auf C statt findet.

Walker [3.47] nimmt in seinem *futures* genannten Konzept eine Erweiterung des RPC vor. Der aufrufende Prozeß soll dabei die Möglichkeit haben, während der Ausführung des RPC parallel andere Arbeiten zu erledigen. Die Vorgehensweise wird an folgendem Code demonstriert ([3.47] entnommen):

```
FUTURE f = FInvokeSelect(object,Statement,InvokeControl);
```

```
perform local processing
```

```
FClaimSelect(f, Rows);
```

```
process Select results
```

Mit der Funktion FInvokeSelect wird der RPC angestoßen. Nachdem der Prozeß andere lokale Aufgaben erledigt hat, wird anschließend mit der Funktion FClaimSelect eine Synchronisation durchgeführt. Dieses Konzept wird in [3.47] ausdrücklich als "an asynchronous remote execution facility" bezeichnet. Ob diese Bezeichnung zutreffend ist, ist zumindest fraglich, da eine explizite Synchronisation an einem definierten Punkt des Codes statt findet. Die Möglichkeit der Verarbeitung vollkommen spontan von externer Seite auftretender Ereignisse geht zumindest aus dem Artikel nicht hervor.

3.2.2 Objekt-orientierte Vorgehensweise

Client-Server-Konzepte werden seit einigen Jahren auch mittels objekt-orientierten Techniken realisiert. Objekt-orientierte Programmierung zeichnet sich durch untereinander agierende Objekte aus, die für sich eine Ansammlung von Daten und Operationen, also einen abstrakten Datentyp, darstellen. Die Interaktion geschieht durch Nachrichtenaustausch. Der verteilte Aspekt ist also der objekt-orientierten Programmierung an sich schon inhärent. Da die Objektorientierung andererseits durch Mechanismen wie Klassenbildung, Vererbung, dynamische Bindung, u.s.w. Vorteile bei der Konstruktion großer Software-Systeme bietet, liegt es nahe, sie für

verteilte Systeme einzusetzen. Einen Überblick über die Technik des objekt-orientierten Programmierens wird in [3.48] gegeben. Neuere Publikationen zu dem Thema sind z.B. [3.49] und [3.50].

Befinden sich Objekte in einem verteilten System, so wird der ansonsten lokale Nachrichtenaustausch zwischen Objekten zu einem Nachrichtenaustausch über das Netzwerk. Das Anstoßen einer Funktion (Methode) eines Objekts (*invocation* genannt), führt daher zu einem RPC. Verteilte, objekt-orientierte Systeme kommunizieren über ähnliche wie die zuvor beschriebenen Konzepte.

Horn gibt in [3.19] einen Überblick über den Einsatz objekt-orientierter Ideen bei verteilten Systemen. Im Wesentlichen kann man von folgenden Mechanismen sprechen:

- *Port-basierte Client-Server-Konzepte:* Hier werden Objekte auf einem Server gehalten. Der Zugriff auf die Objekte erfolgt über RPCs synchron. Es kann für jedes Objekt einen *port* (Kommunikationsschnittstelle) geben oder für eine Menge von Objekten einen gemeinsamen port. Ersteres ist in dem Betriebssystem Mach der Carnegie-Mellon-Universität [3.51] der Fall, letzteres in Amoeba (CWI Center for Informatics and Computer Science Amsterdam) [3.18].
- *Remote Invocation:* Dies ist ein konkurrierendes Konzept. Jedes Objekt läuft in einem eigenen Prozeß und kann mehrere parallele Aktivitäten besitzen, die z.B. jeweils eine Methode des Objektes bedienen. Durch den overhead hat Remote Invocation Nachteile beim Laufzeitverhalten. Remote Invocation wird in Eden (University of Washington) eingesetzt [3.23].
- *Remote Fetch:* Bei diesem Konzept wird das Objekt geholt und lokal ausgeführt. Dieser Mechanismus wird in Ra und Clouds [3.19] des Georgia Institute of Technology benutzt. Durch die Unterstützung durch das Betriebssystem wird ein *verteilt shared memory* realisiert.

Werden die genannten Mechanismen direkt in Betriebssysteme integriert, so hat dies Vorteile bei der Unterstützung verteilter Anwendungen. Sie sind dann allerdings auf bestimmte Umgebungen beschränkt, was dem Gedanken des offenen Systems widerspricht. Konzeptionell sind sie natürlich auch auf Anwendungen übertragbar, die auf der Applikationsebene miteinander kommunizieren wollen.

3.2.3 Anwendbarkeit der Konzepte

Die in den letzten Abschnitten angeführten Konzepte haben für das sich uns stellende Problem zwei entscheidende Nachteile. Erstens arbeiten sie nicht vollständig asynchron. Dies ist aber notwendig, um auf spontane Ereignisse im verteilten, offenen System reagieren zu können, z.B. bei Alarmen in einer technischen Anlage oder in

einem Meßnetz. Zweitens liefern sie auf einen Aufruf eines externen Objekts genau eine Information oder sie führen genau eine Methode aus.

In Kap. 2 war von der Notwendigkeit eines Informationsmanagements die Rede. Darunter wird, wie es in Kap. 4 näher ausgeführt wird, unter anderem eine Filterung zu verschickender Information am Ausgang eines Teilsystems verstanden.

Diese Vorgehensweise bedeutet, daß auf *eine* erstmalige Anfrage an ein Datum das Datum zur Initialisierung des lokalen Wertes zurückgeschickt wird; in der Folge entscheidet aber der externe Prozeß von sich aus, wann das Datum zu aktualisieren ist. Dies geschieht aufgrund von Kriterien über die Signifikanz der Änderung des Wertes.

Es ist klar, daß sich ein solches Informationsmanagement mit den oben genannten Mechanismen in der hier gewünschten Art nicht direkt realisieren läßt. Es muß eine zweifache Erweiterung geschehen:

- Der Nachrichtenaustausch muß vollständig asynchron ablaufen, d.h. Nachrichten unterbrechen das Anwendungsprogramm.
- Es muß eine Erweiterung derart geschehen, daß auf eine Anfrage beliebig viele, zeitlich vom Verhalten des externen Prozesses abhängige, Nachrichten gesandt werden können.

Die zweite Forderung bedingt selbst wieder die erste Forderung nach Asynchronität.

In Kap. 4.2 wird ein Kommunikationsmodell eingeführt werden, welches diese Eigenschaften besitzt und auf Anwendungsebene realisierbar ist, ohne ein spezielles verteiltes Betriebssystem vorauszusetzen. Auf diese Art und Weise ist das Kommunikationsmodell auch in offenen Systemen einsetzbar.

3.3 Computergraphik und User Interface Management

Durch das verstärkte Aufkommen von dialogorientierten, graphischen Benutzerschnittstellen in Verbindung mit der Maus als Eingabegerät erhält die Computergraphik eine verstärkte Bedeutung beim Entwurf von Benutzerschnittstellen. Während vor einigen Jahren noch die Gestaltung Kommando-orientierter Benutzerschnittstellen einen breiten Raum in der Literatur einnahm, kann man ihren heutigen Anteil nahezu vernachlässigen.

Es findet also ein Verschmelzen der Gebiete Computergraphik und User-Interface-Entwurf statt. Daher sollen die Grundlagen dieser beiden Gebiete innerhalb eines Abschnitts behandelt werden, wobei die Übergänge fließend sind.

Aus der Überzeugung heraus, daß für die Gestaltung graphischer Benutzeroberflächen die objekt-orientierte Technik das derzeit beste verfügbare Konzept darstellt,

erfolgt im nächsten Abschnitt eine Beschränkung auf dieses Konzept. Der darauf folgende Abschnitt beschäftigt sich mit *User Interface Management Systems* (in der Folge UIMS abgekürzt), also mit Werkzeugen zur Gestaltung von Benutzeroberflächen.

3.3.1 Objektorientierte Graphik

Dieser Abschnitt folgt im Wesentlichen der Argumentation Wisskirchens in seinem ausgezeichneten Buch über objekt-orientierte Graphik [3.53]. Objektorientierung wird laut Wisskirchens in zwei Zusammenhängen als Begriff benutzt:

- *bei objekt-orientierten Sprachen und Werkzeugen:* In diesem Zusammenhang ist eine Klasse von Sprachen und Werkzeugen gemeint, die sich durch eine Kombination von Datenabstraktion, Objekttypen (Klassen), Vererbungsmechanismen und Nachrichtenaustausch auszeichnen.
- *bei graphischen Benutzerschnittstellen:* Hier wird darunter ein Konzept verstanden, das den direkten Zugriff (*direct manipulation*) auf Dinge (Objekte) erlaubt, die durch Symbole (*icons*) repräsentiert werden. Manipulation mit der Maus gestattet die direkte Beeinflussung und Handhabung der Objekte. Objektorientierung stellt also in diesem Zusammenhang ein Konzept zur Gestaltung von Benutzerschnittstellen dar. Für objekt-orientierte Benutzerschnittstellen braucht man nicht notwendigerweise objekt-orientierte Umgebungen.

Objekt-orientierte Umgebungen besitzen oft graphische Fähigkeiten, decken aber i.d.R. nicht das volle Spektrum der für viele Anwendungen notwendigen Graphikoperationen ab. Graphische Kernsysteme wie z.B. OSF/Motif [3.52] bieten andererseits z.B. komplexe Klassenhierarchien an, besitzen allerdings keine Möglichkeit, die Klassen und Objekte werkzeug-unterstützt zu erzeugen und zu unterhalten. Es muß nach wie vor explizit programmiert werden. Gerade in der Verwendung von objekt-orientierten Werkzeugen liegt aber eine der Stärken des objekt-orientierten Konzeptes. Daher kann man z.B. OSF/Motif nicht als objekt-orientierte Umgebung auffassen.

Bei KI-Systemen werden Kombinationen von Objektorientierung, Regeln, logischer Programmierung u.Ä. unterstützt, die graphischen Fähigkeiten sind allerdings oft prozedural eingebettet (und oftmals nicht ausreichend). Eine wirkliche Synthese wäre wünschenswert, ist aber derzeit noch wenig sichtbar.

Ein weiterer wichtiger Gesichtspunkt ist die Tatsache, daß es wünschenswert ist, daß mentale Modell einer Anwendung möglichst klar und natürlich in die interne Softwarestruktur umzusetzen. Software, die das mentale Modell identisch in den Rechner abbildet, besitzt eine hohe Softwarequalität, da sie leicht verständlich ist und dem realen Modell entspricht.

Dieser Gesichtspunkt war ein Ausgangspunkt bei der Entwicklung des XAVIA-Systems. Während der Automatisierung einer technischen Anlagen fiel immer wieder die Tatsache auf, daß von Seiten der Operateure und Betriebsleitung das Verhalten der technischen Anlage durch Aussagen in der "wenn...dann..."-Form beschrieben wurde. Mehr noch, auch das gewünschte Verhalten der Benutzerschnittstelle wurde z.B. folgendermaßen ausgedrückt:

"Wenn der Prozeß im Zustand x ist, benötigen wir die Information y_1 und y_2 , hingegen ist die Information z_1, z_2 und z_3 unnötig, wenn nicht sogar störend".

Wisskirchen argumentiert weiter, daß objekt-orientierte Konzepte von vorne herein die Abbildung der realen Welt in die Software unterstützen. Nach den Erfahrungen des Projekts kann diese Argumentationsweise voll unterstützt werden. Es wurde allerdings für unser spezielles Problem eine Kombination von Objektorientierung und Regeln realisiert, da Aussagen wie obige in die interne Softwarestruktur eingebettet werden sollten. Dabei ging der Objektorientierung das explizite message-passing verloren. Die Aktionen von Objekten und deren Interaktionen untereinander wurden durch Regelsysteme beschrieben.

Obwohl man für objekt-orientierte Oberflächen keine objekt-orientierten Umgebungen benötigt, bietet sich doch deren Verwendung an. Objekte und deren interaktives Verhalten lassen sich auch nach unseren Erfahrungen vorteilhaft auf objekt-orientierte Weise beschreiben. Detaillierte Äußerungen zu den Vorteilen sind in [3.53] nachzulesen.

Ein Hauptproblem bei der werkzeug-unterstützten Konstruktion von Oberflächen ist die Anbindung der Anwendung (Applikation) an die graphische Benutzerschnittstelle (Präsentation). Bei den aus Kap. 2 abgeleiteten Randbedingungen (Asynchronität, lebende Werte in der Graphik, verteilte Informationen,...) ist dies besonders augenfällig. Auch hier kann Objektorientierung Vorteile bieten. Dies soll kurz anhand des *model - view - controller* - Modells (MVC) erläutert werden.

Typischerweise besteht eine interaktive graphische Anwendung aus drei Teilen von Quellcode:

- code, der die internen Applikationsdaten handhabt (application model),
- code, der die graphische Ein-/Ausgabe durchführt und
- code, der zwischen den beiden ersteren vermittelt.

Der Abbildungsprozeß von der ersten Komponente auf die zweite kann dabei sehr komplex werden. Bei traditionell programmierten Systemen führt eine Änderung der graphischen Darstellung mindestens zu Änderungen in den beiden letzteren Komponenten. Insbesondere die Änderungen in der letzten Komponente ziehen dabei oftmals weitreichende Konsequenzen nach sich.

Beim MVC-Modell hingegen werden die Komponenten

- model (application),
- view (output) und
- controller (input)

streng voneinander separiert (Abb. 3.2).

Das ist bei objekt-orientierter Vorgehensweise unproblematisch, da man für jeden der drei Teile ein Objekt definieren kann. Der Austausch über alle sechs Verbindungspfade ist durch Benutzung des message-passings ebenso einfach. Das MVC-

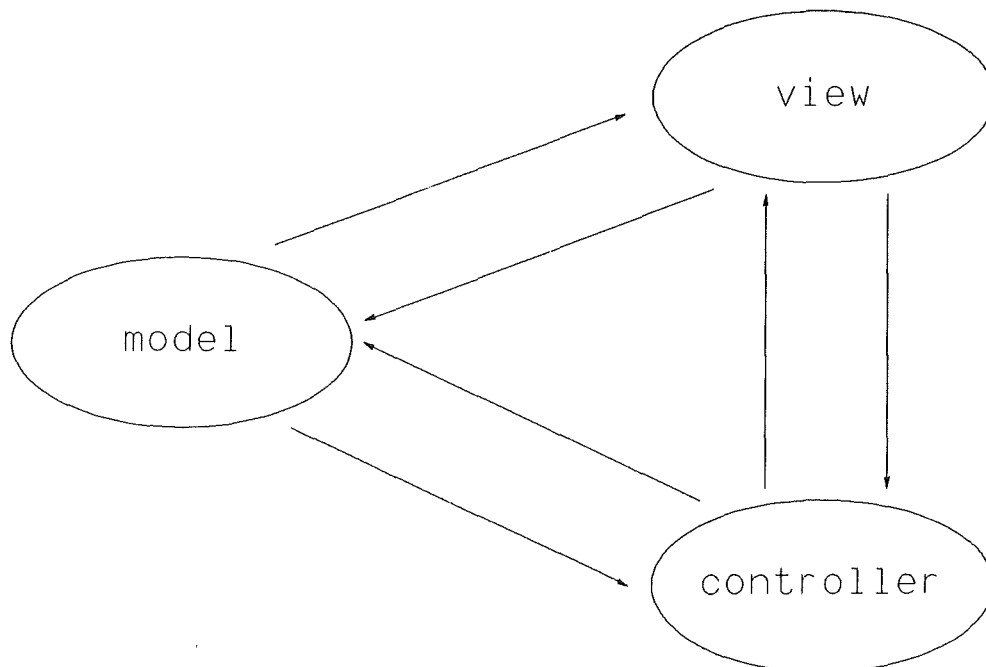


Abb. 3.2: Model - View - Controller

Konzept erlaubt also eine bessere Trennung der einzelnen Teile als dies bei traditioneller Programmierung möglich ist. Insbesondere kann man auch mehrere views einführen, um eine Anwendung je nach Wunsch anders zu präsentieren, z.B. für multilinguale Programme.

Obwohl das MVC-Konzept einen interessanten Ansatz bietet, fand es bei der vorliegenden Arbeit keine Verwendung. Hier sollte versucht werden, Objekte als Ganzes

zu modellieren, wobei die Verhaltensweisen der Objekte und der Objekte untereinander durch Regeln dargestellt werden sollten. Dies wurde für die vorliegenden Probleme als untersuchenswert betrachtet.

Objekt-orientierte Techniken finden in den letzten Jahren verstärkt bei Anwendungen und Forschungen in der Computergraphik Verwendung. Beispiele hierzu finden sich in [3.57,3.58,3.60,3.61]. Ebenso sind wissensbasierte Techniken beim Entwurf von Benutzerschnittstellen seit einigen Jahren Gegenstand der Entwicklung. Insbesondere beim Entwurf von flexiblen, adaptiven Benutzerschnittstellen sind diese Techniken, die z.B. auf Produktionssystemen beruhen, von Interesse (siehe [3.64-3.66]).

3.3.2 User Interface Management

Der Begriff des User Interface Management Systems (UIMS) wird i.d.R. auf einen 1983 in Seeheim durchgeführten Workshop zurückgeführt [3.67]. Demnach ist ein UIMS der Mittler zwischen dem Benutzer und dem Anwendungsprogramm und soll der Konstruktion von interaktiven Anwendungen dienen.

Es existieren eine Reihe von Modellen für UIMS, allerdings beziehen sich die meisten Publikationen auf die Struktur, welche beim Seeheim-Workshop angegeben wurde (Abb. 3.3).

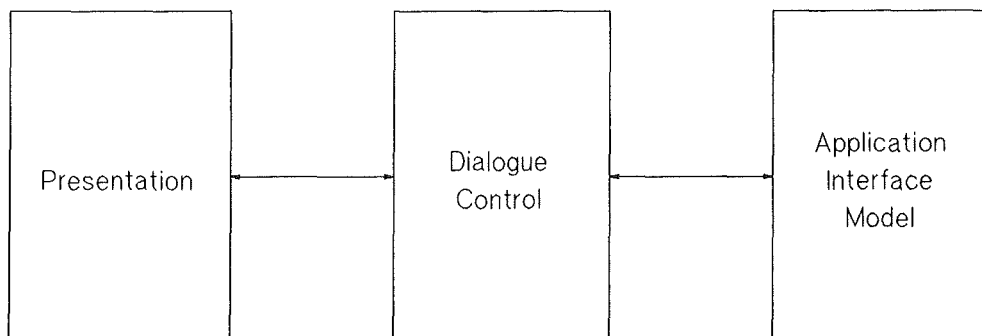


Abb. 3.3: UIMS-Struktur

Die Präsentationskomponente ist dabei die Komponente, die die eigentliche Darstellung vornimmt. Sie handhabt also die Ein-/Ausgabe. Die Dialogkomponente beschreibt die Art und Weise, wie Benutzerdialoge ablaufen. Das Applikations-Interface schließlich verfügt über die abstrakte Notation der Anwendung und muß die Anbindung an die Anwendung vornehmen. Die drei Komponenten müssen über eine definierte Syntax miteinander zusammen arbeiten.

Das Seeheim-Modell und dessen Schwächen für interaktive Anwendungen wird seit Jahren in vielen Publikationen diskutiert. So wird z.B. das Bewegen eines Objektes mit dem Cursor unter Verwendung der "klassischen" UIMS-Architektur umständlich, da alle Informationen über die Dialogkomponente in die Applikation und wieder zurück transportiert werden müssen. Für solche Interaktionen ist das MVC-Konzept mit Sicherheit angebrachter.

Neuere Entwicklungen wie z.B. SCENARIOO [3.69] nutzen die Separation der einzelnen Komponenten dazu, die Anwendung (oder mehrere Anwendungen) voneinander durch separate Prozesse zu trennen. Die Kommunikation geschieht dann mittels UNIX-streams.

Nach umfangreichen Recherchen konnte keines der bislang publizierten Konzepte unsere Bedürfnisse wirklich befriedigen. Vor allem die vollständige Asynchronität ist in keiner der einschlägigen Publikationen zu finden.

Obwohl neuere Systeme objekt-orientierte Vorgehensweisen einschlagen, gehen sie doch nicht soweit, ein verteiltes Objektmodell zu verwenden. Dies dürfte der entscheidende Unterschied zu der in dieser Arbeit vorgestellten Architektur sein.

3.4 Wissensbasierte Anwendungen

In diesem Abschnitt soll auf Anwendungen wissensbasierter Methoden eingegangen werden. Es ist nicht die Intention dieses Abschnitts, in die Grundlagen der wissensbasierten Systeme einzuführen, da im Rahmen der vorliegenden Arbeit keine tieferen Schlußfolgerungsmechanismen eine Rolle spielten. Die uns gegebene Problemstellung ließ sich schlichtweg vorteilhaft durch den Einsatz vorwärtsverkettender Regeln umsetzen. Insofern wird eine KI-Sprache als Umgebung verwendet. Zur Einführung in die KI-Methodik sei auf die Literatur verwiesen [3.70-3.75].

In den beiden folgenden Abschnitten soll vielmehr auf einige Anwendungsgebiete hingewiesen werden, in denen wissensbasierte Systeme heute zur Führung technischer Anlagen und in der Umweltinformatik eingesetzt werden. Die Darstellung hat im Wesentlichen einen aufzählenden Charakter. Auch hier sei auf die Originalliteratur hingewiesen.

3.4.1 Wissensbasierte Methoden in der Prozeßtechnik

Wissensbasierte Ansätze sind bei der Führung technischer Anlagen auf einer Reihe von Gebieten einsetzbar, so z.B. bei

- übergeordneten Regelungsstrategien (supervisory control),
- Optimierung,
- Ablaufplanung (scheduling),
- Alarm-Management und Alarmfilterung,
- Entscheidungsunterstützung (decision support),
- Entwurf und
- Simulation.

Insbesondere spielen sie natürlich eine Rolle, je mehr ein bestimmtes Problem unscharf definiert ist und nicht mehr durch Differentialgleichungssysteme ausgedrückt werden kann.

Die Zahl der Publikationen auf diesem Gebiet in den letzten Jahren ist enorm, da man sich in allen ingenieurtechnischen Gebieten mit KI-Methoden befaßt. In diesem Abschnitt soll auf einige Beispiele eingegangen werden. Einen Überblick über KI-Methoden in der Prozeßtechnik kann man sich mittels [3.76,3.77] verschaffen. In [3.90] werden einige relevante Systeme wie RESCU, PICON und HEXSCON im Überblick beschrieben.

Automatisierungssysteme verfügen über eine große Anzahl lokal arbeitender Regelungs- und Steuerungskomponenten. Ein Nachteil der Lokalität liegt darin, daß die Gesamtstruktur bzw. die Parameter der Regelungssysteme nicht an globalen Veränderungen der Regelstrecke orientiert sind.

Andererseits ist es nur in wenigen Fällen möglich (oder es ist zu teuer), ein valides Gesamtmodell einer technischen Anlage anzugeben.

Übergeordnete Regelungsstrategien sollen solche Anpassungen vornehmen. Dies bedeutet z.B.

- die Modifikation von Regelungsparametern einzelner Regler durch die übergeordnete Strategie,
- die Einstellung von Arbeitspunkten und
- die Durchführung von Programmen wie z.B. Anfahren und Abfahren einer Anlage.

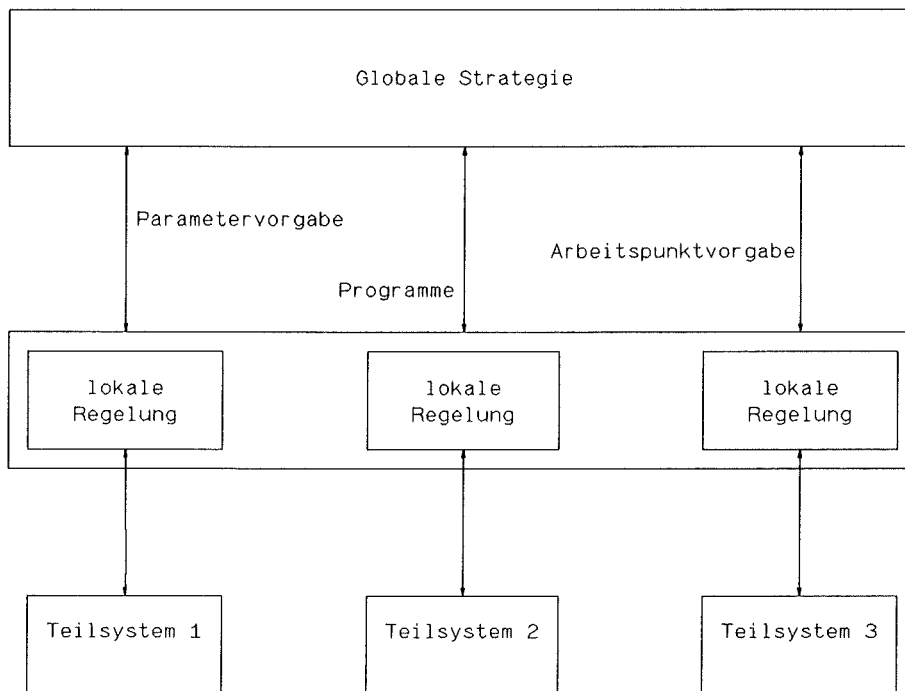


Abb. 3.4: Übergeordnete Regelungsstrategie

Da valide numerische Modelle der Gesamtanlage i.d.R. nicht verfügbar sind, ergibt sich hier eine Möglichkeit, mit KI-Methoden übergeordnete Strategien zu realisieren, z.B. durch ein Regelsystem, welches versucht, den globalen Systemzustand annäherungsweise zu beschreiben und globale Ziele zu verwirklichen. Abb. 3.4 zeigt den Einfluß der übergeordneten Strategie auf die lokalen Komponenten.

Eine Möglichkeit des Einsatzes von KI-Methoden in der Prozeßregelung liegt in dem Aufbau von lernenden, adaptiven Einzelreglern. In [3.91] wird eine solche Vorgehensweise zur Positionsregelung unter Verwendung der Geschwindigkeit als Meßgröße - ein typisches regelungstechnisches Problem - beschrieben. Ähnliche Beispiele zur Einstellung von Reglern finden sich z.B. in [3.92,3.93]

Ein komplexeres Problem ist die Überwachung und Regelung einer vollständigen Polymeranlage, wie sie in [3.94] dargestellt wird. Hier wird über den Zyklus

Diagnose - Fehlererkennung - Fehlerbehebung (recovery)

versucht, den Prozeß unter Verwendung eines Regelsystems in den gewünschten Arbeitspunkt zurückzuführen.

Eine ähnliche Vorgehensweise wird in [3.96] dargestellt. Hier geht es darum, hochwertige Strategien, die sonst von Operateuren durchgeführt werden, ebenfalls in Form von Regelsystemen aufzubauen.

Als abschließendes Beispiel für den Einsatz in der Regelung technischer Anlagen sei der Entwurf von Mehrgrößenregelungen genannt [3.97]. Hier wird durch die mathematische Beschreibung des Optimierungsproblem der Mehrgrößenregelung ein großer Suchraum für den optimalen Regler aufgespannt, der unter Verwendung numerischer Optimierungsverfahren Probleme bereitet. Der Einsatz von Regelsystemen zur Suche in diesem Problemraum ist hier eine realistische Alternative zur klassischen Vorgehensweise.

Als zweiten größeren Problemkreis, bei dem KI-Techniken eine Rolle spielen können, sei der der intelligenten Simulation genannt (Abb. 3.5). Sie kann folgenden Zielen dienen (siehe [3.76])

- der Simulation von Systemen, die unscharf definiert sind,
- der Simulation von Systemen, die über die Beschreibung durch Differentialgleichungssysteme hinaus auch unter Hinzuziehung von Regeln beschrieben werden können bzw. müssen,
- dem Training von Operateuren (Testfälle) und
- beim Entwurf von Regelungsstrategien.

Die in der Praxis auftretenden Systeme lassen sich wie schon gesagt nicht vollständig durch Differentialgleichungen beschreiben. Differentialgleichungssysteme verlieren gar oft ihre Gültigkeit, wenn eines oder mehrere Aggregate ausfallen, da sie solche Ereignisse i.d.R. nicht berücksichtigen. Daher wird die gemischte Simulation unter Hinzuziehung von KI-Techniken immer wichtiger.

Ein weiterer großer Teil der Untersuchungen widmet sich ganz allgemein (unter Setzung unterschiedlicher Schwerpunkte) der Fragestellung, wie man über die Schritte

- Diagnose des Prozeßzustands
- Alarm-Management und Alarmfilterung
- Entscheidungsunterstützung

den Operateuren beim Betrieb einer Anlage Hilfestellungen leisten kann. Wissensbasierte Techniken und entsprechende Systeme werden z.B. in [3.98-3.107] beschrieben. Andere Anwendungen wie z.B. Qualitätskontrolle [3.108] und scheduling [3.109,3.110] sind ebenfalls Felder für KI-Methoden. Als übergreifende Artikel sind [3.111-3.113] zu nennen.

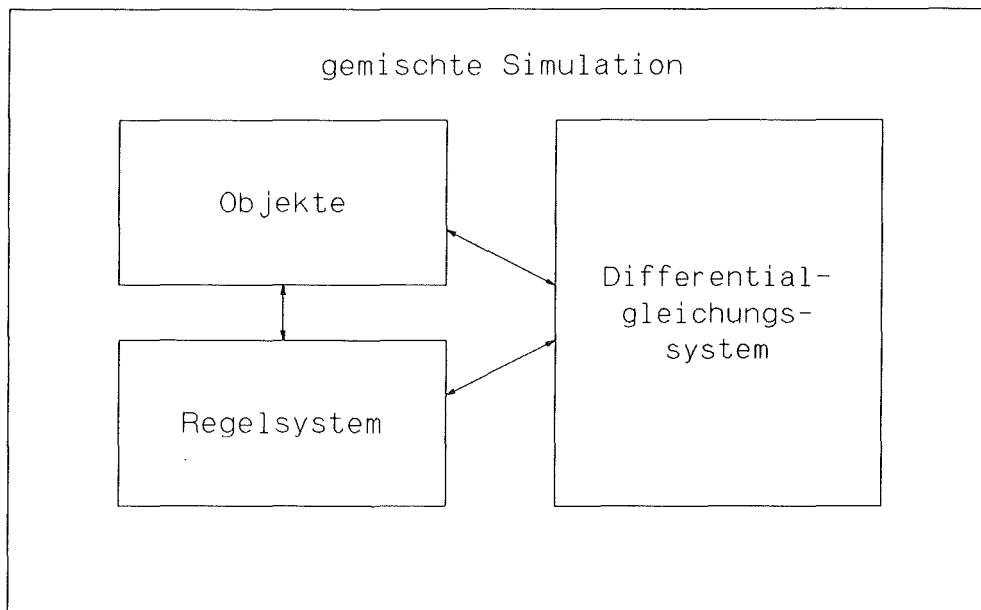


Abb. 3.5: Intelligente Simulation

Obwohl nach wie vor Operateure eine entscheidende Rolle bei der Führung komplexer technischer Anlagen spielen, sind auf dem Gebiet der Benutzerschnittstellen wenige Forschungsarbeiten und Projekte bekannt. Das einzige nennenswerte Projekt auf diesem Gebiet ist das GRADIENT-Projekt [3.114-3.116].

Es ist wahrhaft erstaunlich, wie viel in die Entwicklung von KI-Systemen zur Prozeßführung investiert wird und wie wenig die Benutzerschnittstelle berücksichtigt wird, obwohl diese doch einen entscheidenden Anteil bei globalen Entscheidungen besitzt, denn jeder Operateur kann nur aufgrund der ihm präsentierten Daten und Zustände Entscheidungen treffen. Aufgrund der Erfahrungen des in Kap. 2 erwähnten TAMARA-Projektes erschien gerade die Untersuchung intelligenter Benutzerschnittstellen für die Prozeßführung wichtig. Diese Erfahrungen gaben letztlich den Anstoß zu den vorliegenden Untersuchungen.

3.4.2 Wissensbasierte Methoden in der Umweltinformatik

In der Umweltinformatik werden wissensbasierte Methoden unter anderem zur Entscheidungsunterstützung eingesetzt. In [3.117] wird allerdings betont, daß viele der Projekte allerdings noch recht jung sind und die Risikobereitschaft zur Investition in diesem Gebiet noch recht beschränkt ist.

Als Anwendungsgebiete sind hier z.B. zu nennen:

- Altlasten
- Umweltverträglichkeitsprüfung
- Gefahrguttransporte
- Umweltanalyse

Wissensbasierte Methoden werden hier zur Programmierung von Expertensystemen verwendet. Diese zeichnen sich dadurch aus, daß unter Verwendung von Fakten und Regeln Schlußfolgerungsmechanismen zur Beantwortung der interessierenden Fragen durchgeführt werden. Die eingesetzte Softwaregrundlage reicht von Sprachen wie PROLOG bis zu Expertensystemshells. Einen Überblick gibt [3.119]. Es existieren wenige fortgeschrittene Projekte wie z.B. das XUMA-Projekt [3.118] zur Beurteilung von Altlasten.

Analog zum letzten Abschnitt ist auch in der Umweltinformatik zu erwarten, daß die Verbindung von symbolischen und numerischen Methoden bei der Simulation und Modellbildung von Umweltsystemen in der Zukunft verstärkt eine Rolle spielen wird, da hier die Modelle oft unscharf oder nicht vollständig definiert sind. Hierauf wird in [3.120] hingewiesen.

4. Eine Architektur für Benutzerschnittstellen komplexer Systeme

In diesem Kapitel wird ein Konzept vorgestellt, mit welchem der Entwurf von Endbenutzerschnittstellen in komplexen Systemen möglich wird. Im ersten Abschnitt wird ein Überblick gegeben, wie die Anforderungen, Ideen und Konzepte aus Kap. 2 letztlich in die Architektur eingeflossen sind. In den darauf folgenden Abschnitten werden dann die zwei Kernpunkte - das *Kommunikationskonzept* und die sog. *Integrierte Modellierung* - vorgestellt. Anschließend folgt eine Beschreibung eines implementierten Systems.

4.1 Überblick

Bei der Entwicklung der in diesem Kapitel vorzustellenden Architektur XAVIA (Expert Advanced Visualization Architecture) stand ursprünglich die Verwendung von Expertenwissen zur Optimierung von Dialogen im Vordergrund. Im Lauf der Entwicklung wurden XAVIA allerdings durch die Randbedingungen, die komplexe Systeme stellen, einige Eigenheiten eingeprägt, die darüber weit hinaus gehen. So lassen sich die Kernpunkte der Architektur heute durch folgende drei Aussagen beschreiben:

- Die Architektur dient dem Aufbau *kontext-adaptiver Dialogsysteme* unter Verwendung von Wissen über das komplexe System.
- Die zu visualisierenden Objekte werden auf einer hohen Ebene *integriert modelliert*. Das heißt:
 - Sie bestehen aus technischen und graphischen Eigenschaften (Attributen).
 - Sie besitzen als Methoden Regeln, um Wissen zu modellieren.
 - Sie können auch vollständige Dialogsequenzen in sich aufnehmen.
 - Sie werden in einer Objektdatenbank gespeichert und sind dadurch jederzeit *als Ganzes* in weiteren Anwendungen verwendbar.
- Ein Attribut (Instanzvariable) kann als *extern deklariert* werden. Dann ist die Quelle der Information zu diesem Attribut eine anderer Prozeß an einer definierten Stelle im Netzwerk.

Die Anwendbarkeit dieser drei Kernpunkte bedingt eine Reihe von sekundären Anforderungen, denen die Architektur insgesamt genügen muß. In diesem Abschnitt soll hierüber zunächst ein Überblick gegeben werden.

In Abb. 4.1 wird die Grundidee eingeführt. Man stelle sich drei Anwendungen vor, die der Führung eines technischen Prozesses dienen: eine Prozeßschnittstelle, ein

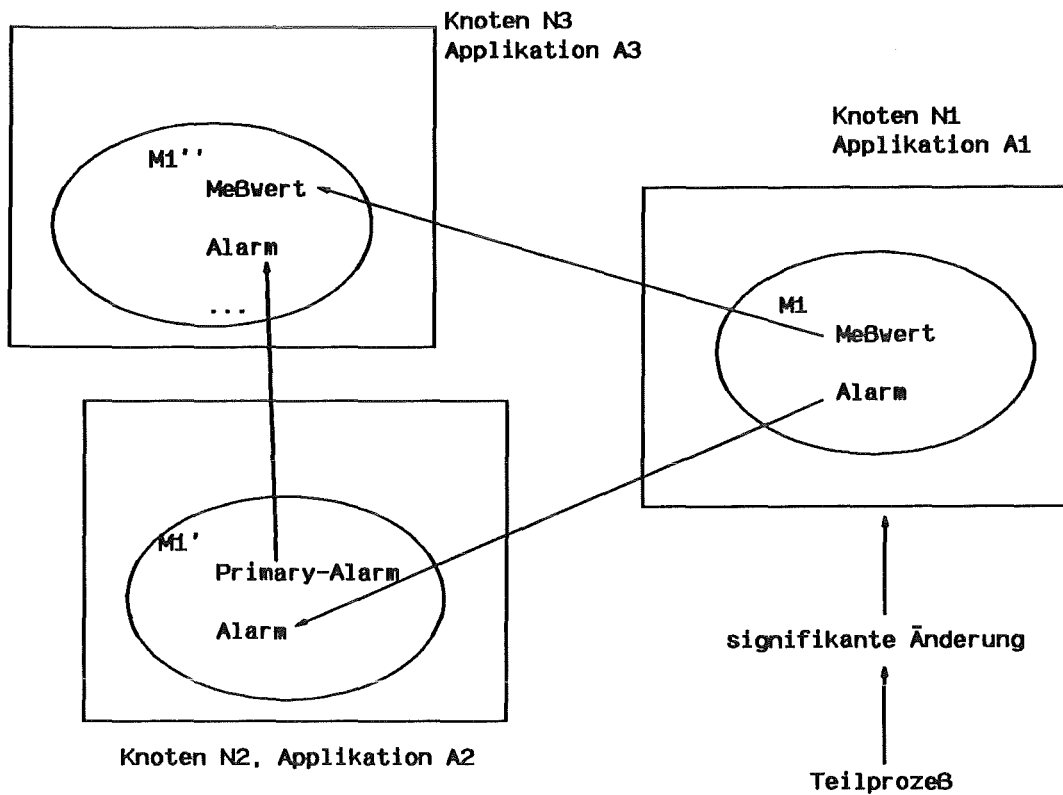


Abb. 4.1: Dezentralität und externe Attribute

Alarmfilter und eine Benutzerschnittstelle. Die Applikation Prozeßschnittstelle A1 auf Knoten N1 verfügt über ein Meßwert-Objekt M1, das wie folgt aufgebaut ist:

```

Object M1
[
  Attribute Meßwert
  Attribute Alarm
  Attribute ...
  Method ...
]

```

Der Wert von **Meßwert** wird durch eine lokale Methode oder Funktion in **A1** erfaßt, gespeichert und überwacht. Diese lokale Erfassung an der Prozeßschnittstelle kann zyklisch geschehen, allerdings ist diese unterste Erfassungsebene die einzige Stelle, an der eine zyklische Erfassung stattfindet. Die Überwachung generiert bei Überschreiten von Grenzwerten, bei Störungen in der Prozeßschnittstelle u.Ä. einen **Alarm**, der z.B. in einer textuellen Meldung oder auch in einer Variablen vom Aufzählungstyp kodiert sein kann. Es ist die Aufgabe der Prozeßschnittstelle, andere Teilnehmer im verteilten System über sie interessierende *signifikante Änderungen* des Wertes von **M1** zu unterrichten. Eine signifikante Änderung kann z.B. die Veränderung des Wertes von **Meßwert** um n Prozent sein, aber auch das Auftreten des Alarms.

Die Anwendung **A2** auf Knoten **N2** sei ein System zur Alarmfilterung. Sie mag z.B. über ein Regelsystem verfügen, welches Wissen über kausale und temporale Abhängigkeiten zwischen Alarmen dazu verwendet, ursächliche Störungen in der Anlage zu detektieren. **A2** ist also prinzipiell nur an Alarmen interessiert. In **A2** existiert daher ein Objekt **M1'**, welches die in **A2** benötigte Information über **M1** lokal hält:

Object **M1'**

[

Attribute **Alarm** - source (**N1**, **M1**, **Meßwert**, state)

Attribute **Primary-Alarm**

]

Da **A2** nur die Alarmwerte benötigt, wird auch nur der Alarm in dem Objekt **M1'** deklariert. In der Deklaration des Attributs wird direkt die Quelle der Information angegeben: die Information kommt vom Meßwertprozeß **A1** auf Knoten **N1** und wird dort im Objekt **M1** als Attribut **Meßwert** geführt. Das Attribut **Primary-Alarm** ist eine Information, die vom Alarmfilter selbst generiert wird. Es erhält einen bestimmten Wert, wenn der Alarmfilter aufgrund des gesamten Alarmzustands des Prozesses festgestellt hat, daß **Alarm** von primärer Bedeutung für den aktuellen Prozeßzustand ist. **Alarm** ist also der Rohalarm, **Primary-Alarm** ein gefilterter Alarm.

Anwendung **A3** auf Knoten **N3** schließlich sei das Visualisierungs- und Bediensystem der Anlage. Dort sei ein Objekt **M1''** zur Anzeige der Meßstelle definiert. Dieses Objekt beinhaltet nun nicht alleine den Meßwert, sondern beschreibt die gesamte Meßstelle inklusive der extern über sie verfügbaren Informationen, inklusive der vollständigen Benutzerdialoge und z.B. inklusive technischer Dokumentation (siehe auch Kap. 5: Filterungstechniken und Dialogmanagement).

Das Objekt **M1** mag also folgendermaßen aufgebaut sein:

Object **M1**

```
[  
    Attribute Meßwert - source (N1, M1, Meßwert, 10%)  
    Attribute Alarm - source (N2, M1', Primary-Alarm, state)  
    Attribute PushButton  
    Attribute PopUpMenu  
    Attribute technical-Document  
    Rule display-value ...  
    ...  
]
```

In **M1** wird die Meßstelle *integriert modelliert*. Das steht für zwei Dinge:

- Eine hochwertige Objektbeschreibung faßt sämtliche Informationen und Verhaltensweisen der Meßstelle in einem Aggregat zusammen. Dieses Aggregat wird als Einheit in einer Objektdatenbank gespeichert.
- Externe Informationen werden in die Objektbeschreibung integriert.

Das Grundkonzept der Informationsverarbeitung ist nun folgendes: wenn die drei Anwendungen unabhängig voneinander auf unterschiedlichen Knoten gestartet werden, sorgt das XAVIA-System selbständig dafür, daß benötigte physikalische Verbindungen über das Netzwerk aufgebaut werden. Beim Laden der Objekte in den Speicher wird festgestellt, ob externe Attribute existieren. Ist das der Fall, so wird an die betreffende Applikation ein *request* (eine Anfrage) um diese Information automatisch verschickt. Die Applikation antwortet damit, den derzeitigen Wert erstmalig zur Initialisierung zurück zu melden. Damit ist eine *virtuelle Verbindung* zwischen Attributen von Objekten hergestellt. Eine virtuelle Verbindung oder *virtual link* wird alleine durch die Externdeklaration automatisch aufgebaut (Abb. 4.2).

Nach Laden aller Objekte in allen Applikationen und nach Austausch aller virtuellen Verbindungen ist das Gesamtsystem initialisiert. Es sei darauf hingewiesen, daß kein irgendwie geartetes zentrales oder hierarchisches Konzept existiert, die Kom-

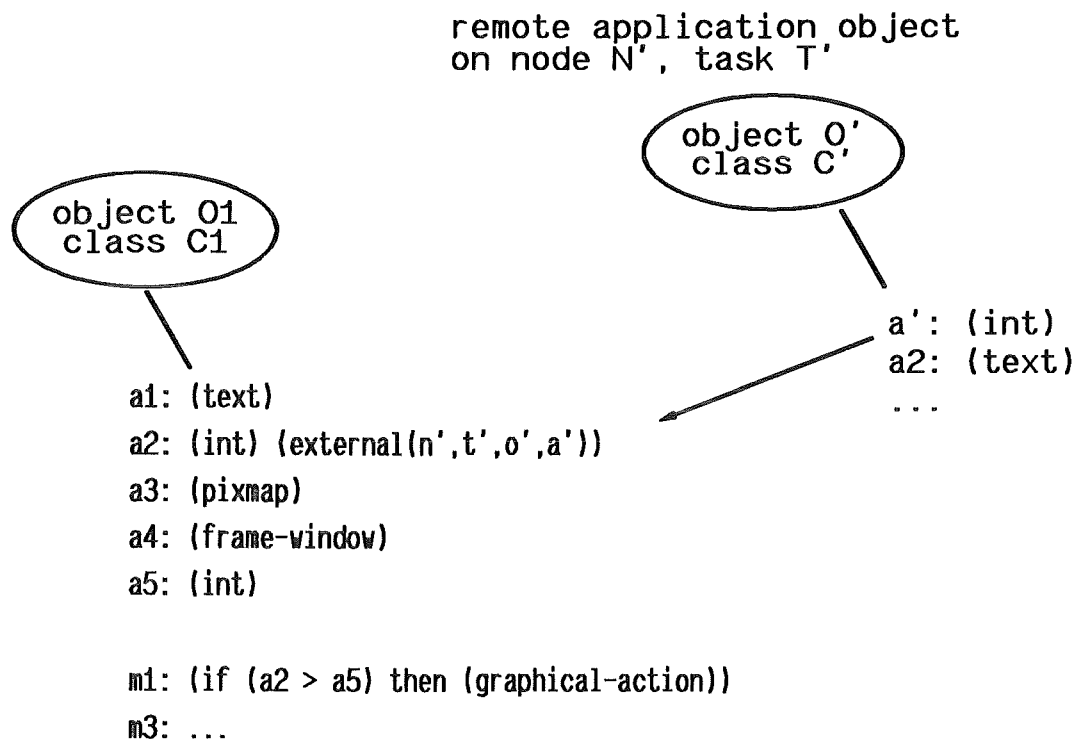


Abb. 4.2: Virtuelle Verbindung (entnommen aus [4.1])

munikation erfolgt kreuz und quer von jedem Teilnehmer zur jedem anderen Teilnehmer.

Im laufenden Betrieb wird sich nun der Wert von **Meßwert** in A1 mehr oder minder schnell ändern. Um zu vermeiden, den Wert bei jeder geringfügigen Änderung an jeden interessierten Teilnehmer zu verschicken, werden *Signifikanzkriterien* verwendet. Dies sind Bedingungen, die den Austausch von Informationen steuern. Sie bilden die Grundlage für das in Abschnitt 2.2 geforderte intelligente Informationsmanagement.

Im vorliegenden Beispiel werden zwei Kriterien verwendet:

- **n %**; d.h. Abweichung des Meßwertes um n Prozent gegenüber dem letzten an den Teilnehmer verschickten Wert
- **state**; d.h. eine beliebige Änderung des Zustands, z.B. Umschaltung von "normal" auf "fehlerhaft" o.Ä.

Wie man bei den Objektbeschreibungen sieht, wird in der Quelle für extern deklarierte Attribute das Signifikanzkriterium angegeben, d.h. die Nachfrage nach den Daten wird bezüglich einer bestimmten Eigenschaft durchgeführt. Dies entspricht der *Qualität eines Dienstes* gemäß Abschnitt 3.1.

Das Signifikanzkriterium ist lediglich ein Name für eine Entscheidung, ob eine Information neu zu verschicken ist. Über diese Signifikanzkriterien und deren Namen müssen im offenen System selbstverständlich Vereinbarungen getroffen werden.

Für jede virtuelle Verbindung muß also bei jeder Änderung des Ursprungswerts entschieden werden, ob er gemäß dem gewünschten Kriterium neu zu verschicken ist. Die Filterung der Information geschieht im Gegensatz zu anderen wie in [4.2] geschilderten Konzepten am *Ausgang* und nicht am Eingang eines Teilsystems.

Es ist das erklärte Ziel der Architektur, daß für den Datenaustausch aller Teilsysteme untereinander keine Programmierung mehr nötig ist. Der Datenaustausch wird selbständig durch ein *Kommunikationssystem* durchgeführt. Das auf dem Kommunikationssystem aufbauende *graphische Objektsystem* ist nach dem Konzept externer Attribute und dem Konzept der integrierten Modellierung aufgebaut.

Tab. 4.1, die aus [4.3] entnommen ist, zeigt die grundlegenden Eigenschaften der XAVIA-Architektur. Die Entwicklung der Architektur ist ausführlich in [4.3] bis [4.9] beschrieben. Zusammenfassend sollen die Kernpunkte der Architektur kurz aufgelistet werden:

- Beliebige Teilsysteme können miteinander und mit dem User Interface kommunizieren. Jedes Teilsystem kann Daten unterschiedlicher Typen, Qualität und Bedeutung erzeugen.
- Jedes Teilsystem ist ein Anbieter (server) für die Daten, die es erzeugt.
- Der Informationsaustausch erfolgt ohne Programmierung alleine durch Deklaration externer Attribute.
- Es werden nur Ereignisse ausgetauscht. Zyklen existieren nur auf der Meßwerterfassungsebene.
- Es findet vor Austausch der Information eine Filterung statt. Hierzu werden gemeinsame Signifikanzkriterien vereinbart.
- Der Informationsaustausch erfolgt spontan, also asynchron.
- Der Informationsaustausch erfolgt aufgrund der Funktionalität der Mechanismen der Task-zu-Task-Programmierung des OSI-Referenzmodells [4.10].
- Beliebige Programmiersprachen können in jedem Teilsystem verwendet werden.

Communication	Data / Knowledge	Interaction
<ul style="list-style-type: none"> ● decentralized, asynchronous message-passing 	<ul style="list-style-type: none"> ● objects and object classes 	<ul style="list-style-type: none"> ● priority-based objects
<ul style="list-style-type: none"> ● independant from networks and languages 	<ul style="list-style-type: none"> ● rules as methods 	<ul style="list-style-type: none"> ● significant attribute state changes trigger rules
<ul style="list-style-type: none"> ● automatic exchange of state changes for external attributes 	<ul style="list-style-type: none"> ● objects may have external attributes 	<ul style="list-style-type: none"> ● rules attached to objects perform interaction in the user interface
<ul style="list-style-type: none"> ● significance criteria 	<ul style="list-style-type: none"> ● external attributes are described not programmed 	<ul style="list-style-type: none"> ● rules describe the objects behaviour in a natural way
<ul style="list-style-type: none"> ● each application is a server for its own data 	<ul style="list-style-type: none"> ● integration of domain knowledge and graphical representation 	<ul style="list-style-type: none"> ● rules make the interface flexible and adaptive

Tab 4.1: Eigenschaften der Architektur (aus [4.3])

-
- Im User Interface System werden Objekte der natürlichen Welt integriert modelliert. Technische und graphische Eigenschaften bilden hochwertige, wieder verwendbare Einheiten.
 - Die Wissensrepräsentation basiert auf Objekten und Objektklassen. Objekte und Klassen besitzen als Methoden Regeln.
 - Objekte können externe Attribute besitzen.
 - Die Objekte sowie der gesamte Informationsaustausch sind mit Prioritäten versehen (diese Eigenschaft wurde nur für technische Prozesse eingeführt).
 - Regeln beschreiben die Interaktion in der Benutzerschnittstelle auf natürliche Art und Weise. Sie machen die Benutzerschnittstelle flexibel und adaptiv (siehe Kap. 5).

In Tab. 4.1 sind wie früher geschehen [4.3] die Eigenschaften des Kommunikationsmodells, des Wissensmodells und des Verarbeitungs- bzw. Interaktionsmodells von

XAVIA angegeben. Es ist allerdings nur schwer möglich ist, diese drei Konzepte voneinander zu trennen. Jedes Konzept wirkt in jedes andere hinein. Daher soll die folgende Darstellung anders als in [4.3] erfolgen.

Zunächst wird das Kommunikationsmodell im Überblick erläutert. Dieses läßt sich noch am ehesten von den beiden anderen Modellen trennen, wenn es auch bis in das Objektsystem des Dialogsystems von XAVIA hinein wirkt. Im darauf folgenden Abschnitt wird die integrierte Modellierung beschrieben, d.h. hier wird eigentlich das Objektmodell der Visualisierungskomponente aufgezeigt. Beides wird anhand der Implementierung des XAVIA-Systems vertieft.

4.2 Kommunikationsmodell

Das Kommunikationsmodell dient dem Zweck, den Austausch von Information zwischen Anwendungen so zu gestalten, daß der Aufwand

- bei der Konstruktion des Systems und
- bei der Informationsübertragung zur Laufzeit

möglichst gering gehalten wird. Um das Gesamtsystem möglichst offen zu gestalten, sollen die Grundannahmen möglichst gering sein, in beliebigen Sprachen implementierbar sein und sich beim Austausch streng an die Task-zu-Task-Kommunikation des OSI-Referenz-Modells halten [4.10]. Das Kommunikationsmodell basiert auf folgenden Elementen:

- auf *externen Attributen* (Instanzvariablen) von *Objekten*,
- auf *requests* (Anfragen),
- auf *significances* (Signifikanzkriterien, Filter),
- auf *messages* (Meldungen über neuen Wert eines externen Attributs) und
- auf *updates* (Aktualisierungen) eines externen Attributs.

Für das Kommunikationsmodell ist es dabei vollkommen nebensächlich, in welcher Sprache einer der Teilnehmer implementiert ist und ob es sich um eine objekt-orientiert Sprache oder eine prozedurale Sprache handelt. Die Verknüpfung

Objekt - Attribut

wird lediglich als ein beliebiger Ort einer Information betrachtet und zeichnet sich durch den Objektnamen, den Namen des Attributs, den Datentyp des Attributs und dessen Wert aus.

Ein *externes Attribut* ist ein Attribut, dessen Wert nicht in der lokalen Anwendung generiert und verändert wird, sondern von einer anderen Anwendung auf dem Netzwerk erzeugt und aktualisiert wird. Ein externes Attribut ist dadurch gekennzeichnet, daß es ein sog. *request element* besitzt (Abb. 4.3).

req =	(apps	source application
		objs	source object
		attrs	source attribute
		types	type of source attribute
		signs	significance criteria
)		

Abb. 4.3: Request element

Dabei ist **apps** der Netzwerkprozeß, der das Quellobjekt **objs** mit dessen Attribut **attrs** vom Typ **types** führt. Der Name eines Signifikanzkriteriums, das in **apps** definiert sein muß, wird mit **signs** angegeben.

Es sei ausdrücklich darauf hingewiesen, daß die Namen **objs** und **attrs** lediglich Synonyme für irgendeine Information darstellen. Jeder Teilnehmer kann eine entsprechende Information so speichern wie es ihm beliebt. Es ist auch kein objekt-orientiertes System dafür notwendig, an der Kommunikation teilzunehmen. Es müssen lediglich entsprechende Synonyme für gewünschte Informationen zur Verfügung gestellt werden. Die Quellapplikation **apps** wird durch zwei Parameter definiert, die gemäß OSI-Referenzmodell der Knotenname **nodes** und der Netzwerk-Taskname (service name) **tasks** sind.

apps =	(nodes	network node
		tasks	network task name
)		

Abb. 4.4: Application

Der Knotenname ist ein Name in der Notation des entsprechenden Netzwerktyps (DECnet, IEEE), der Task-Name ein Synonym für einen Prozeß, der im Sinne von OSI einen sogenannten *transport service access point* darstellt, also einen Dienst.

Ein Attribut $attr_d$ eines Objektes obj_d in einer Applikation app_d wird dadurch zu einem externen Attribut, daß für das Attribut $attr_d$ ein request element

$$req (apps , obj_s , attr_s , type_s , sign_s)$$

definiert wird (Abb. 4.5). Der Index d steht für *destination*.

$$obj_d = (\begin{array}{l} attr_1 \\ attr_2 \\ \dots \\ attr_d \text{ (valued) - } req (apps , obj_s , attr_s , type_s , sign_s) \\ \dots \\ attr_n \end{array})$$

Abb. 4.5: Objekt mit externem Attribut

Damit eine virtuelle Verbindung erfolgreich aufgebaut werden kann, muß natürlich in der Quellapplikation app_s ein entsprechendes Objekt existieren (Abb. 4.6).

$$obj_s = (\begin{array}{l} attr_{s1} \\ attr_{s2} \\ \dots \\ attr_s \text{ (value}_s \text{)} \\ \dots \\ attr_{sm} \end{array})$$

Abb. 4.6 : Quellobjekt

Abb. 4.7 zeigt nun den genauen Ablauf beim Aufbau eines *virtual link*. Wird ein externes Attribut in eine Anwendung geladen (d.h. für sie existiert ein *request element*), so wird eine *request message* an die Quellapplikation verschickt.

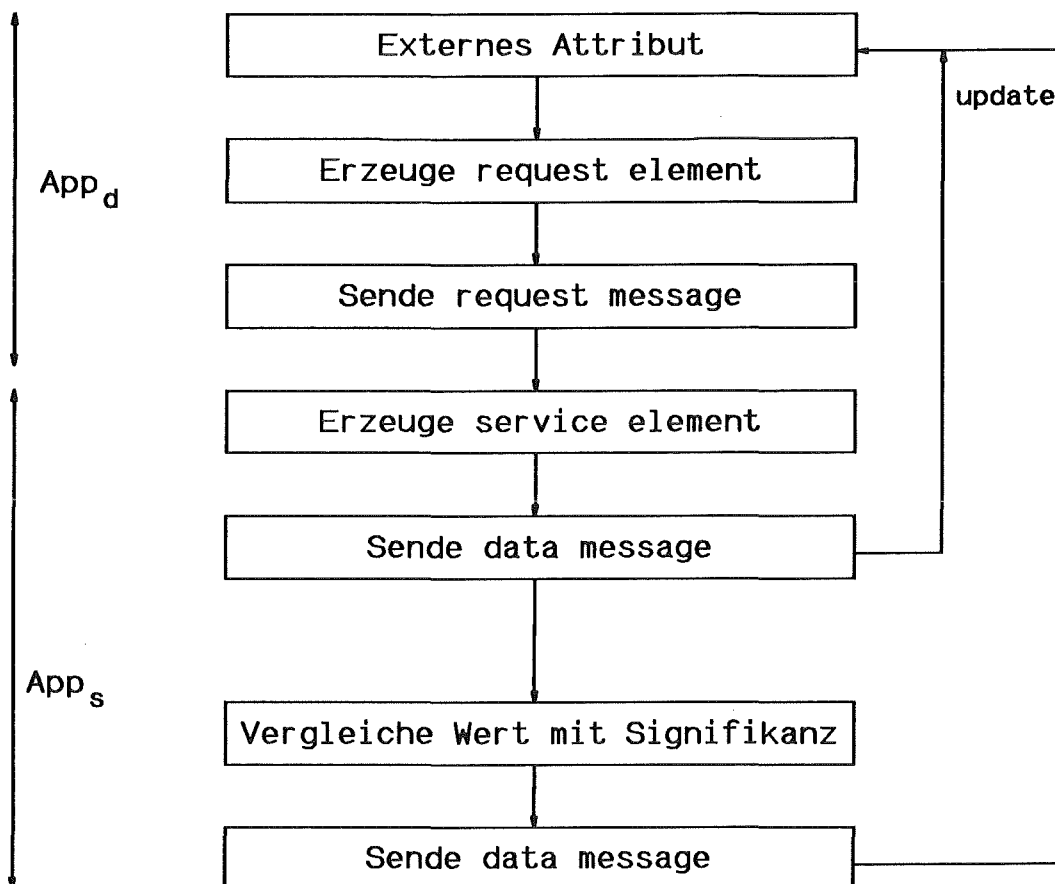


Abb. 4.7: Ablauf beim virtual link

Diese *request message req-msg* beinhaltet alle die Informationen, die dazu nötig sind, sowohl den Informationskonsumenten **app_d** als auch den Informationsproduzenten **app_s** eindeutig zu identifizieren. Außerdem beinhaltet sie noch den Namen des Signifikanzkriteriums.

Kommt die **req-msg** beim Prozeß **app_s** an, so wird dort ein sogenanntes *service element serv* für die Anfrage erzeugt. Anschließend wird erstmalig der Wert **value_s** des Quellattributs **attr_s** mittels einer *data message data-msg* an die Zielapplikation **app_d** geschickt, die dort zu einem *update*, also einer Initialisierung des Wertes **value_d** von **attr_d** führt. Abb. 4.8 zeigt den Aufbau der *request message req-msg*, Abb. 4.9 das

beim Quellprozeß erzeugte *service element* **serv**, Abb. 4.10 schließlich die *data message* **data-msg**.

```
req-msg = ( to: apps
            objs
            attr_s
            from: appd
            objd
            attrd
            type: type_s
            significance: sign_s )
```

Abb. 4.8: Request message

```
serv = ( from: objs
         attr_s
         to: appd
         objd
         attrd
         significance: sign_s
         last value sent: value )
```

Abb. 4.9: Service element

```

data-msg = (
    to:          appd
                objd
                attrd
    type:        types
    value:       new-value )

```

Abb. 4.10: Data message

Nachdem der Wert des externen Attributes erstmals ausgetauscht wurde, speichert die Applikation **apps** den letzten verschickten Wert im *service element serv*. Anhand dieses Wertes wird anschließend durch das gewünschte Signifikanzkriterium jeweils entschieden, ob eine neue *data message* zu verschicken ist. Damit ist die virtuelle Verbindung zustande gekommen.

Natürlich bedingt dieses Konzept zunächst den Aufbau physikalischer Verbindungen, Fehlerbehandlung, u.s.w. Die gesamte Kommunikation muß *asynchron* arbeiten, da nie von vorne herein klar ist, wann sich eine Information signifikant ändern wird. Auf implementierungstechnische Details all dieser Probleme soll hier nicht eingegangen werden.

Wirklich wichtig ist der eigentlich Grundgedanke hinter dem Modell:

- Eine Anwendung kann eine Information als extern deklarieren.
- Sie definiert, für welche signifikanten Änderungen sie sich interessiert.
- Die Anwendung, die die Information besitzt, verschickt dieses Datum bei signifikanter Änderung.
- Die Informationsfilterung geschieht vor dem Verschicken.
- Der gesamte Austausch bis hin zur Aktualisierung des lokalen Wertes erfolgt asynchron und vollständig automatisch.

Eine wesentliche Anforderung an XAVIA war die Offenheit des Systems. Daher müssen Mechanismen zur Verfügung gestellt werden, die die Einbindung beliebiger Programmiersprachen ermöglichen. Hierzu wurde schon früh eine Schichtung des Systems gemäß Abb 4.11 vorgeschlagen [4.11].

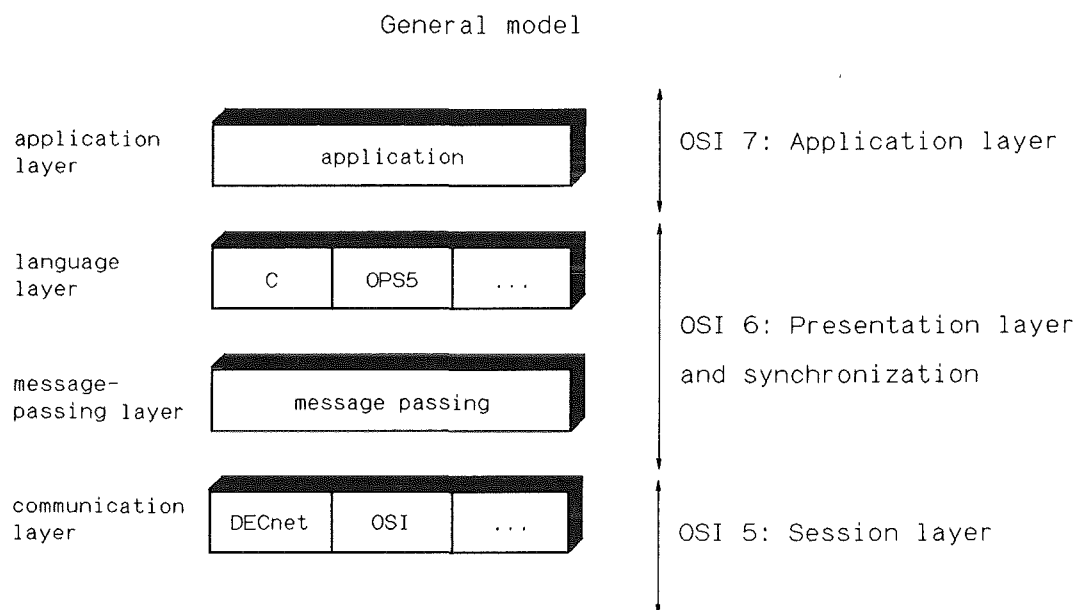


Abb. 4.11: Schichtung des Kommunikationssystems

Über das *communication layer* werden im Sinne der OSI-Schicht 5 Nachrichten ausgetauscht. Jede Anwendung, die im offenen System teilnehmen möchte, muß in dieser Art kommunizieren und sich als Dienst (server) für ihre Daten zur Verfügung stellen.

Im *message-passing layer* werden physikalische Verbindungen verwaltet, Meldungen verschickt, asynchron empfangen und zwischen gespeichert. Hier und im *language layer* werden die Meldungen von bzw. in sprach-spezifische Darstellungsarten umgewandelt. Dies entspricht der Schicht 6 von OSI. Eine weitere nicht triviale Aufgabe dieser beiden Schichten ist die der Synchronisation. Da die Meldungen asynchron eingeht, ist eine Synchronisation über die Sprachschicht hin zur Applikation nötig.

Das vorliegende Modell wurde im Laufe des Jahres 1990 entwickelt [4.5] und implementiert [4.7,4.8]. Es wurde zum Teil in C und zum Teil in OPS5 geschrieben. Das

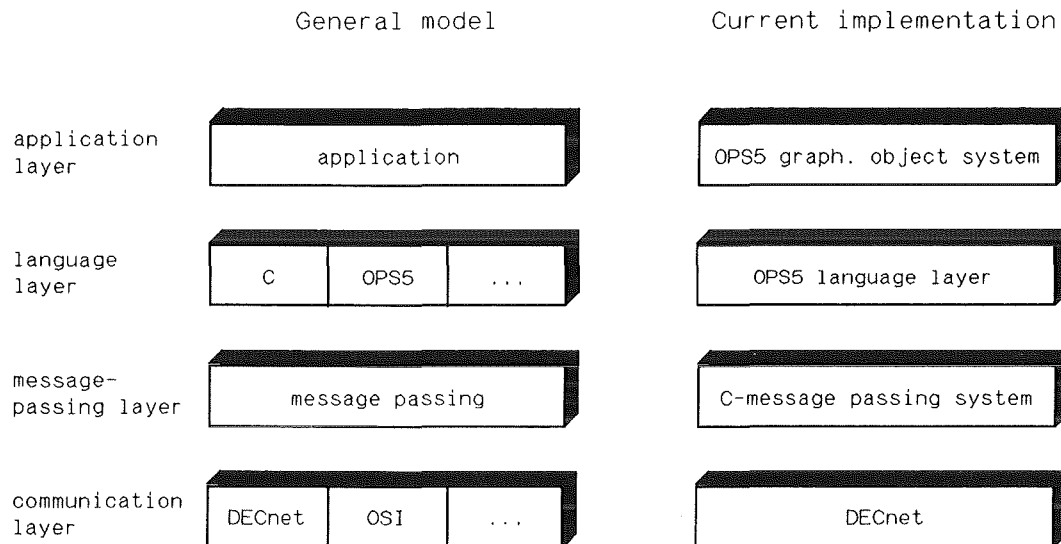


Abb. 4.12: Aktuelle Implementierung von XCS

message passing system XCS (XAVIA Communication System) ist vollständig in C implementiert und verwendet die Kommunikationsmechanismen von DECnet. Die Objektbeschreibung hingegen wurde in OPS5 realisiert. Dort lassen sich die gesamten zur Kommunikation benötigten Elemente und Funktionen recht einfach mittels Regeln beschreiben. Die aktuelle Implementierung von XCS ist in Abb. 4.12 dargestellt.

4.3 Integrierte Modellierung

In diesem Abschnitt wird näher auf die Gedanken eingegangen, welche der *integrierten Modellierung* zugrunde liegen. Wie schon in 4.1 angeführt geht es dabei um zwei Dinge:

- Als Objekte werden hoch-aggregierte Mengen von Informationen angesehen, die inklusive ihrer Verhaltensweisen in einer Objekt-Datenbank gespeichert werden und wieder verwendbar sind.

- Objekte können externe Attribute besitzen. Die Integration von externer Information in das User Interface System geschieht durch Deklaration, die Aktualisierung automatisch anhand der in 4.2 geschilderten Konzepte.

Als Objekt wird z.B. eine Pumpe einer technischen Anlage angesehen. Sie besitzt eine Reihe von Eigenschaften: Zustände des Aggregats, Stör- und Alarmmeldungen, Zustände, Nummern und Kanäle der Ein- /Ausgabebaugruppe, Zustand der Automatisierunginsel der Baugruppe, Zustand des Bussystems, technische Dokumentation in Form von gescannten Plänen, Interaktion mit dem Benutzer (Menüs, Fenster für Pläne, komplette Dialoge), Werte aus Diagnosesystemen, u.s.w.

Eine Reihe von Eigenschaften der Pumpe aus der Sicht des User Interface Systems sind von externen Natur: sie residieren in externen, über das Netzwerk verstreuten Prozessen, z.B. in der Prozeßschnittstelle oder in einem Diagnosesystem.

Die *integrierte Modellierung* betrachtet also Objekte der natürlichen Welt über ihre DV-technische Umsetzung hinweg bis zu ihrer Repräsentation auf dem Bildschirm *als Ganzes*.

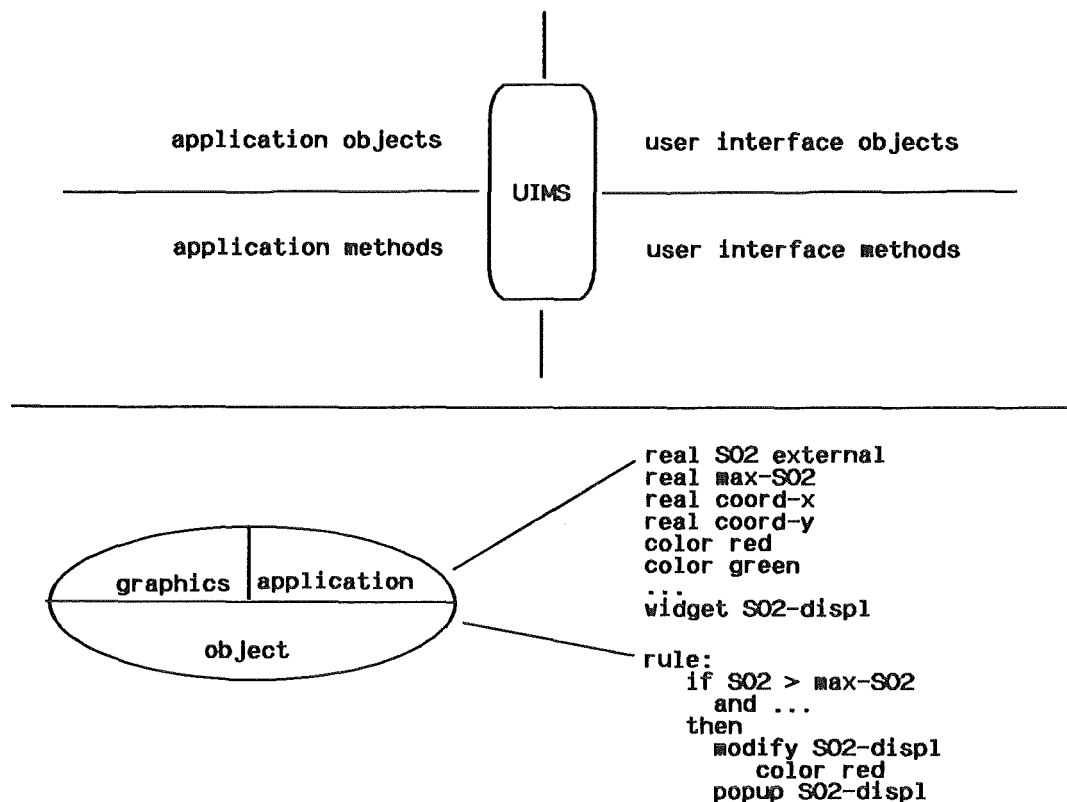


Abb. 4.13: Integrierte Modellierung

Wie die Speichertechnik und wie die Funktionalität umgesetzt wird, wird zunächst von diesem Prinzip nicht vorgeschrieben. Im vorliegenden Fall, in dem es darum ging, Dialoge adaptiv zu gestalten und Filterungsmechanismen für komplexe Informationsmengen bereit zu stellen, wurde die Modellierung der Objekte mit objekt-orientierten Konzepten durchgeführt. Die den Instanzen und Klassen zugeordneten Methoden sollten allerdings Regeln sein, die von einem Regelinterpreter abgearbeitet werden. Davon versprach man sich besondere Vorteile bei der Modellierung des natürlichen Systems und der Dialoge.

Das Verhältnis der *integrierten Modellierung* zu User Interface Management Systemen (UIMS) ist zwiespaltig: einerseits wird die Trennung zwischen Applikation und Repräsentation, die in UIMS vorgenommen wird, hier nicht vollzogen. Die Objektbeschreibung im Dialogsystem faßt das Objekt in seiner Applikationskomponente und in seiner Repräsentationskomponente als Ganzes auf. Andererseits werden Teile des Objektes in externen Prozessen geführt, was wiederum der Trennung bei UIMS analog ist.

Ein Hauptunterschied ist allerdings, daß bei der *integrierten Modellierung* im Prinzip gar kein UIMS existiert. Es existieren im Objektsystem graphische Funktionen, die von Objekten bzw. deren Regeln benutzt werden. Das Dialogsystem an sich besteht nur aus diesen Objekten und Regeln und einem eingebundenen verteilten, graphischen Laufzeitsystem. Das UIMS, welches ansonsten die Implementierung beherrscht, *ist die Wissensbasis selbst*.

Welche Konsequenzen diese Tatsache gegenüber UIMS besitzt, läßt sich derzeit noch schwer beantworten. Die Erfahrungen beim Umgang mit diesem Konzept waren aber, was die Erweiterbarkeit anbelangt, positiv: benötigt man ein neues Konzept in einer bestimmten Benutzerschnittstelle, so modelliert man einfach die entsprechenden Objektklassen und Objekte.

4.4 Implementierung

Die XAVIA-Architektur wurde 1990/1991 weitestgehend implementiert. Als Basis für die Kommunikation wurde das System XCS (**X**AVIA **C**ommunication **S**ystem) in C implementiert. Dieses System bietet sämtliche benötigten Kommunikationsmechanismen [4.7]. Aufbauend auf XCS wurde eine Sprachschicht für OPS5 realisiert.

Für ein verteiltes, graphisches Objektsystem zur Visualisierung mußte eine Entwurfsentscheidung getroffen werden. Es war das Hauptanliegen, objekt-orientierte Konzepte (Klassen, Instanzen, Vererbung,...) mit Regeln als Methoden von Klassen und Instanzen zu verbinden. Rein objekt-orientierte Sprachen bieten keine Regelverarbeitung, KI-Sprachen wie PROLOG und OPS5 keine objekt-orientierten Konzepte. Der Einsatz von KI-shells erschien für den nahen Echtzeitbereich und bei Randbedingungen wie Asynchronität, Verteiltheit und Effizienz nicht als sinnvoll.

Nach Abwägen von Aufwand, Vor- und Nachteilen wurde schließlich der Entschluß gefaßt, das Objektsystem in OPS5 selbst zu entwerfen. Die Tatsache, daß das verfügbare OPS5-System über Synchronisationsmechanismen verfügte, spielte bei dieser Entscheidung eine nicht unwesentliche Rolle.

Nach einigen Versuchen erschien es auch einfacher, in OPS5 ein Objektsystem zu implementieren als z.B. in C++ einen Regelinterpreter, der die Mächtigkeit eines OPS5-Interpreters besitzt. Da der Schwerpunkt auf der Modellierung von Prozeß und Dialog lag (vor allem versprach man sich von den Regeln Vorteile bei den adaptiven Dialogen), wurde die Entscheidung getroffen, die erste Vorgehensweise vorzuziehen.

Eine weitere Entwurfsentscheidung für das graphische Objektsystem war folgende: die selbst implementierte Vererbung sollte nur in einem Entwurfssystem erfolgen, sozusagen generisch. Vererbung zur Laufzeit sollte aus Effizienzgründen vermieden werden. Daher wurde das XAVIA-System in zwei Teile getrennt (Abb. 4.14):

- in ein Entwurfs-Objektsystem
- und in ein Laufzeit-Objektsystem.

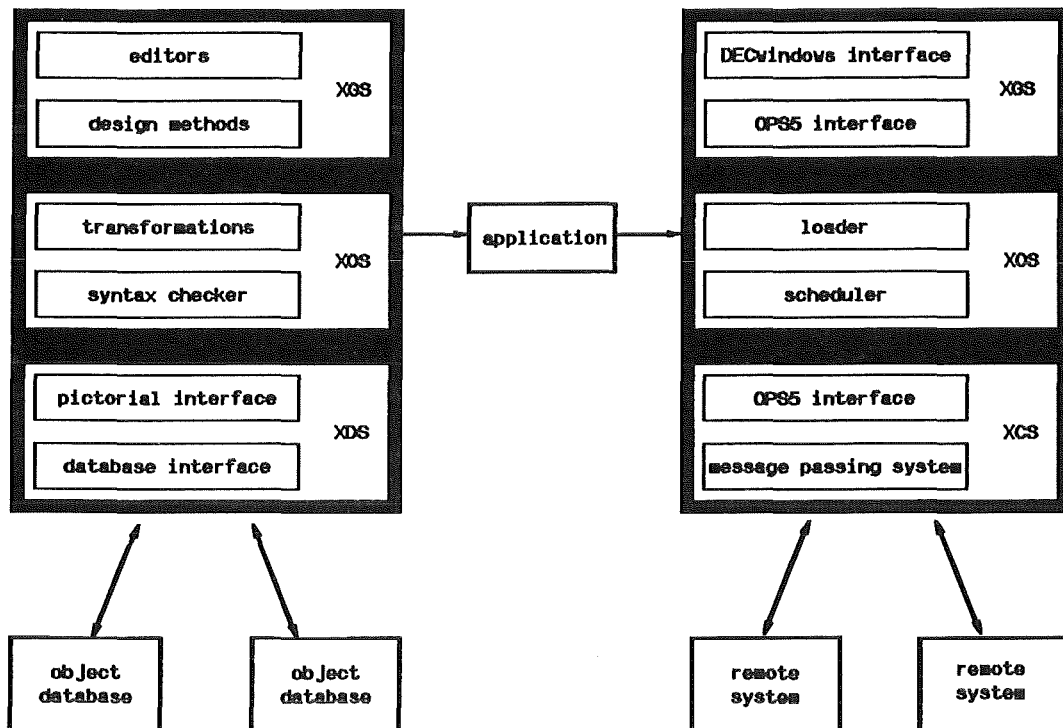


Abb. 4.14: Überblick über das XAVIA-System

Die im Entwurfssystem modellierte Vererbung wird durch eine Transformation in das Laufzeitsystem umgesetzt. Im Laufzeitsystem haben zwar alle Objekte noch Klassen. Regeln, die über Klassen hinweg vererbt werden, werden durch diese Transformation derart verändert, daß der in ihnen aufgeführte Klassenname mit allen zu vererbenden Klassennamen der Kinder einem logischen *oder* unterworfen wird. Dadurch triggert eine solche Regel auf mehrere Klassen und nicht auf eine einzige und die Vererbung wird ohne aufwendiges objekt-orientiertes Laufzeitsystem möglich.

Das XAVIA-System besteht aus vier Grundmoduln:

- XCS - XAVIA Communication System,
- XOS - XAVIA Object System,
- XGS - XAVIA Graphics System und
- XDS - XAVIA Database System.

Das XCS-System besteht wie schon erwähnt aus dem message passing und einer OPS5-Schnittstelle. Das Objektsystem besteht aus zwei Teilen,

- einem Entwurfsteil von XOS und
- einem Laufzeitteil von XOS.

Der Entwurfsteil beinhaltet Zugriffsfunktionen auf die Objekt-Datenbank und einen Syntaxchecker, der derzeit (wie das gesamte Entwicklungssystem) noch in Entwicklung ist.

Das Graphiksystem XGS ist das an der AG Computergraphik der Universität Kaiserslautern entwickelte KGT-System (Kaiserslautern Graphics Toolkit), welches auf DECwindows basiert [4.6,4.9]. Hiervon wird im Laufzeit-Objektsystem das OPS5-Interface zu DECwindows verwendet. Im Entwicklungssystem wird das C-Interface zu KGT verwendet, um graphische Objekte edieren zu können.

Das Objekt-Datenbank-Interface XDS dient der Erzeugung, Speicherung und Wiederverwendung von Objektklassen und Objekten in einzelnen Anwendungen (database interface). Zusätzlich gibt es ein Interface zur Bearbeitung, Speicherung und Verwendung von Pixel-Informationen wie z.B. technische Dokumentation oder topologische Karten (pictorial interface).

Anwendungen, die mit dem XAVIA-System erzeugt werden, werden im Entwicklungssystem aufgebaut. Hierzu werden Hierarchie-Browser und Editoren verwendet. Um eine Anwendung zu testen oder auszuführen, wird aus der Applikations-Datenbank mittels der genannten Transformation ein sog. Applikationsfile erzeugt, derzeit eine sequentielle Datei, es ist aber daran gedacht, in Zukunft direkt aus der Applikationsdatenbank heraus eine Anwendung zu laden und auszuführen.

Dieses Applikationsfile wird vom Laufzeitsystem über einen Lader geladen. Der Lader liest alle Objekte und Regeln ein, erzeugt daraus dynamisch OPS5-Objekte und OPS5-Regeln, und startet anschließend den Regel-Scheduler. Die Regeln werden auf der Priorität ausgeführt, die ein Objekt besitzt, für das die Regel feuerbereit ist. Die Priorität ist also an Objekte gebunden und nicht an einzelne Regeln.

Das Laufzeit-Objektsystem schließlich nimmt zu allen externen Applikationen Verbindung auf und sorgt für das ordnungsgemäße Versenden von *request-messages* und *data-messages*. Die graphischen Teile der Objekte werden ebenso initialisiert und schließlich wird das Wurzel-Fenster der Anwendung dargestellt. Sämtliche Interaktion mit dem Benutzer und sämtliche Veränderungen in den Objekten werden dann vom Regel-Scheduler ausgeführt.

5. Filterungstechniken und Dialoggestaltung

In diesem Kapitel werden Konzepte zur Optimierung der graphischen Benutzerschnittstelle anhand von Prozeßwissen vorgestellt. Die gezeigten Beispiele wurden mit dem XAVIA-System implementiert. Dabei wurden reale Prozeßgrößen, also Meßwerte, Zustände und Ähnliches, in Simulationsprozessen bereitgestellt.

Der Augenmerk soll auf Filterungstechniken für komplexe Informationsmengen und auf die Gestaltung kontext-adaptiver Dialoge gerichtet werden. Es wird auch gezeigt, daß aufgrund des verteilten Ansatzes Teile der Benutzerschnittstelle in unterschiedlichen Prozessen untergebracht werden können. Dadurch kann ein zu großes Anwachsen einer einzelnen Wissensbasis verhindert werden, was durch den Vorgang der conflict set resolution zu erheblichen Nachteilen im Laufzeitverhalten führen kann.

5.1 Eine Benutzerschnittstelle für eine verfahrenstechnische Anlage

Die dargestellte Architektur wurde in einem Kooperationsprojekt der AG Computergraphik der Universität Kaiserslautern mit dem Institut für Datenverarbeitung in der Technik (IDT) des Kernforschungszentrums Karlsruhe (KfK) implementiert. Das Forschungsvorhaben wurde im Rahmen des TAMARA/TPAS-Projektes der KfK durchgeführt [5.1 bis 5.6].

TAMARA (Testanlage zur Müllverbrennung, Abgasreinigung, Rückstandsverwertung, Abwasserbehandlung) ist eine Forschungs- und Versuchsanlage zur Untersuchung der bei der Müllverbrennung vorkommenden verfahrenstechnischen und chemischen Vorgänge und wird seit 1986 in mehreren Versuchskampagnen pro Jahr betrieben. TPAS (TAMARA-Prozeßautomatisierungssystem) ist das vom IDT entwickelte und implementierte Automatisierungssystem für TAMARA. Das Gesamtvorhaben wird im Rahmen des Projektes Sicherheit und Umwelt (PSU) der KfK durchgeführt. Die Entwicklung und Umsetzung der XAVIA-Architektur erfolgte - bis zum heutigen Stand - von der Mitte des Jahres 1989 an.

In diesem Kapitel sollen drei Aspekte hervorgehoben werden. Der erste Aspekt ist die Adaption des Dialogs an den Prozeßzustand. Dies wird anhand des ersten implementierten Beispiels, der Speisewasserversorgung der Anlage, verdeutlicht.

Zum zweiten soll auf die Herausforderungen beim Aufbau einer vollständigen Benutzerschnittstelle für den Feuerungsteil von TAMARA eingegangen werden. Die Menge an Information, die hier darzustellen ist, ist derart groß, daß es schon schwierig wird, sie in ihrer Gesamtheit auf einem Bildschirm *sinnvoll* unterzubringen. An dieser Stelle soll auch auf die Möglichkeiten unterschiedlicher *views* (Ansichten) eingegangen werden.

Schließlich werden einige Aspekte beim objekt-orientierten Aufbau der Graphik betrachtet. Je nach Aufbau der Hierarchie der Objektklassen ist es möglich, bestimm-

te Aufgaben der Benutzerschnittstelle in sehr wenigen Attributen und Regeln für alle oder für eine große Menge von Anlagenobjekten auszudrücken. Hier mag einer der größten Vorteile des objekt-orientierten Ansatzes liegen. Die aufgezeigten Beispiele geben eine Idee von der Mächtigkeit des Konzeptes.

5.1.1 Prozeß-Adaptiver Dialog

Die Adaption der Benutzerschnittstelle wurde anhand zweier Konzepte untersucht. Das erste Konzept modifiziert die optische Darstellungsweise von Objekten. Dies dient der Übersichtlichkeit eines Prozeßbildes insbesondere dann, wenn eine Vielzahl von Störungen auftreten. Dieses Konzept soll *optische Filterung* genannt werden. Das zweite Konzept modifiziert die Dialoge mit Objekten zur Vermeidung unnötiger oder unmöglicher Interaktion. Dies wird in der Folge als *Dialogfilterung* bezeichnet. Dialogfilterung dient der Beschleunigung des Dialogs und der schnellen Auffindbarkeit wichtiger Information. Beide Konzepte bedienen sich des Prozeßwissens und werden in der Hauptsache durch Regeln realisiert.

Um die folgenden Ausführungen verständlicher zu machen, werden nochmals die Begriffe *Primärstörung*, *Sekundärstörung* und *Verriegelung* erläutert.

Unter einer Primärstörung wird eine Störung oder ein Alarm verstanden, der bei einem Objekt **O1** festgestellt wurde und für dessen Ursache nur ein Grund innerhalb des Objektes **O1** beobachtet werden kann. Dies ist z.B. dann der Fall, wenn ein Aggregat überlastet ist und dessen Sicherung fällt. Ein weiterer häufig vorkommender Fall ist z.B. die Überschreitung eines Temperaturgrenzwertes, die einen Alarm auslöst. Oft ist es in solchen Fällen nur schwer möglich anzugeben, warum dieser Grenzwert überschritten ist, da es an validen Modellen für analoge Größen fehlt. (Steuerungstechnische Zusammenhänge sind i.d.R. einfacher zu verwenden und eher bekannt als chemisch-physikalische.)

Unter einer Sekundärstörung wird eine Störung oder ein Alarm eines Objektes **O1** verstanden, für die eine Ursache bei einem anderen Objekt **O2** festgestellt werden kann. Dies kann z.B. der Fall sein, wenn ein Aggregat deswegen notabgeschaltet wird, weil eine andere Prozeßgröße (ein Meßwert oder ein Aggregat) einen kritischen Zustand eingenommen hat. Sekundärstörungen sind in der Regel aufgrund des steuerungstechnischen Wissens zur Laufzeit bekannt.

Unter einer Verriegelung wird ein Zustand eines Objektes **O1** verstanden, in dem das Objekt nicht seine volle Funktionalität erfüllen kann. Dabei liegt der Grund für die Verriegelung bei einem anderen Objekt **O2**. Dies kann der Fall sein, wenn ein Aggregat nicht ein- oder ausgeschaltet werden darf, weil dies eine andere Prozeßgröße negativ beeinflussen würde. Verriegelungen sind steuerungstechnische Zusammenhänge und sind ebenfalls in der Regel aufgrund des steuerungstechnischen Wissens zur Laufzeit bekannt.

Sekundärstörungen und Verriegelungen sind bei der Behebung von Störungen einer Anlage von besonderer Bedeutung, da sie sich auf kausale (und auch temporale) Verknüpfungen einer Menge von Objekten beziehen können.

Die optische Filterung dient der Veranschaulichung des globalen Anlagenzustands. Bei einem größeren Störfall können eine große Menge von Alarmen zeitlich nahezu parallel eintreffen. Ein solcher Störfall ist unter Umständen durch den Ausfall eines einzigen Aggregates verursacht, führt aber in kurzer Zeit zu einer wahren Informationsflut.

Anhand des ersten Beispiels werden die Möglichkeiten der Dialogfilterung aufgezeigt. Bei diesem Beispiel handelt es sich um die Speisewasserversorgung des Dampfkessels der TAMARA-Anlage.

Das Beispiel an sich erscheint relativ trivial. Allerdings ist es anhand komplexerer Beispiele äußerst schwer, die Konzepte zu verdeutlichen, da hierzu eine große Menge an Wissen über den technischen Prozeß vorausgesetzt werden muß und dies den Rahmen jeglicher schriftlicher Darstellung sprengt.

Abb. 5.1 zeigt ein Blockdiagramm des Feuerungsteils der Anlage. Über die Speisewasserversorgung wird der Dampferzeuger gespeist. Jener fungiert als Wärmetauscher; er wandelt die Energie des aus dem Feuerraum strömenden Rauchgases in Prozeßdampf um und kühlt das Rauchgas damit ab.

Abb. 5.2 zeigt das Fließbild der Speisewasserversorgung im Überblick. Die Art und Weise der Darstellung ist an die üblichen Darstellungsweisen für chemisch-verfahrenstechnische Prozesse angelehnt.

Aus den Behältern **B13**, **B11** und **B14** wird der Speisewasserbehälter **B10** mit Frischwasser und Korrekturchemikalien gespeist. Die Speisung erfolgt über das Ventil **V20** und über die Pumpen **Pu3** und **Pu4**. Über die Hauptspeisewasserpumpen **Pu1** und **Pu2**, die redundant ausgelegt sind, und über das Ventil **V26** wird wiederum der Dampferzeuger (Kessel) gespeist.

Bei diesem einfach erscheinenden Prozeßteil existieren eine Vielzahl möglicher Störungen, Verriegelungen und Automatikschaltungen, auf die hier nur in so fern eingegangen werden soll wie sie in den folgenden Beispielen von Bedeutung sind.

In Abb. 5.3 wird zunächst die Bedeutung der Menüpunkte für die folgenden Beispiele gezeigt. Der erste Menüpunkt (das Dreieckssymbol im Kreis) stellt jeweils den Zustand dar, in den die Pumpe geschaltet werden kann. Der zweite Menüpunkt (**i**) dient der Auswahl kontext-abhängiger Information. Diese Information wird anhand des Prozeßwissens ausgewählt. Der dritte Menüpunkt schließlich stellt einen stilisierten technischen Plan dar.

Insbesondere das Fehlen der technischen Dokumentation in der Benutzeroberfläche ist ein großer Nachteil der heute in der Praxis eingesetzten Systeme zur Prozeß-

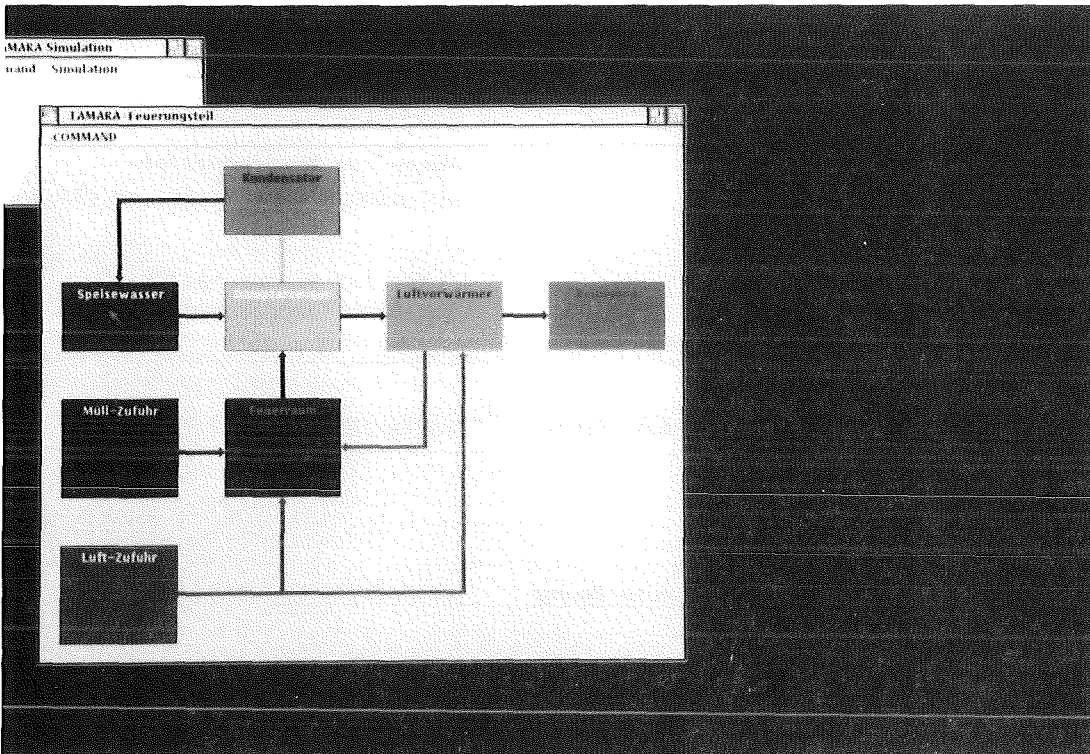


Abb. 5.1: Blockdiagramm des Feuerungsteils

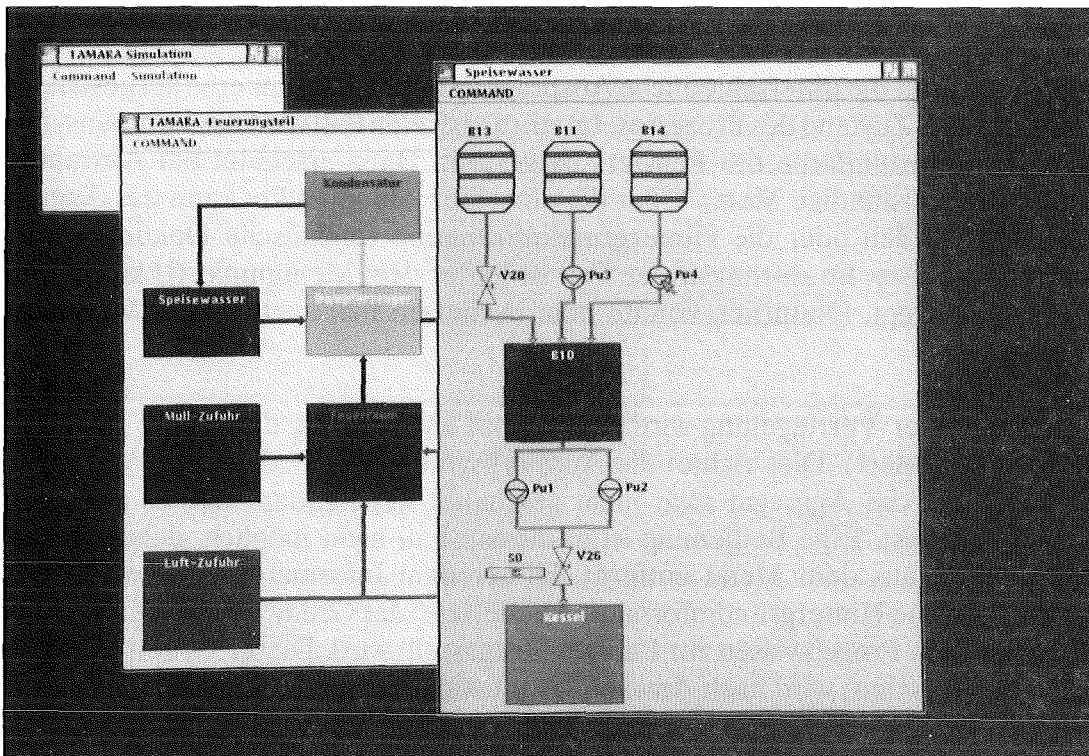


Abb. 5.2: Fließbild der Spisewasserversorgung






<i>Menüpunkt</i>	<i>Ausprägungen</i>	<i>Bedeutung</i>
		<i>Pumpe ist ausgeschaltet und kann eingeschaltet werden</i>
		<i>Pumpe ist eingeschaltet und kann ausgeschaltet werden</i>
	<i>nur eine Ausprägung</i>	<i>kontextabhängige Information</i>
	<i>nur eine Ausprägung</i>	<i>technische Dokumentation</i>

Abb. 5.3: Bedeutung der Menüpunkte

führung. Dem kann unter Verwendung von hochauflösender Graphik abgeholfen werden.

Abb. 5.4 zeigt die unterschiedliche Ausprägung des Dialogs für eine ausgeschaltete Pumpe je nach Zustand des Prozesses. Im störungsfreien Fall wird das Pumpensymbol in der Hintergrundfarbe des Fensters dargestellt. Dann erscheint bei Anwahl der Pumpe das vollständige Menü. Über die einzelnen Menüpunkte kann das Aggregat geschaltet werden oder die Hintergrundinformation (technische Dokumentation) angezeigt werden. Im störungsfreien Fall ist hinter dem Menüpunkt (i) keine Information hinterlegt. (Natürlich könnte man in diesem Fall auch diesen Menüpunkt ausblenden).

Im Fall einer Verriegelung wird das Symbol andersfarbig dargestellt (in diesem Fall grau realisiert). Dies richtet die Aufmerksamkeit des Bedieners direkt auf die Tatsache, daß das Aggregat zwar nicht geschaltet werden darf, daß es aber keine Störung aufweist. Eine Bedienung ist in diesem Fall nicht möglich, daher wird das Schaltsymbol aus dem Menü entfernt. Hinter dem Informationssymbol (i) ist in diesem Fall eine Hintergrundinformation hinterlegt: der Grund für die Verriegelung, wie er aus dem Prozeßwissen zur Laufzeit festgestellt wird. Dieser Grund (oder eine Liste von Gründen) wird nach Anwahl des Informationssymbols angezeigt.

Im Fall einer Primärstörung wird der Hintergrund des Pumpensymbols rot gefüllt. Hier kann bei Anwahl der Pumpe eine Liste möglicher oder auch direkt eine konkret

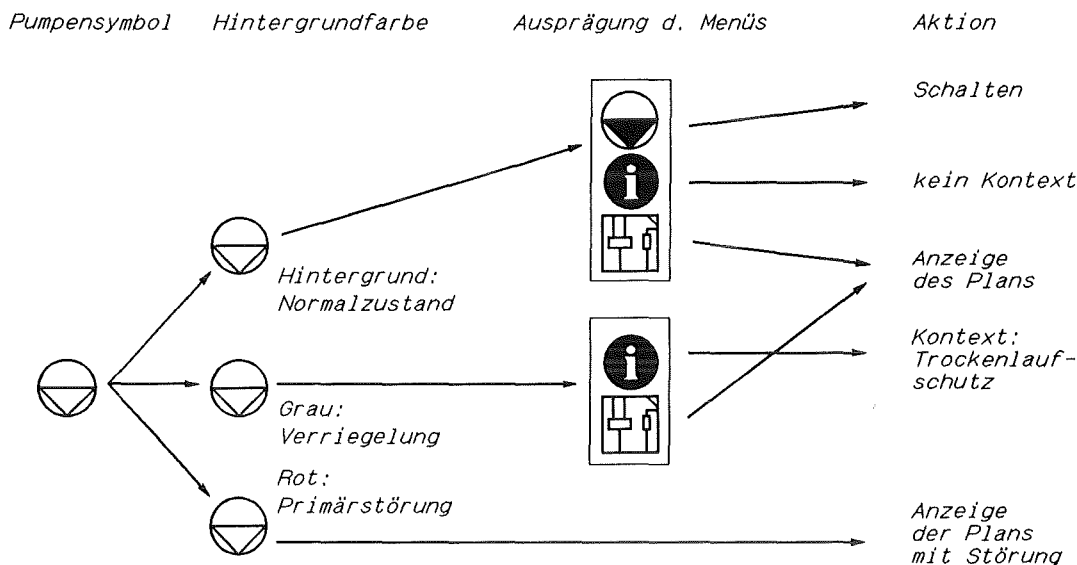


Abb. 5.4: Unterschiedliche Ausprägungen des Dialogs

beobachtete Störung dargestellt werden. Dies kann unter Umgehung des Menüs erfolgen, sofern die Störung eindeutig festgestellt werden konnte.

In Abb. 5.5 wird die Bedienung der Pumpe **Pu4** gezeigt. Nach Anwahl der Pumpe mit dem Zeiger erscheint ein Menü, welches an den jeweiligen Prozeßkontext angepaßt wird. (Dies wird in den folgenden Abbildungen gezeigt werden). Das offene Dreieck in **Pu4** zeigt an, daß die Pumpe ausgeschaltet ist und kein Durchfluß vorhanden ist. Ebenso sind die Verbindungslinien zur Pumpe hin und von der Pumpe weg offen dargestellt. Im Menü wird analog der Zustand angezeigt, auf den die Pumpe geschaltet werden kann (geschlossenes Dreieck entspricht "ein" bzw. "Durchfluß"). Nach Schalten der Pumpe werden Pumpensymbol und Verbindungslinien geschlossen dargestellt um anzuzeigen, daß nun Durchfluß vorhanden ist (Abb. 5.6). Nach einer erneuten Anwahl des Pumpensymbols ist das Menü gemäß dem aktuellen Zustand des Aggegats modifiziert.

Abb. 5.7 zeigt ein Fenster mit technischer Dokumentation der Pumpe, in diesem Fall ein einfacher Schaltplan mit der Ankopplung an das Automatisierungssystem. Im allgemeinen können Störungen auf jeglicher Ebene der Automatisierung auftreten, vom Aggregat über Ein-/Ausgabebaugruppen bis hin zum Automatisierungssystem selbst. Eine beträchtliche Anzahl dieser Störungen ist zur Laufzeit bekannt, so z.B. Störungen auf Baugruppen. Auch diese Störungen des Automatisierungssystems

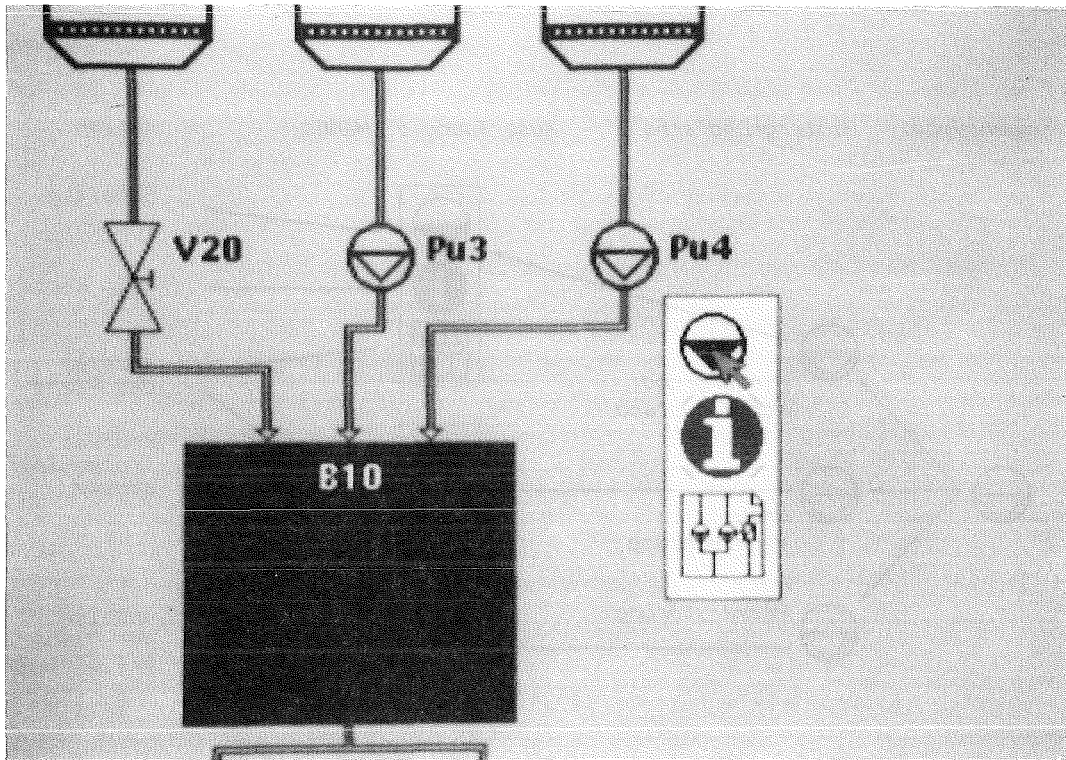


Abb. 5.5: Schalten der Pumpe

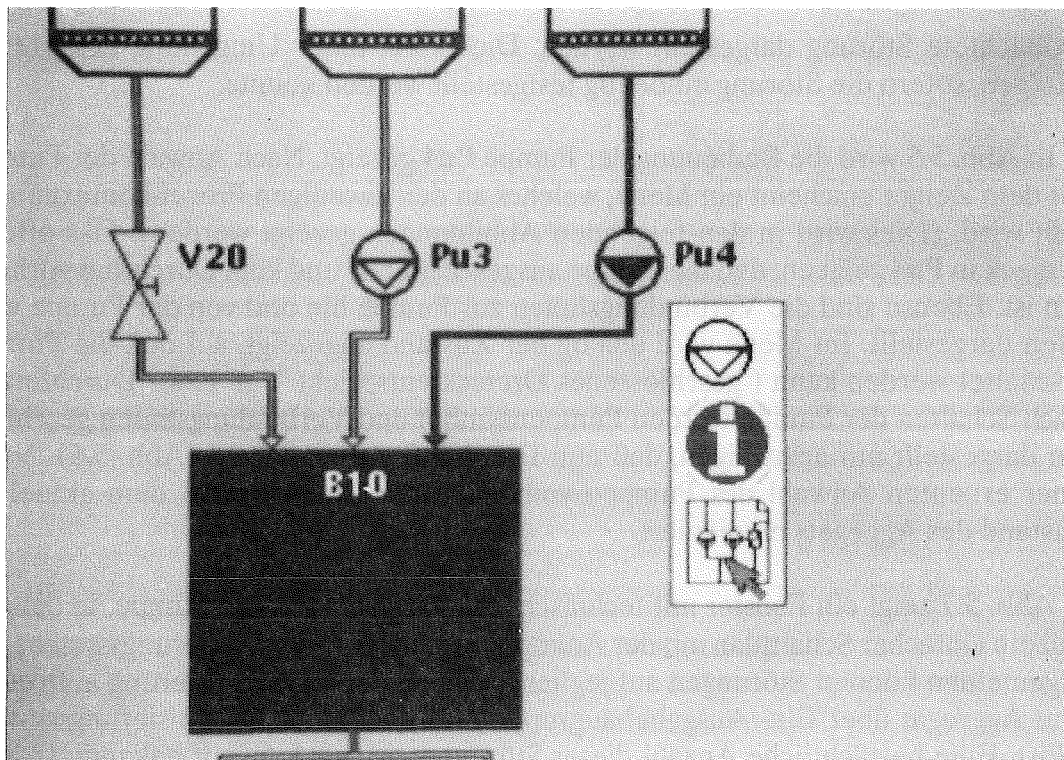


Abb. 5.6: Pumpe eingeschaltet; Anwahl des Plans

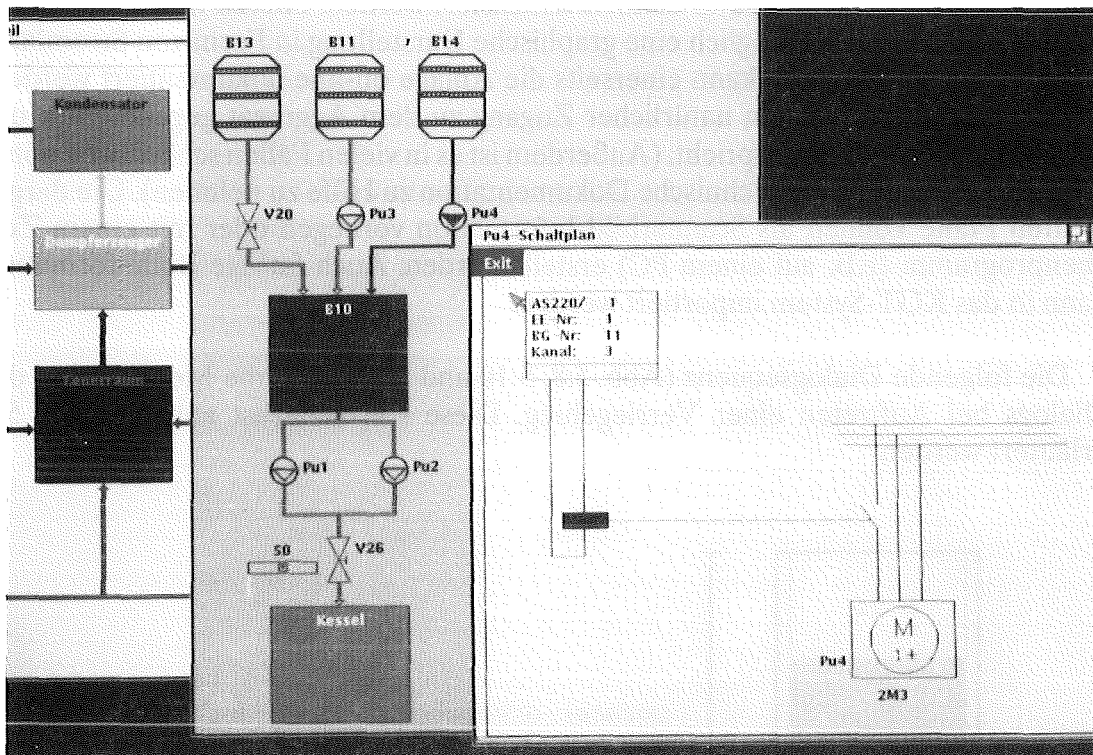


Abb. 5.7: Integration technischer Dokumentation

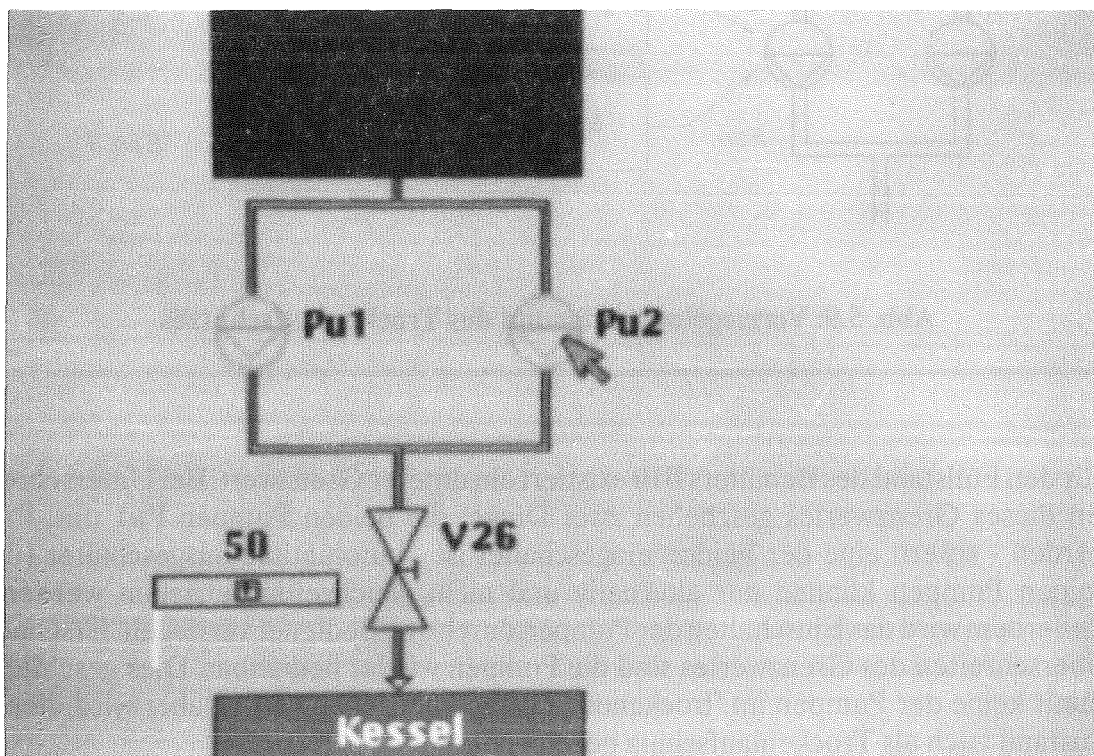


Abb. 5.8: Verriegelung der Pumpen Pu1 und Pu2

selbst sind von Bedeutung und müssen visualisiert werden. Hierzu wurde im vorliegenden Fall so weit als möglich eine graphische Darstellung in Form von Stromlaufplänen gewählt. Dadurch kann einerseits die Anlage on-line dokumentiert werden. Andererseits wird hier ein natürlicher Zugang zu dem Aggregat gegeben, der der Denkweise besonders entspricht. (Außerdem ist es in vielen Fällen schlichtweg nötig, zur Fehlerbehebung die technische Dokumentation zu Hilfe zu nehmen). Die dargestellten Pläne können als Scannerbild-Information vorliegen oder mit einem Zeichenprogramm (z.B. auf einem PC) erstellt werden. Auch farbige Bildinformation kann in das KGT-System importiert werden.

Die folgende Dialogsequenz (Abb. 5.8, 5.10 und 5.11) zeigt die Modifikation des Dialogs bei Auftreten einer Verriegelung. Diese soll zunächst anhand Abb. 5.9 erläutert werden.

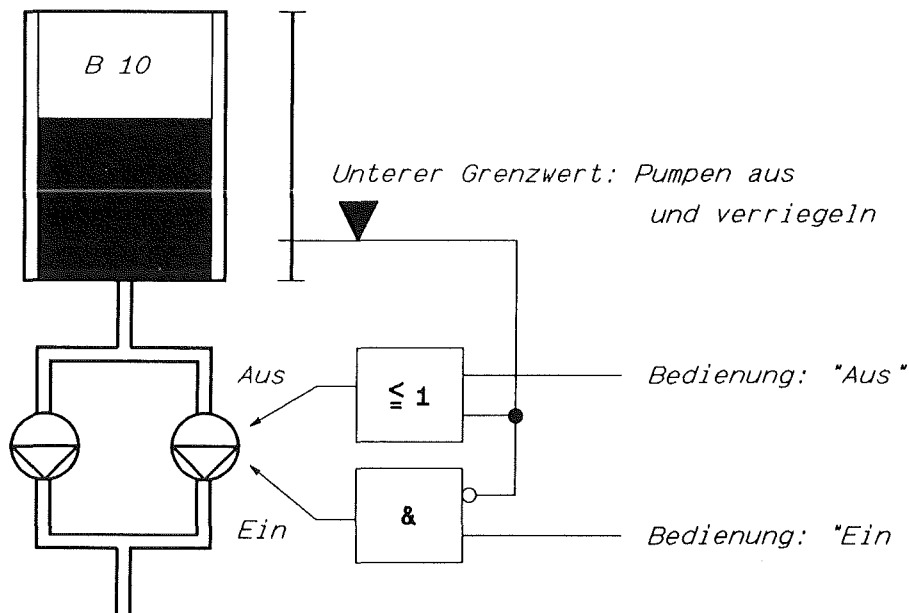


Abb. 5.9: Verriegelung aufgrund des Trockenlaufschutzes

Für den Füllstand des Behälters **B10** existiert ein unterer Grenzwert. Bei Unterschreiten dieses Grenzwertes geschehen zwei Dinge: die beiden Pumpen **Pu1** und **Pu2** werden - sofern eine der beiden eingeschaltet ist - automatisch ausgeschaltet (die beiden Pumpen können nur alternativ und nicht gleichzeitig betrieben werden). Außerdem wird das Einschalten der Pumpen durch den Bediener verriegelt. Erst nach Überschreiten des Grenzwertes sind die Pumpen wieder bedienbar. Dies geschieht, damit keine der Pumpen im "trockenen" Zustand betrieben wird. Daher wird dieser Zustand auch als Trockenlaufschutz bezeichnet.

Abb. 5.8 zeigt die beiden Pumpen bei vorhandenem Trockenlaufschutz. Die Pumpensymbole sind zur Verdeutlichung des Zustandes grau dargestellt. Zum Aufbau einer leicht verständlichen Benutzerschnittstelle werden sinnvollerweise ähnliche Zustände in der gesamten Benutzerschnittstelle gleich dargestellt, z.B. alle Primärstörungen rot, alle Sekundärstörungen orange und alle Verriegelungen grau. Einheitlichkeit und Konsistenz erleichtern die Bedienung.

Abb. 5.10 zeigt das Menü nach Anwahl der Pumpe. Das Symbol zum Schalten der Pumpe existiert in diesem Menü nicht, da ein Bedienen in diesem Zustand ohnehin nicht möglich ist. Darüberhinaus hebt auch das Menü den verriegelten Zustand der Pumpe hervor. Hinter dem Menüpunkt (i) (kontextabhängige Information) ist der Grund der Verriegelung auffindbar. Abb. 5.11 zeigt den Bildausschnitt nach Anwahl von (i).

Diese Dialogsequenz soll folgendes aufzeigen: unter Verwendung von Prozeßwissen ist es möglich, den Dialog derart zu gestalten, daß die Aufmerksamkeit des Operateurs jeweils auf den Prozeßkontext gelenkt wird. Dabei wird grundsätzlich davon ausgegangen, daß wenn möglich alle Informationen über ein bestimmtes Prozeßobjekt auch direkt bei diesem Objekt verfügbar sind. Außerdem wird der Dialog jeweils derart modifiziert, daß der Zugriff auf die zu einem Zeitpunkt wichtige Information so schnell als möglich gemacht wird.

Die folgende Dialogsequenz zeigt einen noch weiter reichenden Fall. Sie zeigt **Pu2** im gestörten Zustand, wobei die Störung an der Pumpe selbst festgestellt werden konnte. Die Ursache für die Störung ist ein gefallenes Bimetall. Es handelt sich also um eine Primärstörung. Die Pumpe wird daher mit rotem Hintergrund dargestellt (Abb. 5.12).

Nach Anwahl des Pumpensymbols erscheint kein Menü mehr, weil in diesem Fall die Kontextinformation - also die Information, die man hinter (i) hinterlegen würde - sich auf die Elektrik der Pumpe bezieht. In diesem Fall wurde es als sinnvoller angesehen, die Störung auch dort darzustellen, wo sie vorhanden ist, nämlich auf dem Stromlaufplan. Damit soll ein möglichst natürlicher Zugang zu der Information erreicht werden.

Die Anwahl der Pumpe führt also unter Umgehung des Menüs direkt zur Darstellung des Elektrolans (Abb. 5.13). Der Plan ist mit dem Plan identisch, den man auch im nicht gestörten Fall sehen würde. Allerdings ist zusätzlich die Störursache im Plan eingetragen und darüberhinaus rot hervorgehoben. Die Aufmerksamkeit wird direkt auf diese Information gelenkt. Die graphische Darstellung in Form eines Stromlaufplans impliziert unmittelbar, daß die Störung in der Elektrik vorliegt. Darüberhinaus wird die Information angegeben die nötig ist, um die Störung zu beheben.

Nach einer Abschätzung des Feuerungsteils von TAMARA ist es möglich, sämtliche Primärstörungen und - soweit bekannt oder gemessen - deren Ursachen sowie die Strategie zur Behebung in analogen graphischen Darstellungen anzugeben.

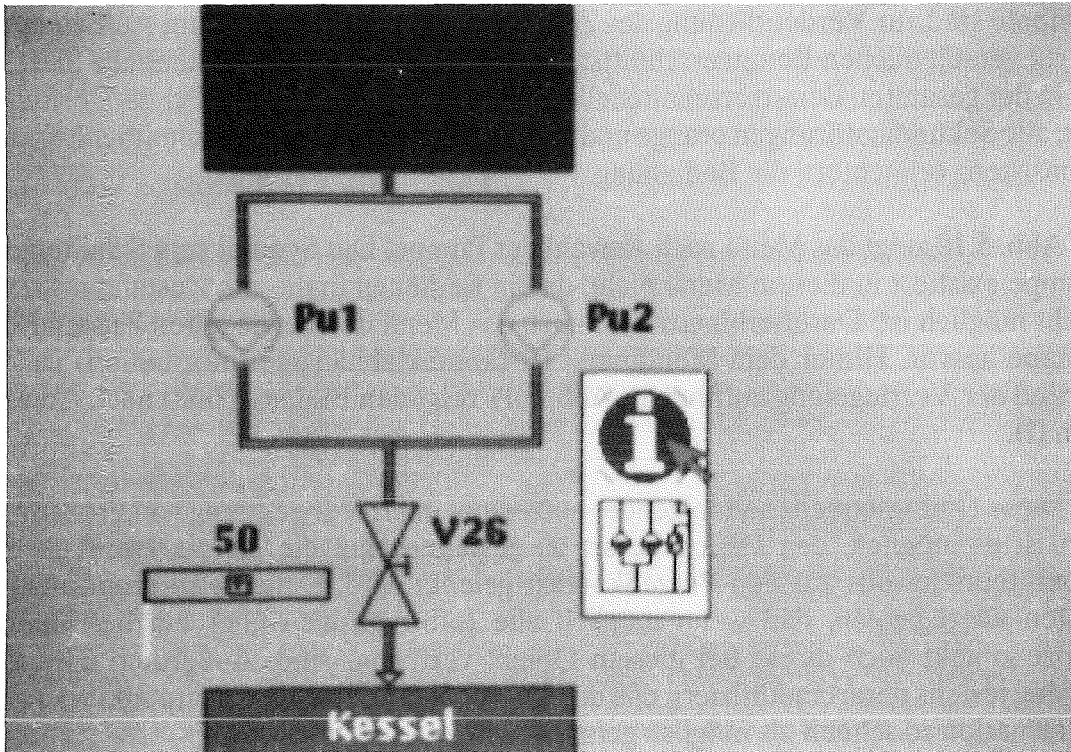


Abb. 5.10: Modifiziertes Menü

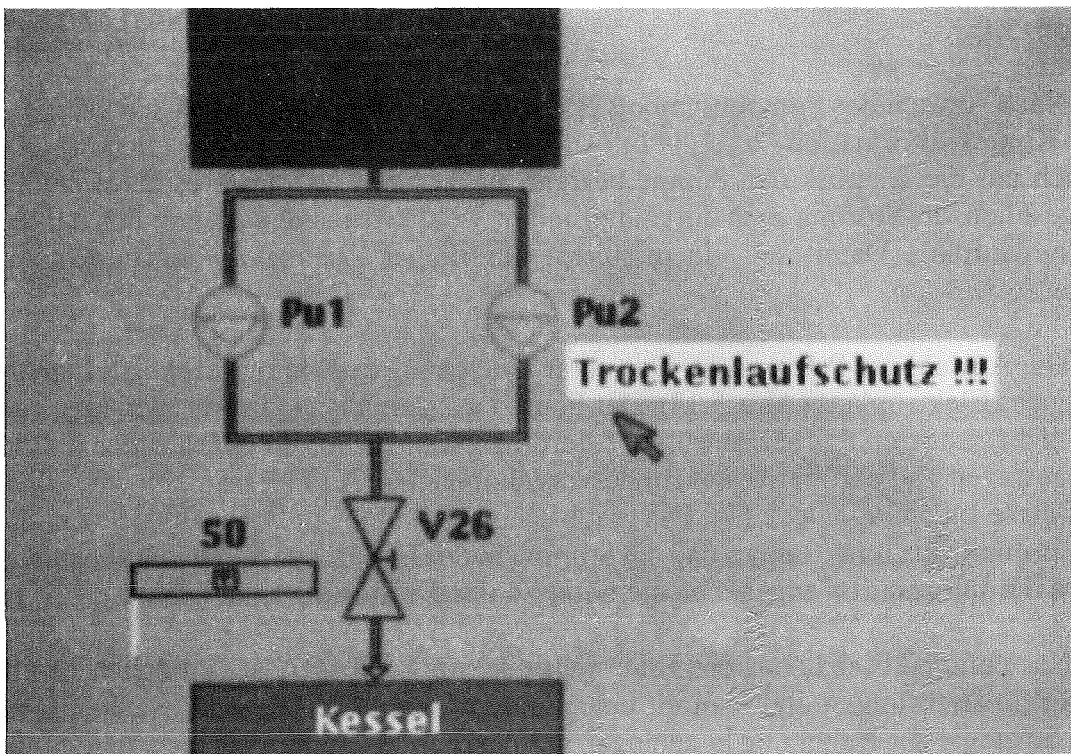


Abb. 5.11: Kontextinformation

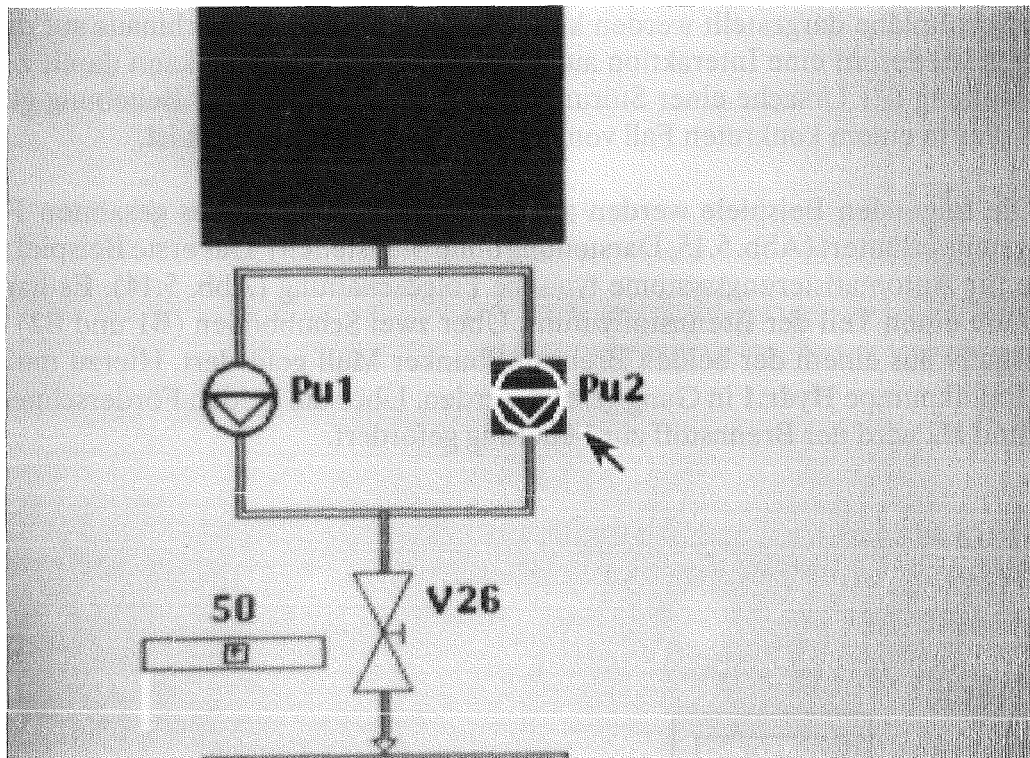


Abb. 5.12: Primärstörung bei Pu2

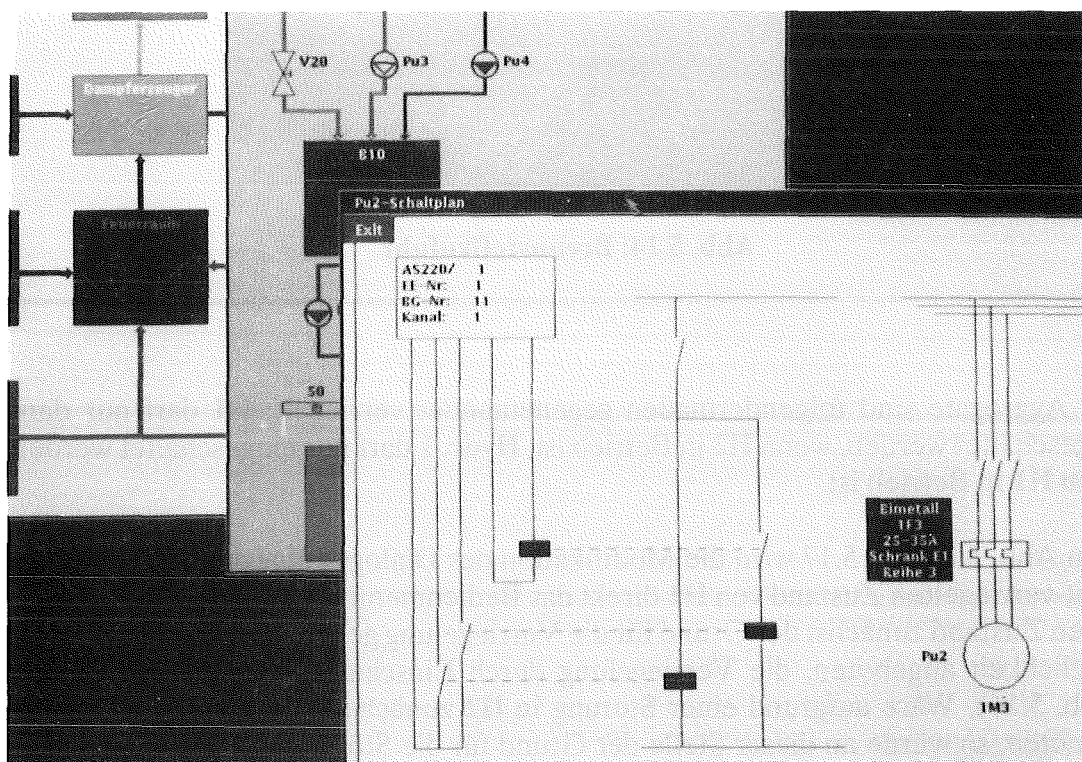


Abb. 5.13: Unmittelbare graphische Störanzeige

Entscheidend ist hierbei, daß mit dem KGT-System nicht nur Pixelinformation wie z.B. Schaltpläne dargestellt werden können, sondern daß darüberhinaus auf diesen Plänen wiederum eine Interaktion aufgebaut werden kann. Man kann damit zu der Darstellung der Ursache einer Störung auch die Möglichkeit zur Behebung geben, falls dies in einem konkreten Fall von der Schaltwarte aus möglich ist.

Die folgenden Beispiele werden anhand eines Fließbildes des gesamten Feuerungsteils erläutert (Abb. 5.15, Darstellung ohne Meßstellen). Das erste Beispiel zeigt eine für Automatisierungssysteme typische Folgeschaltung (Abb. 5.14). Es handelt sich um einen Teil der Brennstoffzufuhr. Über zwei Schubböden (**B1** und **B2**) wird alternativ aus einem der beiden Brennstoffbunker Müll gefördert. Hierzu muß die Hydraulikpumpe **Hydr.1** in Gang gesetzt werden. Über die beiden Förderschnecken **H1** und **H2** wird der Brennstoff zur Feuerung gefördert.

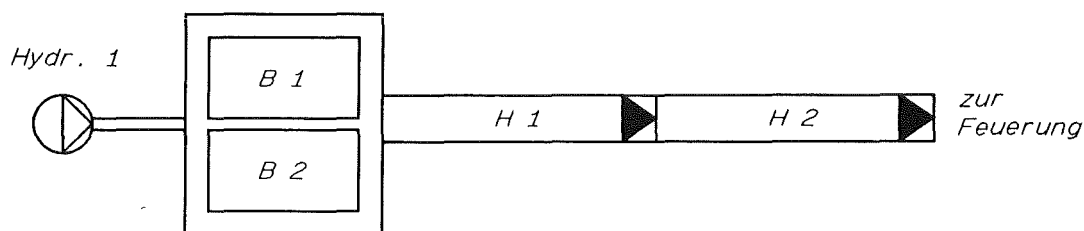


Abb. 5.14: Brennstoffzufuhr

Die Aggregate sind folgendermaßen gegeneinander verriegelt: **H1** darf nur dann eingeschaltet werden, wenn **H2** in Betrieb ist. **Hydr. 1** darf nur eingeschaltet werden, wenn **H1** in Betrieb ist.

In Abb. 5.16 und 5.17 wird die Modifikation des Dialogs aufgezeigt. Während im nicht-verriegelten Zustand von **H1** direkt das Bedienmenü erscheint, wird im verriegelten Zustand zunächst der Grund für die Verriegelung genannt und gleichzeitig die Möglichkeit angeboten, die Verriegelung durch Einschalten von **H2** zu beheben (Abb. 5.16). Wäre aufgrund einer Störung in **H2** zunächst eine Störungsbehebung vonnöten, so würde an dieser Stelle der Grund für die Störung in **H2** genannt. Hier

kommen die Verkopplungen mehrerer Aggregate ins Spiel. Nach Einschalten von H2 kann H1 geschaltet werden (Abb. 5.17).

Im vorliegenden Beispiel liegen die Aggregate nahe beieinander und der Zugriff wäre auch ohne Dialogfilterung möglich, allerdings mit zwei zusätzlichen Interaktionen. Oft liegt der Grund für eine Verriegelung an einer vollkommen anderen Stelle der Anlage und damit auch an einer entfernten Stelle in der Graphik. Womöglich ist die betreffende Stelle derzeit noch nicht einmal auf dem Bildschirm sichtbar. In diesem Fall beschleunigt die Dialogfilterung die Interaktion erheblich.

In diesem Kapitel wurden an ausführlichen Beispielen die Möglichkeiten gezeigt, die sich mit Dialogfilterung unter Verwendung von Prozeßwissen ergeben. Dialogfilterung dient hier in der Hauptsache der Beschleunigung des Verständnisses und der Interaktion, um bei Störfällen möglichst schnell reagieren zu können.

Für alle erwähnten Beispiele wurden die Dialoge implementiert. Die verwendeten Zusammenhänge sind in Form von steuerungstechnischem Wissen bekannt. Verwendete Zustände sind on-line meßbar.

Steuerungstechnische Zusammenhänge - insbesondere Verkopplungen - lassen sich auf natürliche Weise durch den in XAVIA gemachten Ansatz abbilden. Die objekt-orientierte Beschreibung der Aggregate spiegelt die Realität (genauer gesagt: deren Aussehen) identisch wider. Die Verwendung von Regeln führt dazu, daß programminterne Transaktionen (also die Verhaltensweise von Objekten bzw. von Mengen von Objekten) ebenfalls die natürliche Denkweise repräsentieren.

Es sei ausdrücklich darauf hingewiesen, daß die Prozeßgrößen in nebenläufigen Simulationsprozessen untergebracht wurden. Hierzu ist aufgrund des verteilten Objektsystems keinerlei Programmierung vonnöten. Die in der Benutzerschnittstelle verwendeten Prozeßgrößen werden lediglich als extern deklariert.

5.1.2 Dialogmanagement auf komplexen Prozessen

In diesem Kapitel werden zwei weitere Aspekte erörtert. Der eine Aspekt ist der Frage gewidmet, wie man hoch komplexe Prozesse sowohl vollständig als auch übersichtlich auf den Bildschirm abbilden kann. In diesem Zusammenhang werden auch mehrere nebenläufige Programme für eine Benutzerschnittstelle eingesetzt. Der zweite Aspekt ist die optische Filterung bei Störfällen. Beide werden anhand eines Fließbildes des Feuerungsbereiches von TAMARA erläutert.

Die im vorigen Kapitel gezeigten Beispiele wurden aus Platzgründen nicht vollständig dargestellt. In Wirklichkeit existieren für jedes der Aggregate noch eine Reihe weiterer Störursachen. Gruppen von Aggregaten weisen Automatikschaltungen auf, verschiedene Aggregate haben Hand-Notabschaltungen und dergleichen mehr. Die Information, die eine graphische Benutzerschnittstelle für eine gesamte Anlage

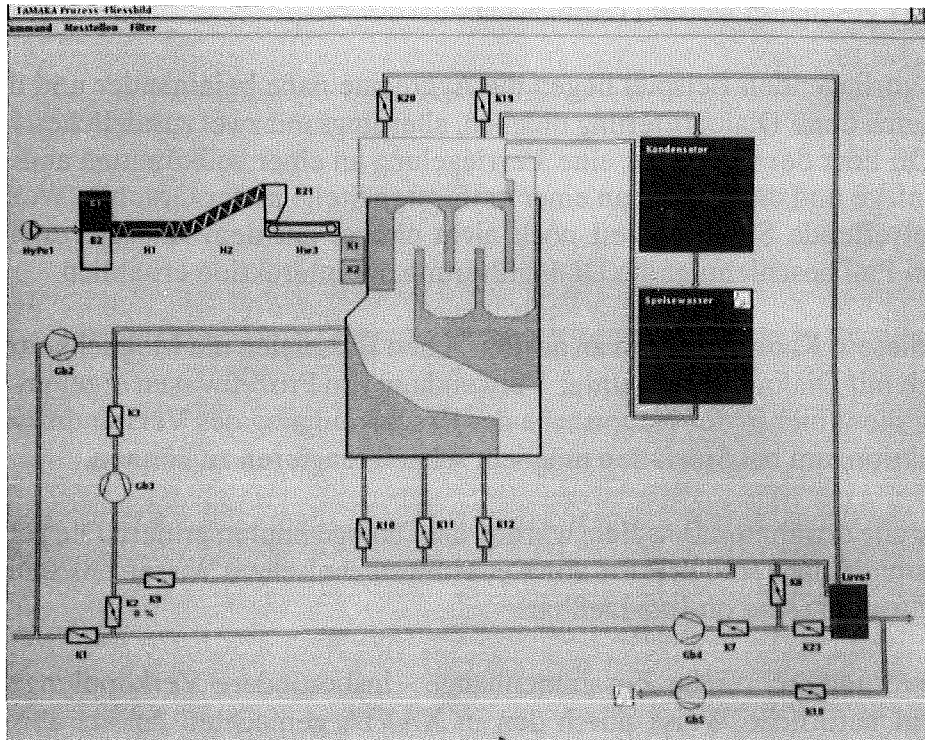


Abb. 5.15: Prozeßbild der Feuerung

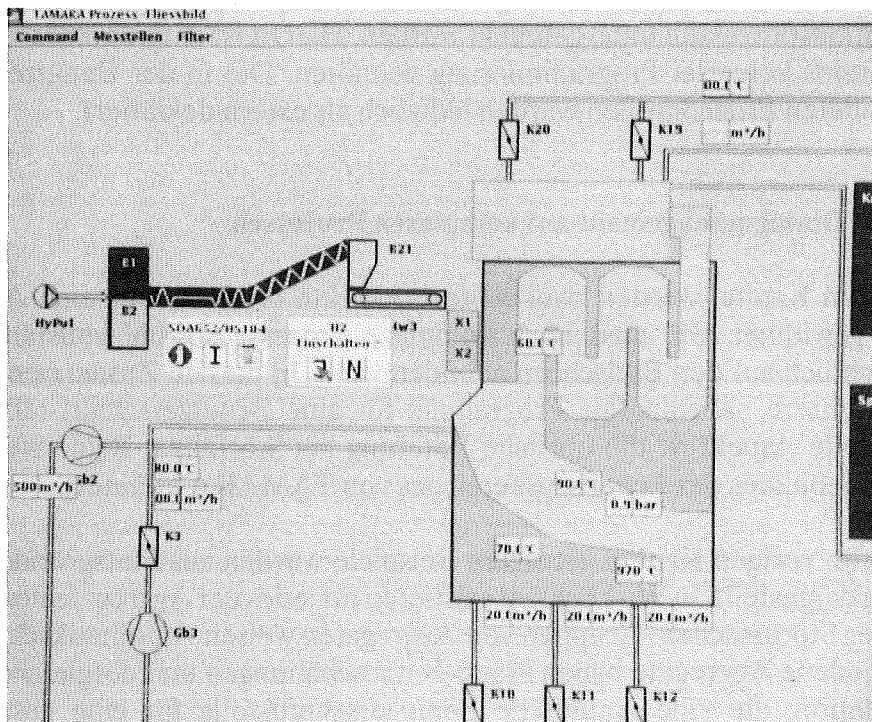


Abb. 5.16: Modifikation des Dialogs wegen Verriegelung

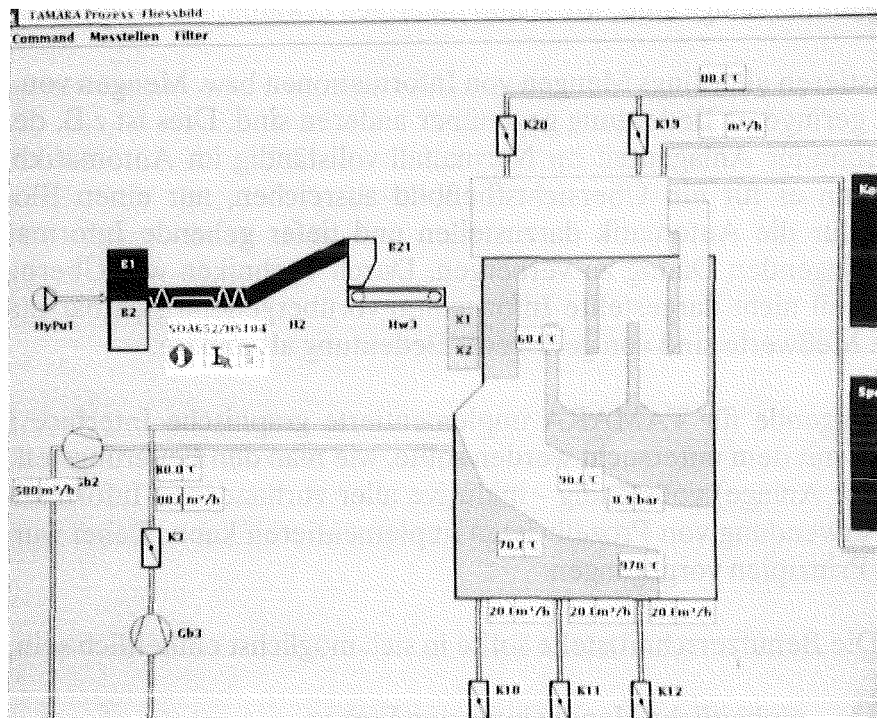


Abb. 5.17: Fortsetzung der Dialogsequenz

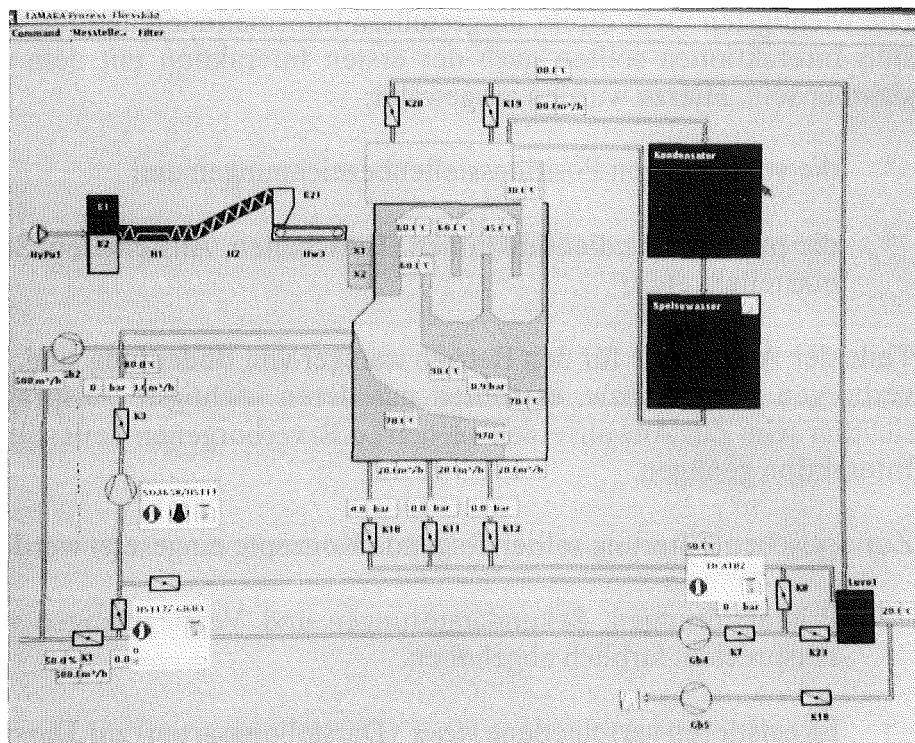


Abb. 5.18: Prozeßbild der Feuerung mit Menüs

übermitteln muß, ist daher - will man sie vollständig abbilden - überaus umfangreich. Die Zusammenhänge sind, wie schon mehrfach erwähnt, von komplexer Natur.

Nun existieren allerdings Mengen von Informationen bzw. Mengen von Aggregaten, die von geringerer Bedeutung gegenüber anderen sind. Dies ist z.B. der Fall, wenn ein bestimmter Anlagenteil im Normalfall vollständig im Automatikbetrieb läuft. Dann kann es für ein Übersichtsfließbild ausreichen, nur einen Block mit dem Schalter für die Automatik darzustellen und tiefer gehende Information in dem dahinterliegenden Dialog zu verbergen. Durch Techniken wie Überblenden kann man aktuell nicht dargestellte Information vorübergehend sichtbar machen. Auch manche Meßwerte sind von geringerer Bedeutung als andere.

Das folgende für TAMARA implementierte graphische Interface ist der erste Prototyp, mit dem untersucht werden sollte, wie man den Feuerungsteil (also ca. die Hälfte der Anlage) vollständig - inklusive aller vorhandenen Information - adaptiv unter Verwendung von Prozeßwissen implementieren kann. Dabei wurde nach folgenden Prinzipien vorgegangen:

- Die Benutzerschnittstelle sollte in sich möglichst einheitlich sein, d.h.
 - einheitliche Handhabung von Farben
 - einheitliche Handhabung von Symbolen
 - einheitlicher Aufbau von Menüs (soweit möglich)
- Die wichtigsten nicht direkt dargestellten Informationen und die wichtigsten Interaktionen sollten nach der ersten Interaktion mit dem Objekt sichtbar sein. Hierzu wurden ausgewählt:
 - die vollständigen Positionsstellenbezeichnungen und
 - die primären Bedienfunktionen (Einschalten, Ausschalten, Sollwert einstellen, etc.)
- Teile der Anlage, die für den Betrieb weniger von Bedeutung sind, sollten verborgen werden, bzw. es sollten nur deren wichtigste Ausprägungen sichtbar sein. Zur Anwahl eines solchen i.d.R. verborgenen Teils sollte eine Interaktion genügen.
- Zur optischen Filterung sollten folgende Konzepte eingesetzt werden:
 - Primärstörungen, Sekundärstörungen und Verriegelungen werden entsprechen farblich abgehoben.
 - Es existieren verschiedene *views* (Darstellungsarten) zur Darstellung bestimmter Aspekte.

Anhand der folgenden Beispiele soll der Entwurf der Benutzerschnittstelle unter Verwendung vorstehender Prinzipien gezeigt werden. Es sei voraus geschickt, daß die Auswahl der Farben und Symbole in einzelnen Fällen vom ergonomischen Standpunkt aus gesehen mehr oder minder glücklich getroffen sein mag. Dem Projekt stand allerdings kein Ergonom zur Verfügung. Zum großen Teil sind aber auch Symbole durch DIN-Normen vorgegeben und bestimmte Farben für bestimmte Vorkommnisse in der chemisch-verfahrenstechnischen Industrie gebräuchlich. Es wurde versucht, sich hier so weit als möglich an der Praxis zu orientieren.

Abb. 5.19 gibt eine Übersicht über die zur Filterung verwendeten Farben und über die zur Dialogsteuerung verwendeten Symbole. Die zur Darstellung der Aggregate





	<i>Ausprägung</i>	<i>Darstellung</i>	<i>Bedeutung</i>
Farben	<i>Rot</i>	<i>Hintergrund füllen</i>	<i>Primärstörung</i>
	<i>Orange</i>	<i>Hintergrund füllen</i>	<i>Sekundärstörung</i>
	<i>Grau</i>	<i>Hintergrund füllen</i>	<i>Verriegelung</i>
Symbole		<i>an der Stelle eines verborgenen Teilsystems</i>	<i>Beginn eines Dialogs</i>
		<i>rechts oben im Dialogfenster</i>	<i>Ende eines Dialogs</i>
		<i>im ersten Menü</i>	<i>Kontextinformation</i>
		<i>im ersten Menü</i>	<i>techn. Dokumentation</i>

Abb. 5.19: Verwendete Farben und Symbole

verwendeten Symbole (Pumpen, Gebläse, Klappen etc.) sind hier nicht verzeichnet. Sie sind in den folgenden Bildschirm-Fotographien sichtbar.

Die Aggregate werden in ihrem Zustand symbolisch dargestellt, Meßwerte durch ihren Zustand (z.B. rot bei Bereichsüberschreitung), ihren Wert und die physikalische Einheit. Bei Anwahl eines Aggregats bzw. Meßwertes werden als Primärinformation die Positionsstellenbezeichnung und bei Aggregaten die wichtigste Bedienfunktion angeboten. Weitere Informationen sind hinter den Informationssymbolen und Dokumentationssymbolen vorgehalten.

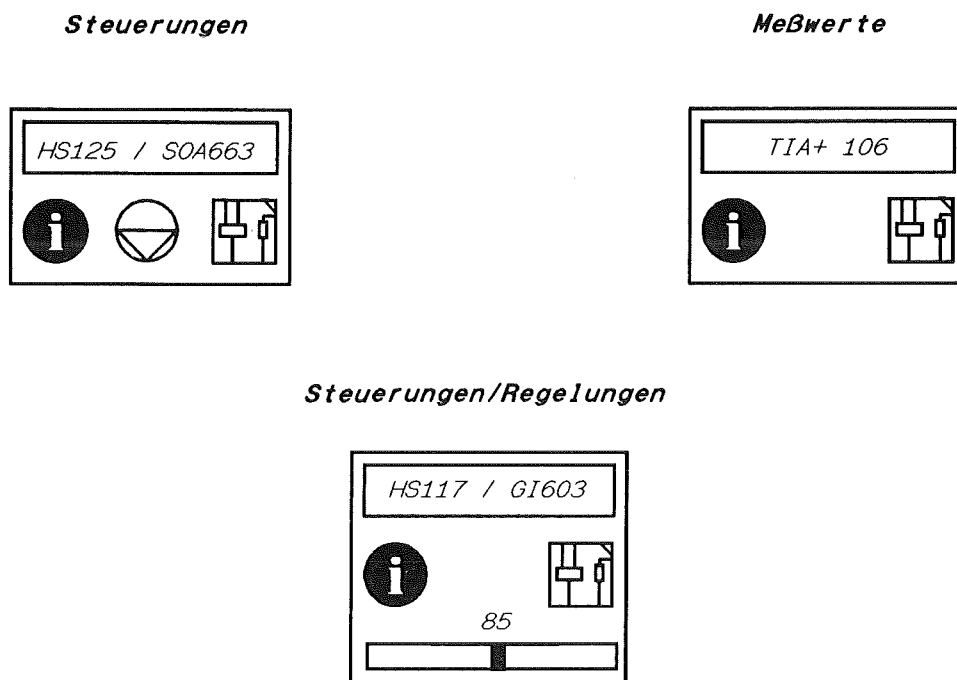


Abb. 5.20: Verwendete Menütypen

Es wurde versucht, die Menüs möglichst einheitlich zu gestalten, was bei Aggregaten aufgrund deren unterschiedlicher Bedienfunktionen nicht ganz einfach ist. Abb. 5.20 zeigt die realisierten Menüs. Die Bedienfunktionen werden wie im Beispiel des letzten Abschnitts bei Nicht-Bedienbarkeit jeweils ausgeblendet.

Abb. 5.18 zeigt einige der realisierten Menütypen auf dem Bildschirm. Die Bedienung und Anpassung an den Prozeßzustand ist analog der Vorgehensweise im letzten Abschnitt.

Das folgende Beispiel zeigt die Anwendung von Informationszooming durch ein überdeckendes Fenster. Für den Betrieb der Luftzufuhr der Feuerung ist vorrangig die sog. Primärluft von Bedeutung. Über die Klappen K10, K11 und K12 wird Verbrennungsluft von unten in den Feuerraum eingeblasen. Außer der Sekundärluft, die über Gb3 und K3 gespeist wird, gibt es noch die Möglichkeit, einen Teil des Rauchgases zurückzuführen. Dies geschieht über das Gebläse Gb5.

Die Rauchgasrückführung ist von ihrer Bedeutung für die Prozeßführung her der Primärluft untergeordnet. Da im Bereich unterhalb des Feuerraums eine zu große Menge an Information untergebracht werden muß und das Prozeßbild leicht unübersichtlich wird, wurde eine Möglichkeit geschaffen, die Rauchgasrückführung wegzu- blenden. Sie wird lediglich in den Teilen weggeblendet, in denen innerhalb der

Graphik die Gefahr der Unübersichtlichkeit gegeben ist. Dies betrifft die Klappen **K27**, **K28** und **K29**, sowie die Meßstellen **F753**, **F754** und **F755**. Das Gebläse **Gb5** und die Klappe **K18**, mit denen die Rauchgasrückführung global gefahren werden kann, wurden von dieser Ausblendung ausgenommen.

Bei Auftreten eines Notprogramms werden sämtliche Gebläse automatisch abgeschaltet. Nach Behebung der ursächlichen Störung sind diese wieder in Betrieb zu nehmen. Das Schalten des Gebläses **Gb5** ist also eine primär wichtige Bedienfunktion. Dies kann durch diese Aufteilung direkt auf dem Prozeßbild geschehen. Es ist also eine direkte Bedienbarkeit für diese wichtige Funktion gegeben, wohingegen weniger kritische Funktionen (die Einstellung der Rauchgasrückführung) aus dem Prozeßbild ausgelagert sind. Abb. 5.18 zeigt die Rauchgasrückführung im ausgeblendeten Zustand. Nach Anwahl des Dialogsymbols wird das Fenster der Rauchgasrückführung über das Prozeßbild gelegt (Abb. 5.21). In diesem Fenster sind die Aggregate und Meßstellen wie alle anderen analog bedienbar. Das Fenster wird durch weitere Anwahl des Dialogsymbols rechts oben im Fenster wieder unsichtbar.

In ähnlicher Weise wurde im nächsten Beispiel vorgegangen. Die schon bekannte Speisewasserversorgung wird durch die beiden Pumpen **Pu1** und **Pu2** primär bedient. Die nachgeordneten Aggregate werden im Regelfall im Automatikbetrieb betrieben. Im Prozeßbild werden daher lediglich diese zwei Aggregate dargestellt. Nach Anwahl des Dialogsymbols erscheint die Speisewasserversorgung, allerdings diesmal in einem eigenen frei beweglichen Shell-Window (Abb. 5.22). Dieses Fenster wird in einem nebenläufigen Programm betrieben, d.h. *die Benutzerschnittstelle an sich ist verteilt*.

Durch den verteilten Ansatz ist diese Vorgehensweise ohne besondere Umstände möglich. Zur Steuerung der Anwahl wird je ein Kontrollobjekt in jedem Programm verwendet (Abb. 5.23). Das Attribut **Bild-an** des Kontrollobjekts **St-1** des Feuerungsbildes wird bei Anwahl des Dialogsymbols in den Zustand "ein" überführt. Das Kontrollobjekt **St-2** des Speisewasserbildes besitzt ebenfalls ein Attribut **Bild-an**, dessen Wert als extern deklariert ist. Nach Austausch der Meldung feuert eine Regel, die das Speisewasserbild auf dem Bildschirm darstellt. Die Pfeile in Abb. 5.23 symbolisieren die Externdeklarationen der Attribute.

Da die Aggregate und Meßwerte auch im Speisewasserbild in ihrem aktuellen Zustand dargestellt werden müssen, benötigen auch sie die Prozeßzustände. Hierzu kann ein Zustand (in der Abbildung der Zustand der Pumpe **Pu1**) als extern im Prozeßinterface deklariert werden. Dadurch ergeben sich zwei Meldungen, die über das Netz von **node1** nach **node2** transportiert werden. Der Zustand kann alternativ auch aus dem Feuerungsbild gewonnen werden. In diesem Fall wird eine Meldung über das Netz transportiert und eine Meldung innerhalb des Visualisierungsrechners **node2**.

Jeder Teil einer graphischen Anwendung, der ein eigenes Shell-Window besitzt, läßt sich auf diese Art und Weise parallelisieren. Der Aufwand beim Entwurf ist dabei geringfügig.

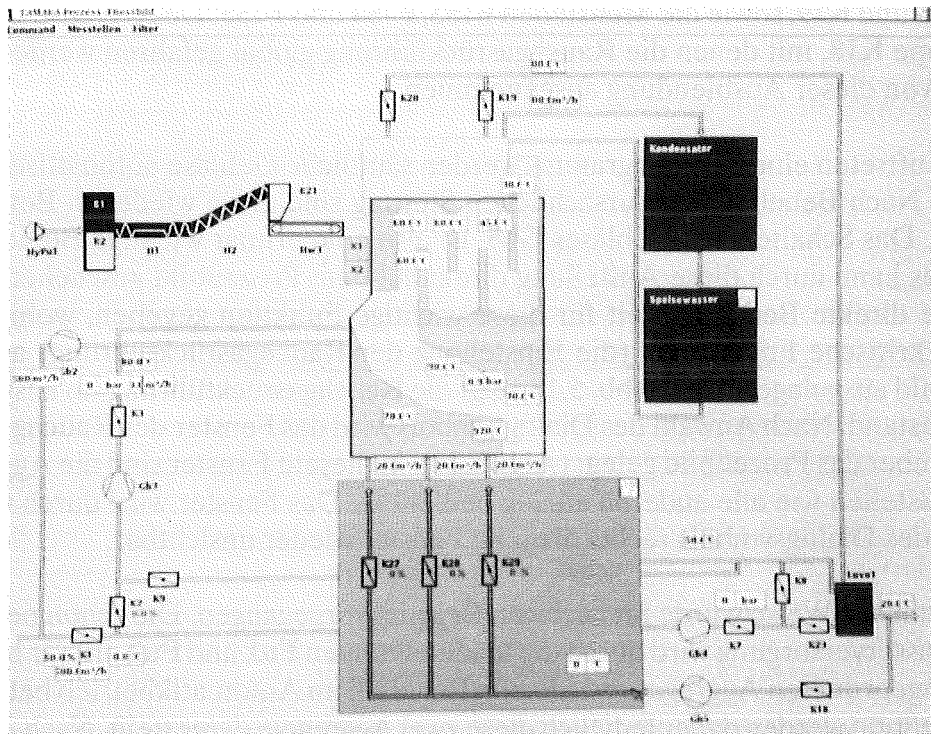


Abb. 5.21: Informationszooming durch Überdeckung

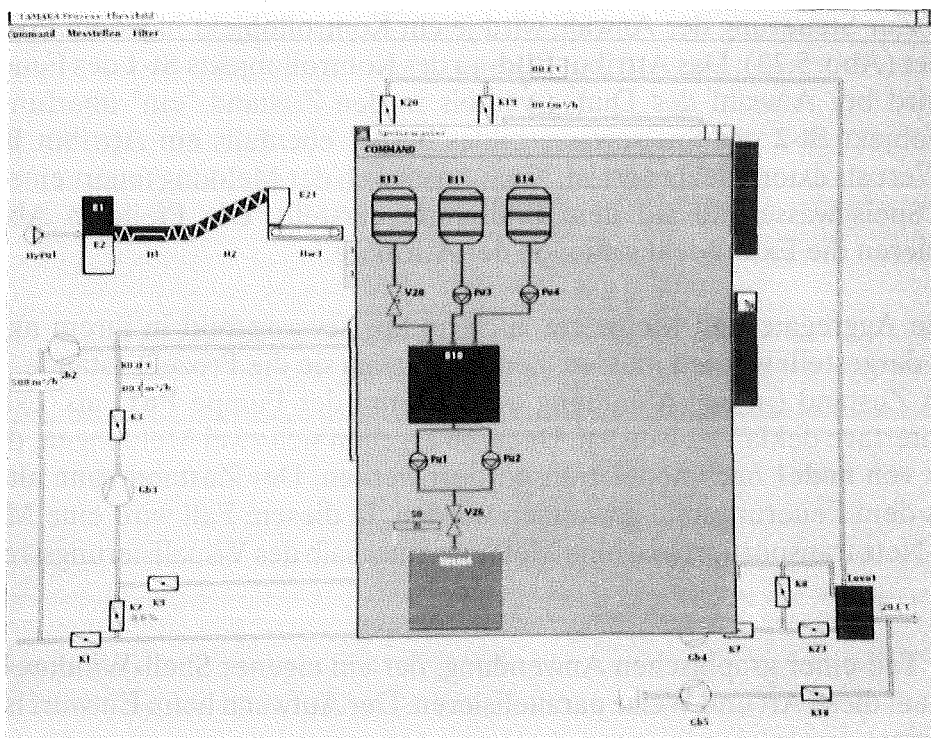


Abb. 5.22: Teilprozeß in einem nebenläufigen Programm

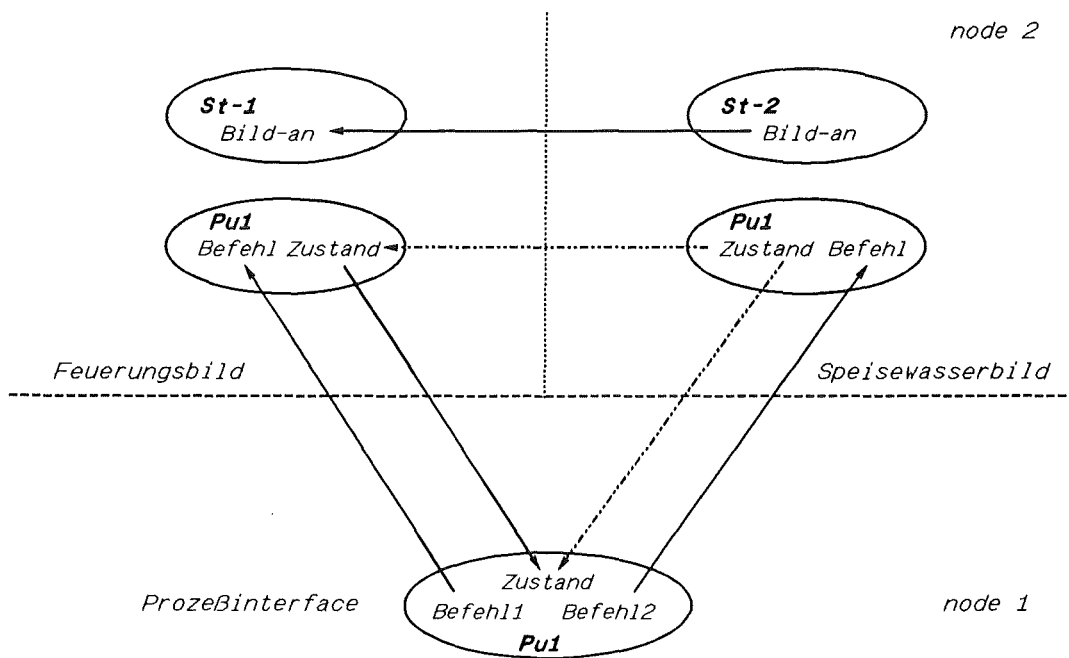


Abb. 5.23: Objekte im verteilten User Interface

Die folgenden Beispiele sind der optischen Filterung in einem vollständigen Anlagenbild gewidmet. Abb. 5.24 zeigt einen (fiktiven) Alarmzustand, bei dem Primärstörungen, Sekundärstörungen und Verriegelungen farblich unterschiedlich dargestellt werden. Der Zweck dieser Vorgehensweise liegt darin, zunächst die Aufmerksamkeit auf die Primärstörungen zu lenken. Die Suche nach Fehlern in der Anlage wird damit unterstützt. Nach Beseitigung einzelner Primärstörungen können u.U. schon verschiedene Sekundärstörungen und Verriegelungen wieder aufgehoben sein.

Es wäre auch denkbar, daß man zuvor eingeschaltete Aggregate, die wegen einer fremden Ursache ausgeschaltet und verriegelt wurden, mit einer weiteren Farbausprägung kennzeichnet. Hierdurch soll die Aufmerksamkeit darauf gelenkt werden, daß das betreffende Aggregat wieder eingeschaltet werden soll. Dies wurde allerdings bei diesem Beispiel nicht realisiert.

Die abschließenden Beispiele zeigen eine Filterung durch Verwendung unterschiedlicher Schnitte (*views*) durch die Meßwerte der Anlage. Die Meßwerte wurden hierzu in zwei Mengen aufgeteilt: eine technologische Menge und eine Wichtigkeitsmenge. Die Teilmengen der technologischen Menge beinhalten jeweils Meßstellen gleicher Art, also Temperaturen, Durchflüsse, Drücke, etc. Die Teilmengen der Wichtigkeitsmenge sind nach der Bedeutung der Meßstellen für die Prozeßführung aufgliedert. So sind z.B. in der Primärluft die Durchfluß-Meßstellen **F746**, **F747**

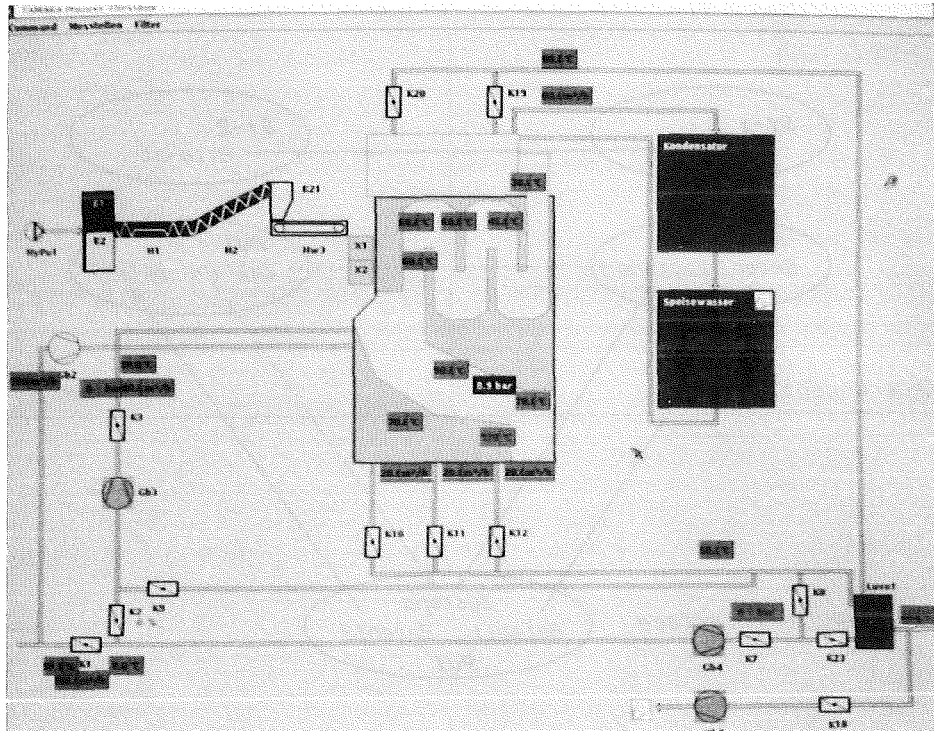


Abb. 5.24: Optische Filterung

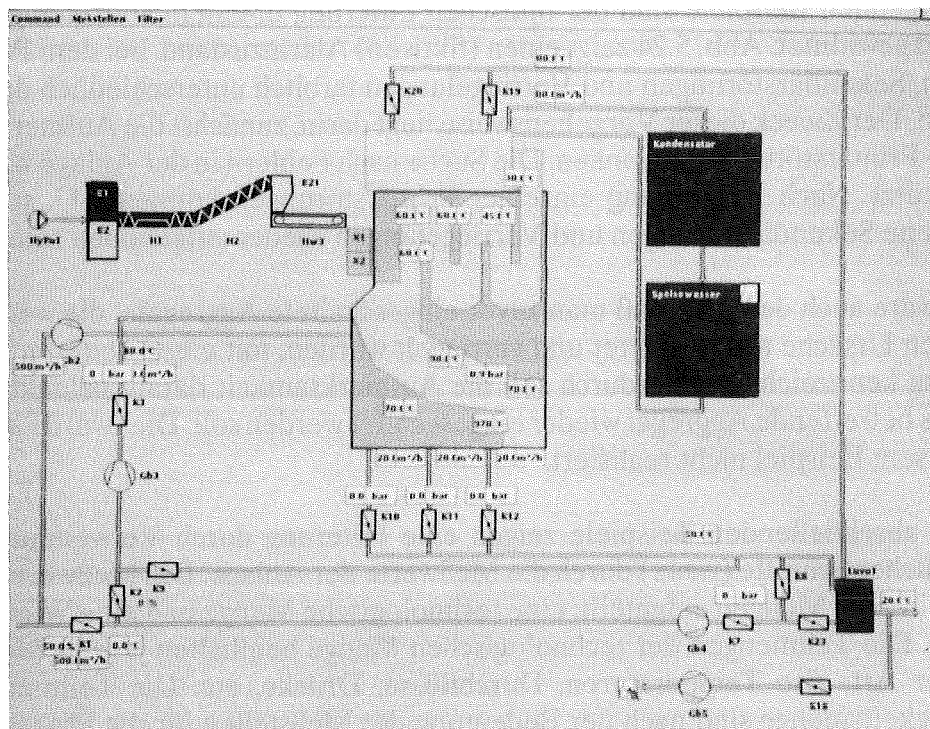


Abb. 5.25: View 0 - Anzeige aller Meßstellen

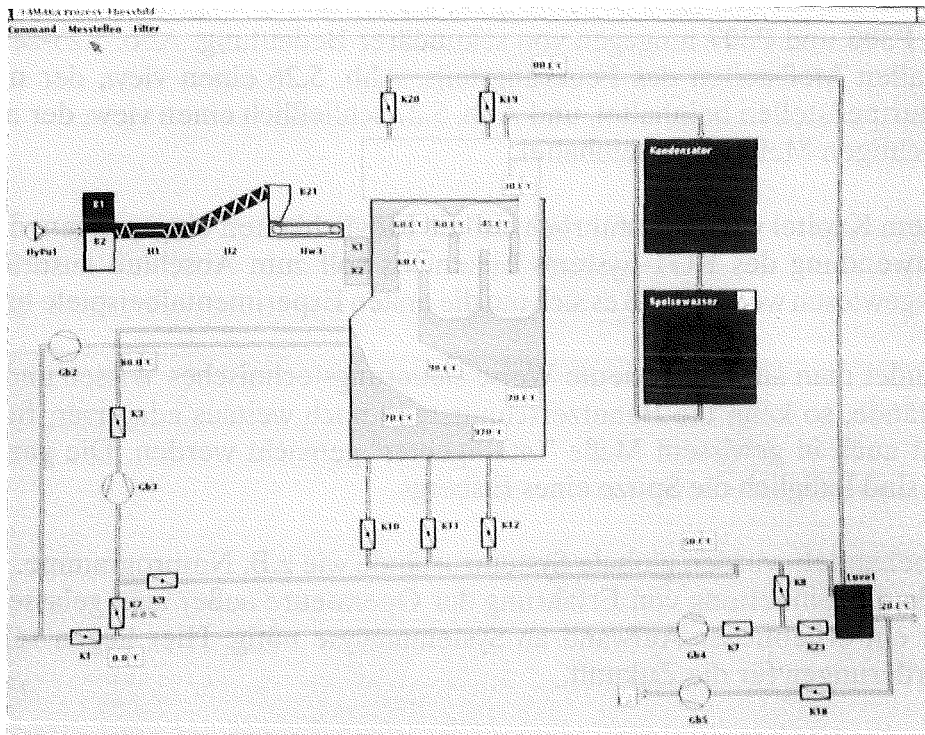


Abb. 5.26: View 1 - Anzeige der Temperaturen

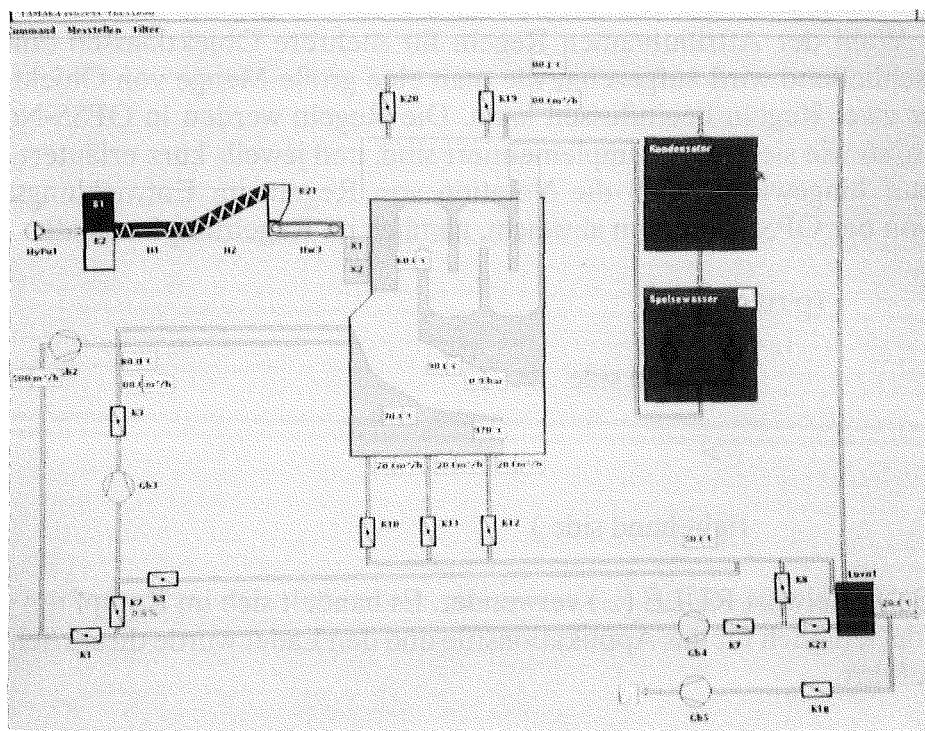


Abb. 5.27: View 2 - Anzeige der primären Meßstellen

und F748 von primär wichtiger Bedeutung für die Prozeßführung, die Druckmeßstellen P305, P306 und P307 hingegen von sekundärer Bedeutung. Abb. 5.25 zeigt ein Bild mit allen Meßstellen des Feuerungsteils, Abb. 5.26 einen view, der nur die Temperaturmeßstellen beinhaltet, und Abb. 5.27 schließlich einen view, der nur die primär wichtigen Meßstellen beinhaltet.

In diesem Kapitel wurde ausführlich auf die Möglichkeiten eingegangen, die sich unter Verwendung des KGT-Systems bieten. Es soll zum Abschluß ausdrücklich darauf hingewiesen werden, daß es sich um die ersten Experimentalbeispiele handelt.

Verwendet man auf konsequente Weise steuerungstechnisches Wissen und meßbare Zustände, so kann die Benutzerschnittstelle noch weitaus adaptiver, flexibler und damit auch in gewissem Maße "intelligenter" gemacht werden. Die gezeigten Beispiele sind lediglich die Spitze eines Eisbergs.

Nicht behandelt wurden globale Systemzustände wie z.B. Notprogramme. Ebenfalls wurde die Einbettung von Erfahrung der Operateure außer acht gelassen. Für beides ist ein erheblicher Aufwand an Systemanalyse nötig. Hier liegen reizvolle Herausforderungen für die Zukunft.

5.1.3 Modellierung des Feuerungsteils

Dieser Abschnitt soll einen Einblick in die konkrete Modellierung der voran gegangenen Beispiele geben. Zunächst werden jeweils die Modelle der Meßstellen, der Klappen und Gebläse kurz erläutert. Es wird darauf eingegangen, wie man durch geeignete Wahl der Attributnamen Regeln für mehrere Objektklassen einführen kann. Anschließend wird aufgezeigt, wie man eine große Menge von Objekten mit einer oder zwei Regeln beeinflussen kann. Die Regeln werden in OPS5-Notation angegeben, so wie sie konkret implementiert sind, und jeweils kurz erläutert. Es sei noch darauf hingewiesen, daß die Notation von Regeln im Entwicklungssystem minimal von der OPS5-Notation abweicht, nämlich im Regelrumpf: anstatt

```
(p rule-name
    left hand side
    -->
    right hand side )
```

wird das Schlüsselwort RULE (...) verwendet. Es handelt sich im Rumpf um echten OPS5-Code, lediglich für das Applikationsfile und den Lader wurde dieses Schlüsselwort eingeführt.

Eine Meßstelle ist wie folgt modelliert:

<i>OBJECT</i>	<i>Name =</i>	T101
	<i>Class =</i>	Meßstelle
	<i>Priority =</i>	8

Sie besitzt folgende nicht-graphische Attribute:

MstWert

externes Attribut, welches als Empfänger des Meßwertes dient

Grenzwert

Wert für Alarmschwelle

Wichtigkeit

Schalter zum Unterscheiden primärer und sekundärer Meßstellen

Hinzu kommen die folgenden graphischen Attribute:

MstZeigeWert

graphische Darstellung des Werts

MstZeigeEinheit

Darstellung der Einheit

MstWorkArea

Enthält das Meßstellenmenü

MstBezeichner

Positionsstellenbezeichnung der Meßstelle

Info

Info-Pushbutton

Plan

Plan-Pushbutton

PlanWorkArea

Fenster für Anzeige des Plans; die auf dem Plan wieder auftretenden graphischen Merkmale werden des Umfangs halber hier nicht dargestellt

Von den Regeln sollen zwei kurz erläutert werden. Folgende Regel ändert die Anzeige des Meßwertes, falls ein neuer Wert vom Meßwertprozeß eintrifft:

```
RULE (AendereWert
      (ATTRIBUTE ^My_Obj <object>
                ^Name MstZeigeWert
                ^Mode Show)
      (ATTRIBUTE ^My_Obj <object>
                ^Name MstWert
                ^Value <val> )
      -->
      (CALL KGT$SetLabel <object>
        MstzeigeWert (OPSFormat <val> 3))
    )
```

(Dabei dient die Funktion *OPSFormat* der Formatierung des Wertes). Die folgende Regel modifiziert das Aussehen des Meßwertes bei Überschreitung des Grenzwertes:

```
RULE (Grenzwert
      (ATTRIBUTE ^My_Obj <object>
                ^Name MstWert
                ^Value <val> )
      (ATTRIBUTE ^My_Obj <object>
                ^Name Grenzwert
                ^Value { <= <val> })
      -->
      (CALL KGT$SetBackground <object>
        MstzeigeWert Red )
      (CALL KGT$SetBackground <object>
        MstzeigeEinheit Red )
      (CALL KGT$SetForeground <object>
        MstzeigeWert White )
      (CALL KGT$SetForeground <object>
        MstzeigeEinheit White )
    )
```

Beide Regeln feuern jeweils auf eine Veränderung des externen Attributs *MstWert* hin, die zweite allerdings nur, wenn der Vergleich mit dem Grenzwert entsprechend ausfällt.

Als nächstes Beispiel sei das Modell eines Gebläses angegeben. Ein Objekt der Klasse Gebläse ist wie folgt realisiert:

<i>OBJECT</i>	<i>Name =</i>	Gb3
	<i>Class =</i>	Gebläse
	<i>Priority =</i>	8

Es besitzt als externes technisches Attribut das Attribut

Wert

Dieses Attribut stellt den Zustand des Gebläses dar und kann den Wert "ein" oder "aus" annehmen.

Hinzu kommen die graphischen Attribute:

GbSymbol

Enthält das aktuelle Gebläsesymbol

GbText

Bezeichner des Gebläses

GbWorkArea

Enthält das Menü

GbHeader

Vollständige Positionsstelle im Menü

Info

Info-Pushbutton

Switch

Schalter-Pushbutton

Plan

Plan-Pushbutton

Wie oben werden die graphischen Attribute der Schaltpläne nicht angegeben.

Anhand einer Regel soll aufgezeigt werden, wie dem Schaltsymbol zustandsabhängig unterschiedliche Pixmaps eingepägt werden:

```

RULE (SetzePixmapUndZeigeMenu
        (PRIORITY
        (OBJECT
            ^Wert <prio>)
            ^Name <gebl>
            ^Class Gebraese
            ^priority <prio>)
        (SYMBOLCLICKED
            ^Object <gebl>
            ^Name GbSymbol)
        { <menu> (ATTRIBUTE
            ^My_Obj <gebl>
            ^Name GbWorkArea
            ^Mode { <> Show}})
            (ATTRIBUTE
            ^My_Obj <gebl>
            ^Name Switch)
        -->
        (Call KGT$ChangePixmap <gebl> Switch GbSchalterAus
        (MODIFY <menu> Black White)
            ^Mode Show)
        )

```

Bei Anwahl des Gebläsesymbols wird dynamisch das Menü mit der Pixmap *GbSchalterAus* belegt, falls das Gebläse ausgeschaltet ist. Anschließend wird das Menü angezeigt.

Abschließend soll die Modifikation von einer Menge von Objekten, auch aus unterschiedlichen Objektklassen, mit einer kleinen Menge von Regeln (oft nur einer einzigen) aufgezeigt werden. Angenommen, die Meßstellen seien wie folgende Meßstelle modelliert (hier wird nur der interessierende Teil des Modells angegeben):

```

OBJECT   Name =           T101
          Class =          Meßstelle
          Priority =        8

ATTRIBUTE Name =           ZeigeWert
          My_obj =          T101
          Graphic_Type =    Text
          ...                 ...

```

so kann man durch die Einführung eines Steuerobjektes

```

OBJECT   Name =           SteuerObjekt
          Class =           SteuerClass
          Priority =         8

```


mit dem Attribut

<i>ATTRIBUTE</i>	<i>Name =</i>	<i>SteuerElement</i>
	<i>My_obj =</i>	SteuerObjekt
	<i>Value =</i>	aus

einen Mechanismus einführen, der es erlaubt, eine Menge von Objekten ein- bzw. auszublenden, so wie es bei den unterschiedlichen views des vorangegangenen Abschnitts der Fall war. Dies geschieht bei Anwahl eines views über die Menüleiste durch folgende Regel:

<i>RULE</i>	<i>(ZeigeMeßstellen</i>		
	<i>(OBJECT</i>	<i>^Name</i>	<i><object></i>
		<i>^Class</i>	Meßstelle)
	{ <i><mst></i>	<i>^My_Obj</i>	<i><object></i>
	<i>ATTRIBUTE</i>	<i>^Name</i>	<i>ZeigeWert</i>
		<i>^Mode</i>	<i>{ < > Show } }</i>
	<i>ATTRIBUTE</i>	<i>^My_Obj</i>	<i>SteuerObjekt</i>
		<i>^Name</i>	<i>SteuerElement</i>
		<i>^Value</i>	ein)
	-->		
	<i>(MODIFY</i>	<i><mst></i>	<i>^Mode</i>
	<i>)</i>		Show)

Diese Regel besagt folgendes: für jedes Objekt der Klasse **Meßstellen**, das derzeit nicht auf dem Bildschirm sichtbar ist (Graphikmodus: **< > Show**), wird falls der Wert des Steuerelementes **ein** ist, der Zustand auf **Show** gesetzt. Die Darstellung wird anschließend durch Regeln des Graphik-Laufzeitsystems durchgeführt (detailliertere Information hierüber ist [5.7] zu entnehmen). Diese Regel beeinflusst sämtliche Meßstellen dieser Klasse. Die Beeinflussung einer Menge von Objekten mit OPS5-Regeln ist wie man sieht relativ einfach aufzubauen, da mit OPS5 im Prinzip Mengenoperationen auf der linken Seite der Regel möglich sind.

Es spielt bei diesem Beispiel keine Rolle, in welcher graphischen Hierarchieebene sich die Objekte befinden. Sie müssen nicht den gleichen graphischen Vater haben und können sich irgendwo in der Gesamtapplikation befinden, auch in unterschiedlichen top-level-windows.

Durch Einführung beliebiger anderer Steuerelemente wurden die unterschiedlichen views des voran gegangenen Abschnitts realisiert. Neben der Auswahl der Klassen Temperatur, Druck, u.s.w. wurde auch eine Klassifikation nach Wichtigkeit der Meßstellen vorgenommen (primäre und sekundäre Meßstellen). Dies erreicht man durch Einführung eines zusätzlichen Attributs bei den Meßstellen und eines zusätzlichen Steuerattributs. Die Regel wird dann auf der linken Seite durch folgende zwei Klauseln erweitert:

```
(ATTRIBUTE      ^My_Obj    <object >
                ^Name      Wichtigkeit
                ^Value     <filter >)
```

```
(ATTRIBUTE      ^My_Obj    Filter
                ^Name      FilterElement
                ^Value     <filter >)
```

Wenn dann der aktuelle Filterwert (z.B. primär oder sekundär) mit dem Filterwert der Meßstelle übereinstimmt, wird die Meßstelle angezeigt.

Es ist prinzipiell ohne besondere Schwierigkeiten möglich

- einzelne Objekte,
- Objekte einer Klasse,
- Objekte mehrerer Klassen, die eine gemeinsame Superklasse besitzen,
- Objekte mehrerer Klassen, die keine gemeinsame Superklasse besitzen und
- Objekte aller Klassen

mit einer Regel zu modifizieren. Hierzu müssen sie lediglich einen gemeinsamen Attributnamen besitzen. Die Regeln lassen sich auf alle graphischen Eigenschaften der Objekte anwenden, so z.B. auch auf pop-up-menus und Ähnliches. Die folgende Regel läßt z.B. das überlagerte Fenster (workarea), welches erscheint, wenn man eine Meßstelle anwählt, (bei Anwahl eines buttons im Fenster) das Fenster wieder verschwinden:

```
RULE (BeseitigeWorkArea
      (OBJECT
        { <click>  SYMBOLCLICKED ^Name    <object >
                                     ^Class  Meßstelle)
                                     ^Object  <object >
                                     ^Name    Bezeichner)}
        { <main>  ATTRIBUTE      ^My_Obj  <object >
                                     ^Name    WorkArea
                                     ^Mode    Show)})
      -->
      (MODIFY  <main >          ^Mode    Hide)
      )
```

SYMBOLCLICKED ist dabei eines der Speicherelemente, die vom Graphik-Laufzeitsystem erzeugt werden, um die Wissensbasis über Interaktionen des Benutzers zu unterrichten.

Das nächste Beispiel zeigt die Ausdehnung einer Regel auf mehrere Objektklassen, nämlich auf die Klassen der Temperatur-, Druck- und Feuchtemeßstellen. Dabei spielt es wie schon erwähnt keine Rolle, ob diese Klassen eine gemeinsame Superklasse besitzen.

```

RULE (BeseitigeWorkAreaMehrererKlassen
(OBJECT  ^Name      <object>
        ^Class
        << TempMst DruckMst FeuchteMst >> )
{ <click>  SYMBOLCLICKED ^Object  <object>
                                ^Name   Bezeichner) }
{ <main>  ATTRIBUTE      ^My_Obj  <object>
                                ^Name   WorkArea
                                ^Mode   Show) }
-->
(MODIFY  <main>          ^Mode    Hide)
)

```

Die Regel feuert deshalb auf Objekte mehrere Klassen, weil ein *logisches oder* (bezeichnet in OPS5 durch den Konstrukt << >>) auf der linken Seite der Regel eingeführt wurde.

Regeln, die schlichtweg auf alle Klassen triggern sollen, kann man einfach dadurch implementieren, daß man die Bindung an die Klasse

```

^Class ...

```

wegläßt.

Dieser Abschnitt sollte einen Einblick in die Modellierung der Applikation geben und einige der Möglichkeiten des regel-basierten Programmierens aufzeigen. Die Erfahrungen bei der Implementierung waren zwiespaltig: viele Wünsche lassen sich mit äußerst kompaktem und einfachem Code schnell realisieren. Andererseits ist der Aufbau sehr komplexer, adaptiver Dialoge nach den bisherigen Erfahrungen nicht ganz unproblematisch.

5.2 Eine Benutzerschnittstelle für das Luftgütemeßnetz Kärnten

Für das österreichische Bundesland Kärnten wurde vom Österreichischen Forschungszentrum Seibersdorf (ÖFZS) ein Luftgüteüberwachungssystem entwickelt. Dieses System mißt die Immissionsbelastung des Bundeslandes flächendeckend. Die Arbeiten an verschiedenen Komponenten des Systems sind derzeit noch im Gang. Zur Visualisierung der Meßwerte werden graphische Methoden eingesetzt.

Die AG Computergraphik der Universität Kaiserslautern unterhält wissenschaftliche Kontakte zum ÖFZS. Sie sind aus gemeinsamen Aktivitäten im Arbeitskreis

"Visualisierung von Umweltdaten" der GI-Fachgruppe 4.6.1 (Informatik im Umweltschutz) entstanden.

Im Rahmen der Kontakte wurde von der AG Computergraphik eine alternative Benutzerschnittstelle zur Visualisierung vorgeschlagen. Die Verantwortlichen des ÖFZS zeigten sich an der neuartigen Technologie interessiert und stellten die erforderlichen Unterlagen freundlicherweise zur Verfügung. An dieser Stelle soll hierfür der ausdrückliche Dank an die Verantwortlichen und Mitarbeiter des ÖFZS ergehen, die es durch ihre Aufgeschlossenheit und Diskussionsbereitschaft ermöglichten, die in dieser Arbeit beschriebene Architektur anhand eines konkreten Systems zur Immissionsüberwachung zu erproben.

Wir sahen hinsichtlich der graphischen Benutzeroberfläche vor allem die Möglichkeit, mit dem XAVIA-System die technischen Hintergrundinformationen über Meßstationen und Meßgeräte einzubeziehen. Dies wird bei der derzeit verwendeten Oberfläche nicht getan.

Unter Einbeziehung der technischen Stati der einzelnen Komponenten kann man auch die Visualisierung der Meßstationen vom technischen Zustand abhängig machen und somit auf Fehler der Stationen direkt hinweisen. Auf Anwahl der Meßstation bei einem fehlerhaften Zustand ist die Darstellung einer Zustandsmeldung direkt möglich. Darüberhinaus verwendeten wir als Hintergrund für die Darstellung eine gescannte Karte.

5.2.1 Beispiel zur Oberflächengestaltung

Die folgenden Abbildungen zeigen, wie man die Oberfläche unter Verwendung von direct-manipulation-Techniken aufbauen kann. Es wurde versucht, möglichst viel verfügbare Information auf möglichst einfache Weise zugänglich zu machen.

Abb. 5.28 zeigt die Gesamtübersicht bei Anwahl der Komponente SO₂. Die Balken der Meßstationen sind abhängig vom jeweiligen Meßwert der Komponente, hier sind sie allerdings noch nicht mit echten Werten initialisiert.

Abb. 5.29 zeigt die unterschiedlichen Meldungsrückblicke. Die linke Listbox beinhaltet den globalen Meldungspuffer, die rechte Listbox die derzeit noch anstehenden Meldungen.

Die folgenden Abbildungen sind dem Dialog einer Meßstation gewidmet. Auf Anwahl einer Meßstation im Normalzustand (Abb. 5.30) erscheint die Übersicht. Sie beinhaltet Tag, Datum, Stationsstatus und Kurzbezeichnung der Station (Abb. 5.31). Auf Anwahl des Dialogsymbols werden die einzelnen Komponenten der Station sichtbar (Abb. 5.32). Eine Komponente kann wiederum gewählt werden, was zu der Darstellung einer Liste der letzten Meßwerte führt (Abb. 5.33).

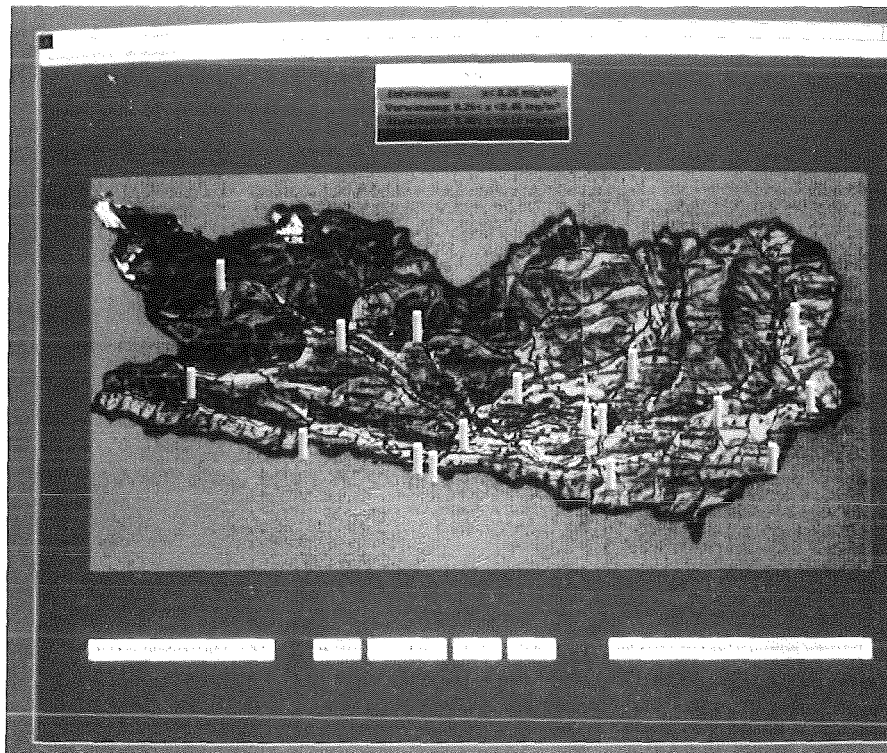


Abb. 5.28: Oberfläche des Luftgütemeßnetzes

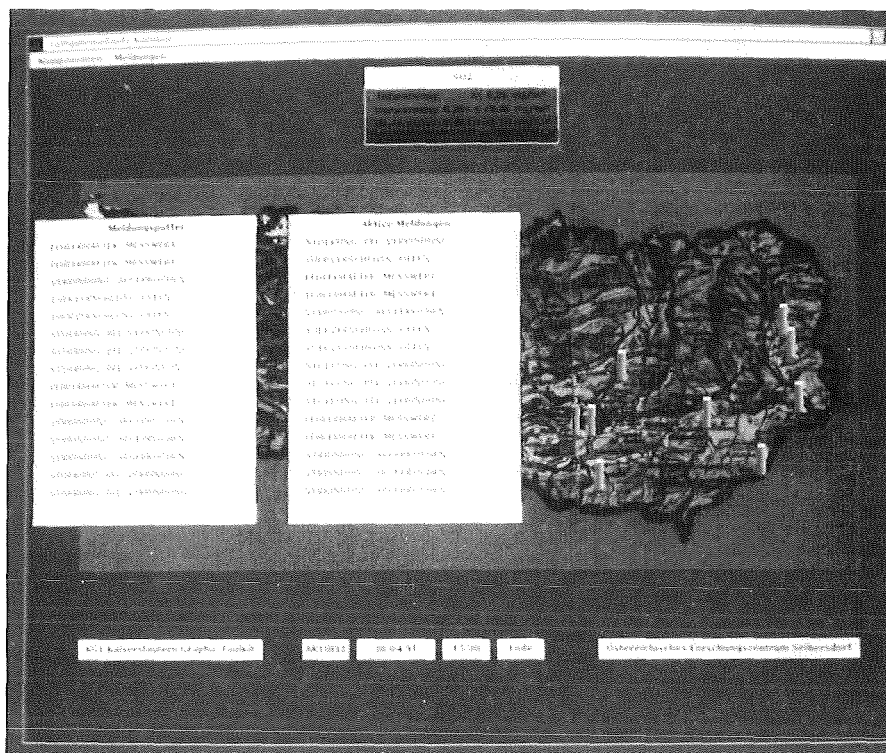


Abb. 5.29: Meldungsrückblicke

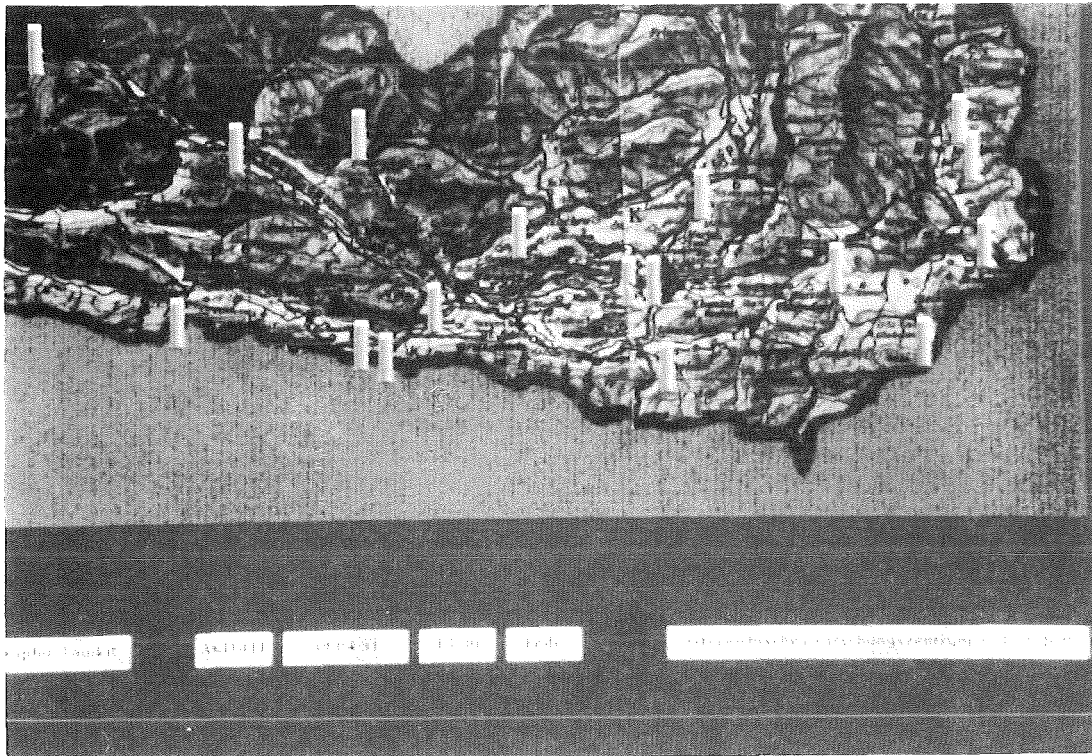


Abb. 5.30: Anwahl einer Meßstation

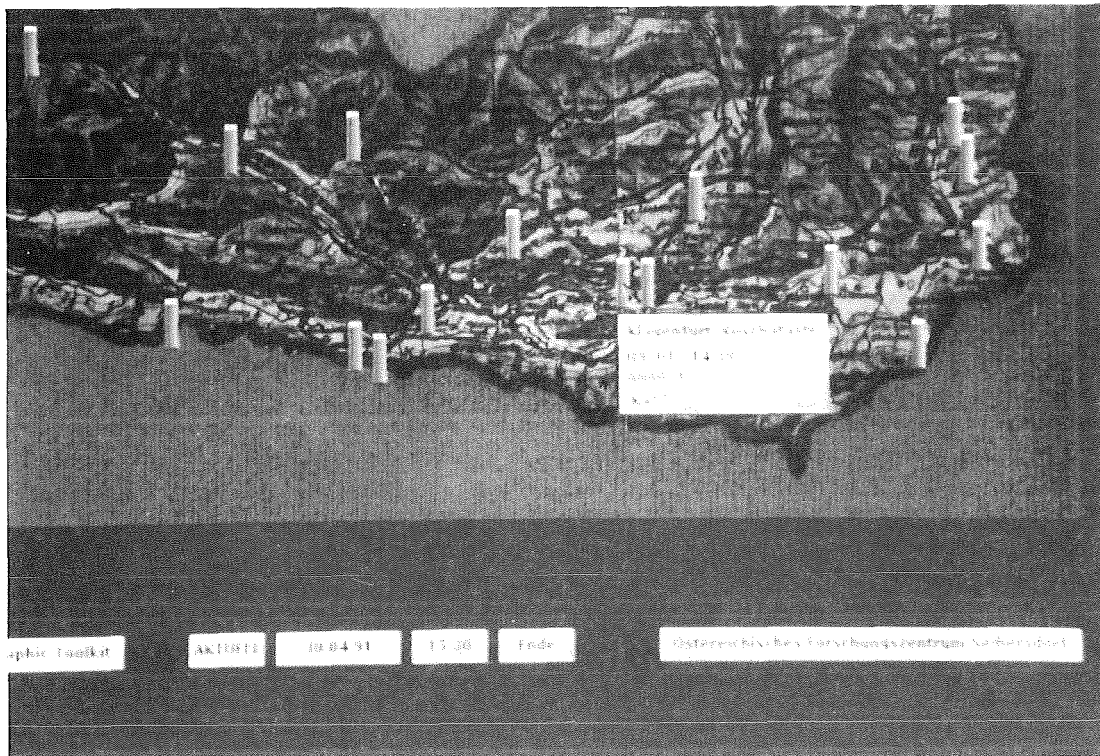


Abb. 5.31: Übersicht über die Meßstation

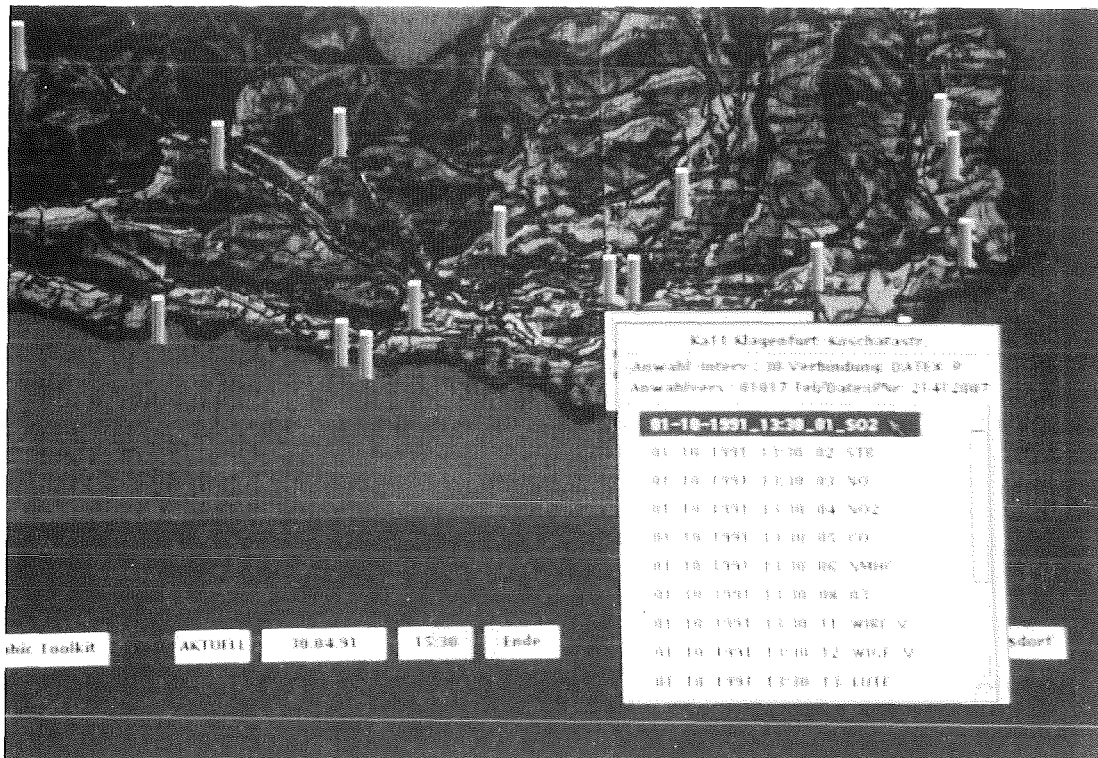


Abb. 5.32: Komponentenübersicht

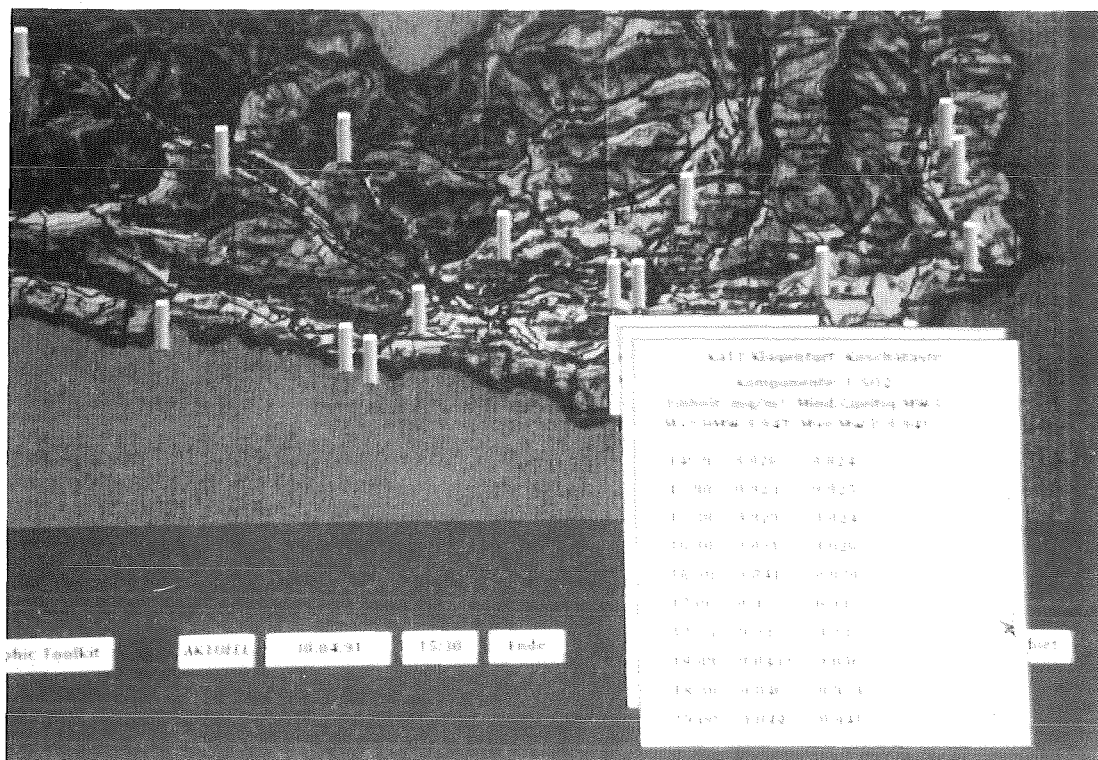


Abb. 5.33: Anzeige der Komponente

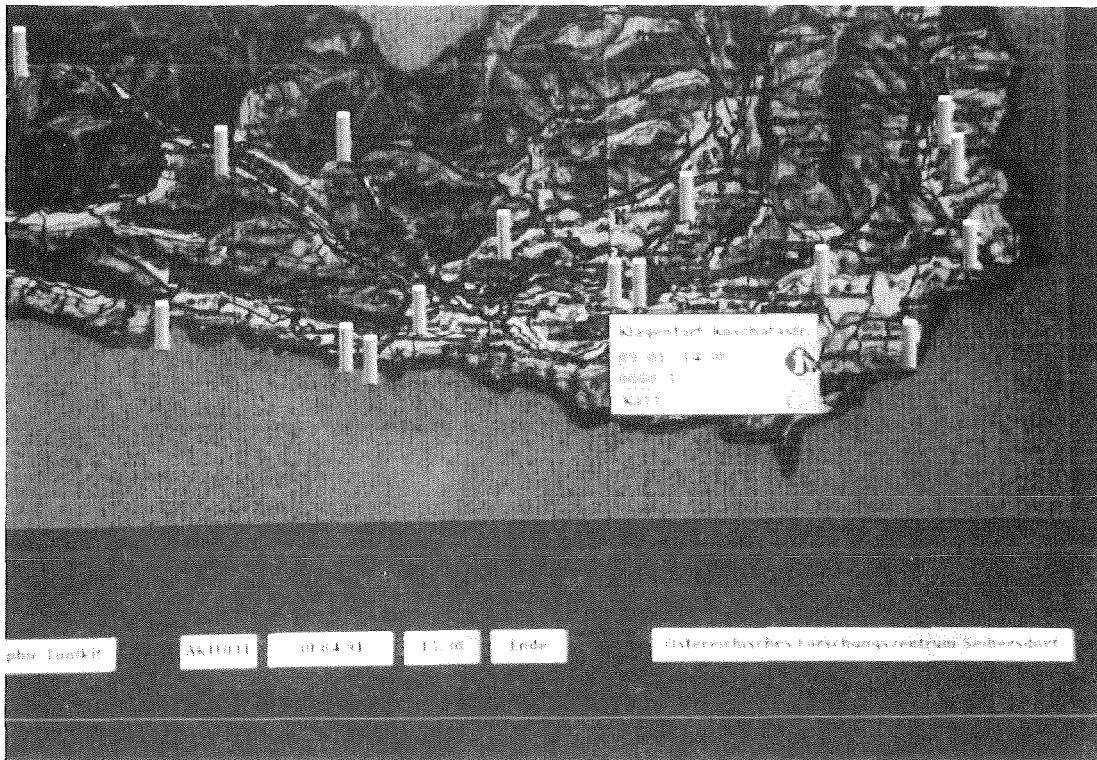


Abb. 5.34: Modifikation des Dialogs

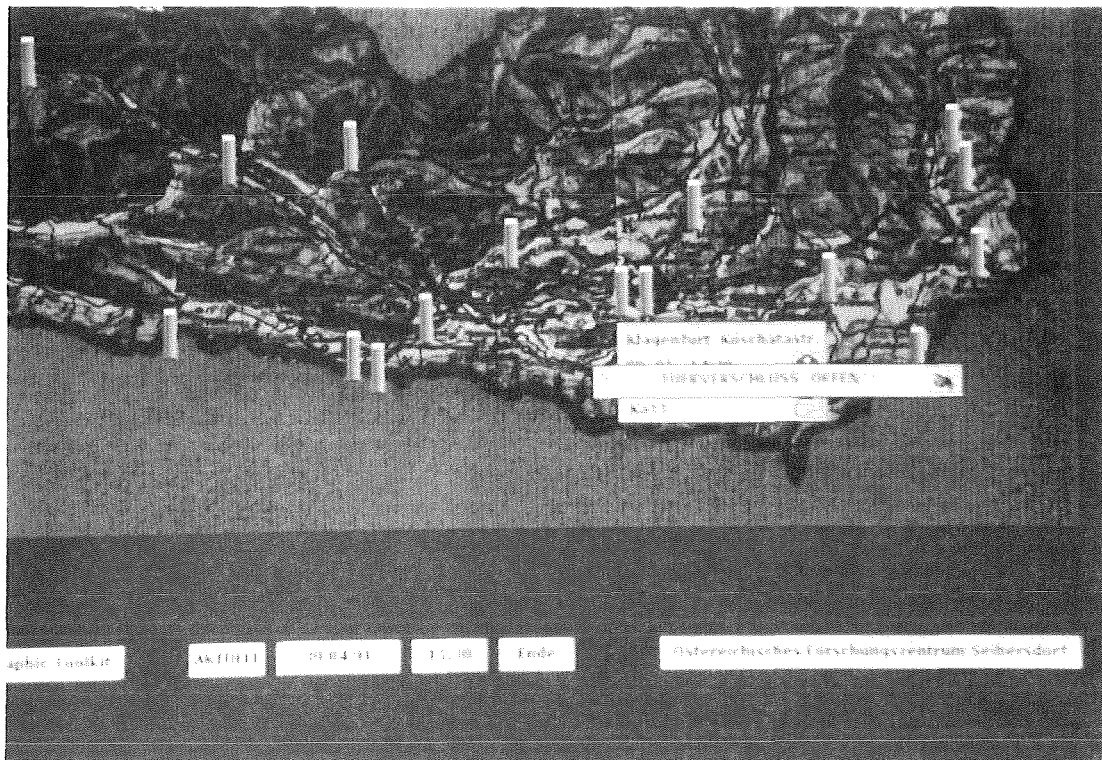


Abb. 5.35: Kontextinformation

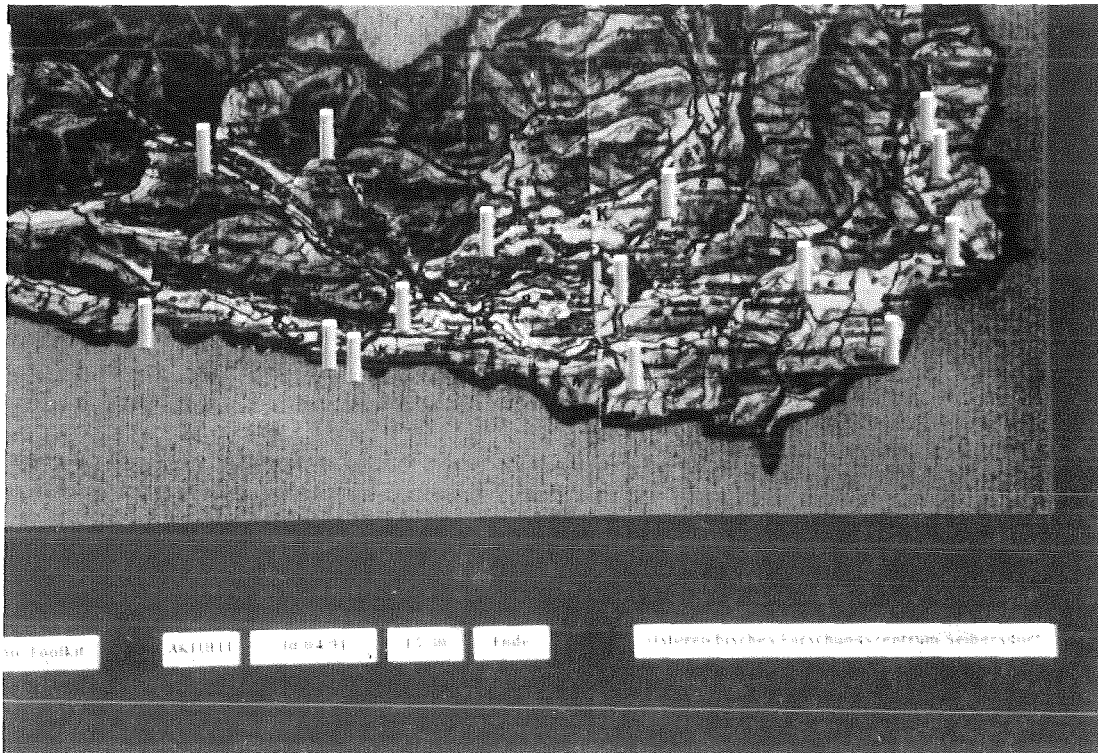


Abb. 5.36: Fehler der Klasse F

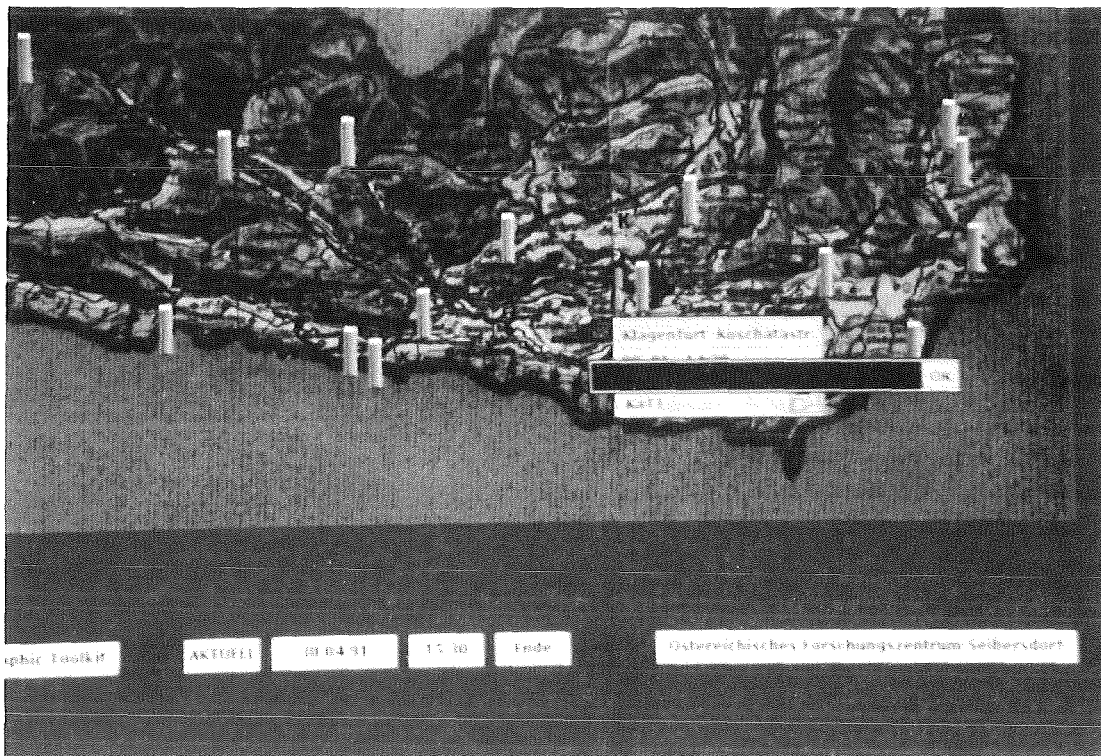


Abb. 5.37: Direkte Störanzeige der Klasse F

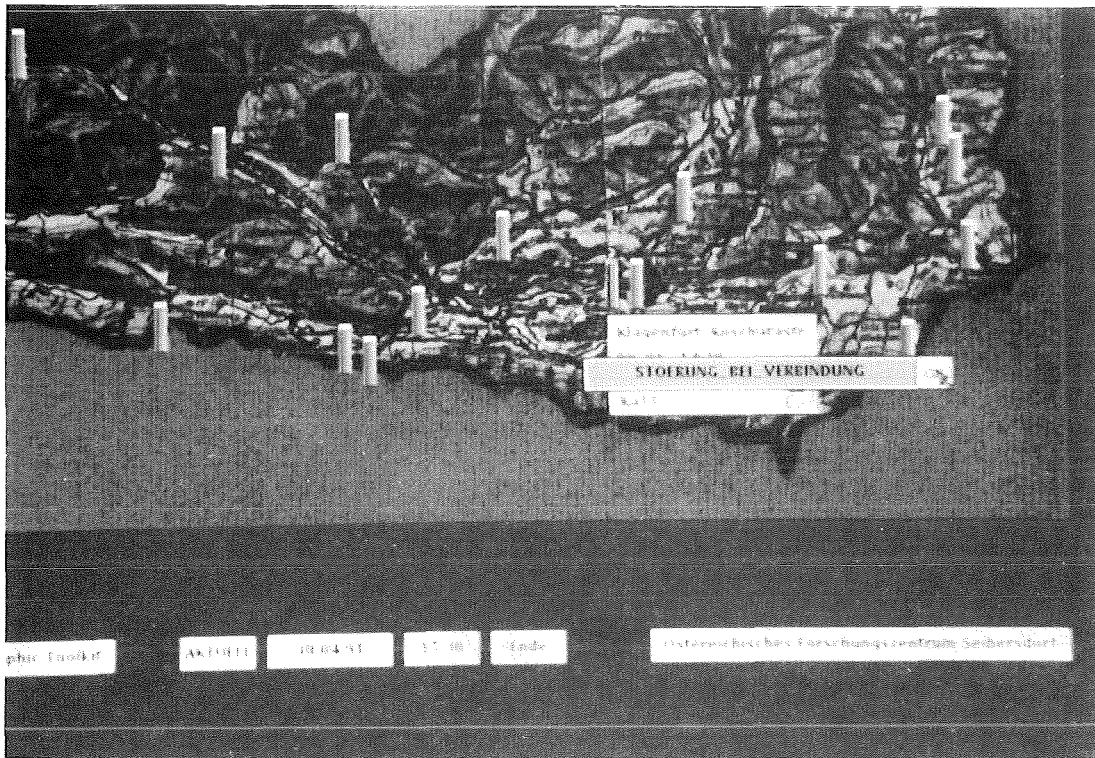


Abb. 5.38: Direkte Störanzeige der Klasse W

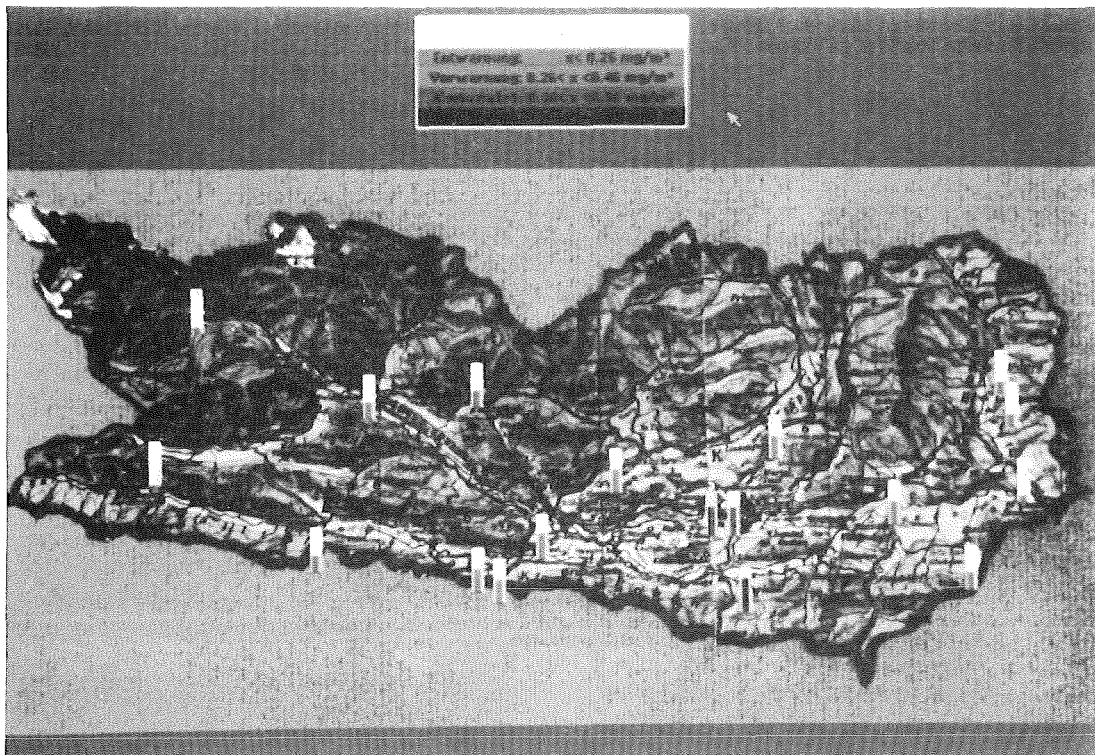


Abb. 5.39: Farb- und Höhenkodierung

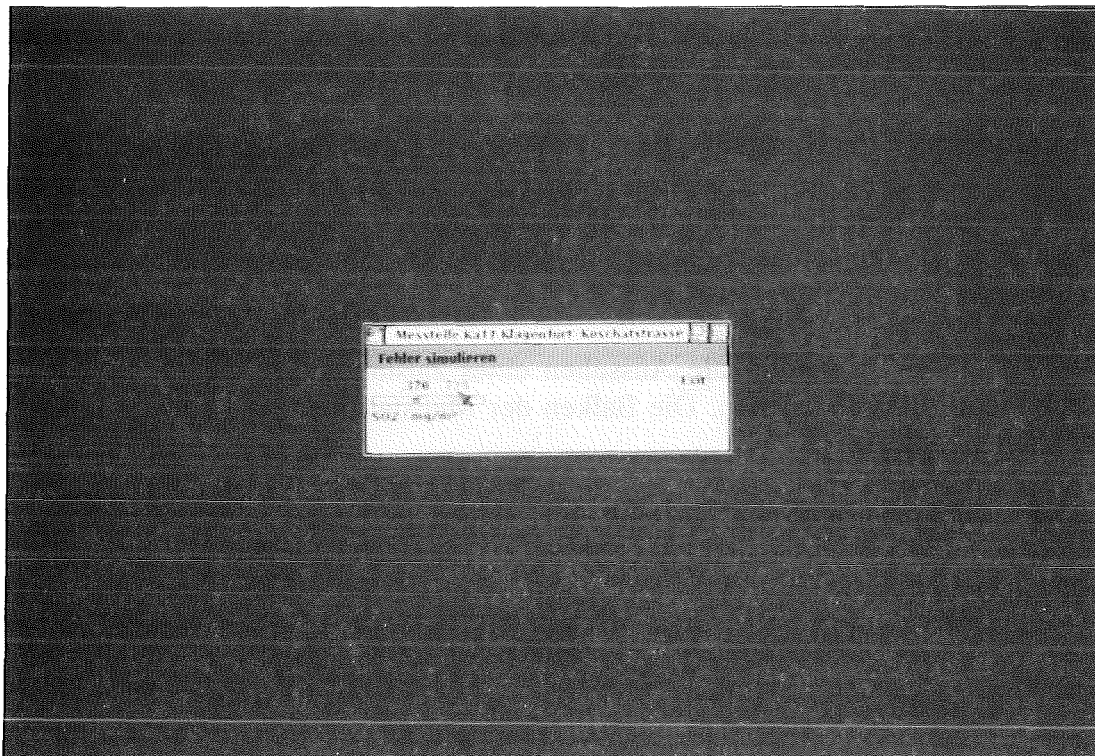


Abb. 5.40: Meßwertprozeß

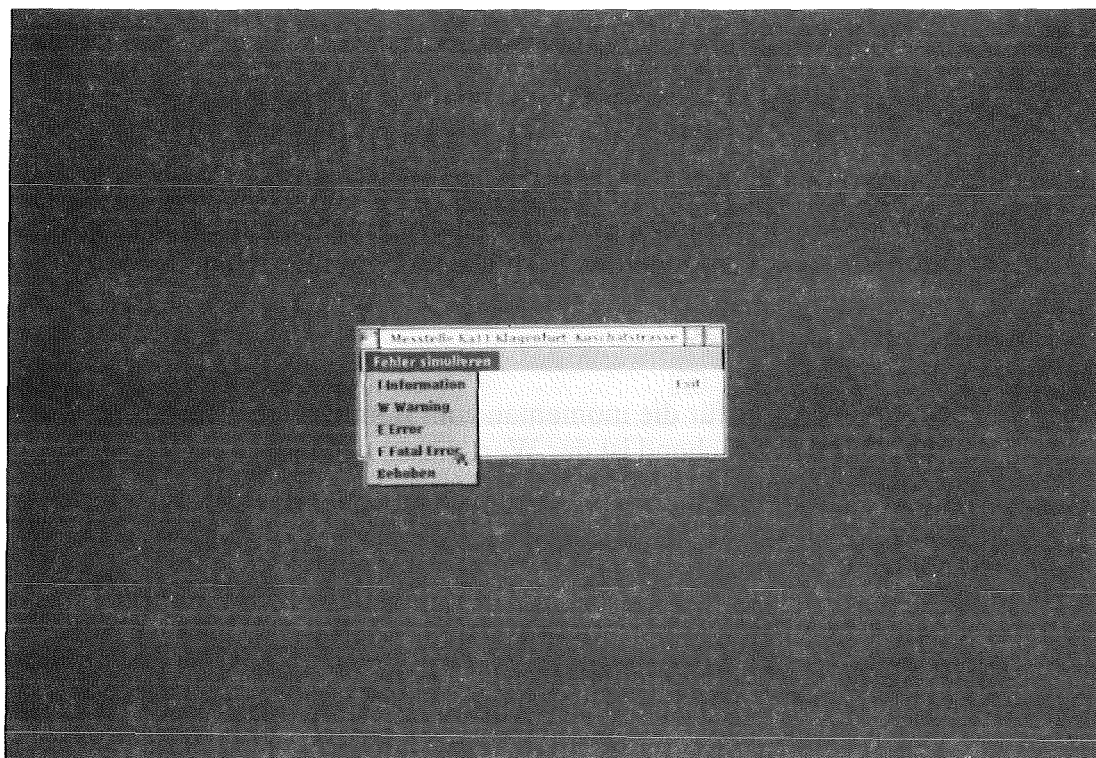


Abb. 5.41: Einstellen von Störmeldungen

Liegt für eine Station eine Information der Klasse **I** vor (Information), so wird der Dialog durch das schon bekannte **(i)** - Symbol erweitert (Abb. 5.34). Hinter diesem Symbol wird die entsprechende Information vorgehalten (Abb. 5.35).

Informationen der Klassen **W**, **E** und **F** (Warning, Error, Fatal Error) modifizieren direkt das Aussehen der Meßstation (Abb. 5.36). Auf Anwahl der Station wird dort ebenfalls direkt die Störmeldung eingeblendet, die auch gleich quittert werden kann. Abb 5.37 zeigt eine solche Meldung für einen Fehler der Klasse **F**, Abb. 5.38 eine Meldung für einen Fehler der Klasse **W**.

In Abb. 5.39 ist das globale Aussehen eines Zustands des Meßnetzes für die Komponente **SO₂** angegeben. Die Balken der Meßstationen sind sowohl farb- als auch höhencodiert.

Die Abbildungen 5.40 und 5.41 schließlich zeigen das Aussehen der nebenläufigen Meßwertprozesse, mit denen für diese Beispiele die Werte der Komponenten eingestellt wurden.

5.2.2 Modellierung

Auch anhand dieser Benutzerschnittstelle soll aufgezeigt werden, wie die einzelnen Einheiten der Interaktion modelliert wurden. Jede Meßstation besitzt eine Anzahl von sog. Komponenten, die jeweils für ein Meßgerät und mithin für einen Meßwert stehen. Die Meßstationen wurden folgendermaßen modelliert:

Die Meßstation besitzt folgende nicht-graphische Attribute:

MstWertnnn

externes Attribut, welches als Empfänger des Meßwertes dient. Mit **< nnn >** wird die Komponente spezifiziert. Folglich muß eine Meßstation für jede Komponente ein solches Attribut besitzen.

LokaleKomp

dient als Anzeiger für die momentan vom Benutzer gewählte Komponente. Dementsprechend ist der Wert dieses Attributs eine Komponentenbezeichnung.

MstMeldung

externes Attribut, dient als Empfänger der Statusmeldung. Der Wert des Attributs ist eine Zeichenkette.

MstStatus

externes Attribut, erhält ein Statuskennzeichen vom Meßwertprozeß (I = Information, W = Warning, E = Error, F = FatalError, O = OK).

Desweiteren besitzt die Meßstation folgende graphische Attribute:

MstRahmen

Hintergrund des Meßstationssymbols

MstBalken

Vordergrund des Meßstationssymbols, verändert in Abhängigkeit des aktuellen Meßwertes Höhe und Farbe.

MstWorkArea

Workarea für die Übersicht (erstes Menü), ist der Vater von MstZeile1 bis MstZeile4, MstStartWatch, MstInfoButton.

MstZeile1, ..., MstZeile4

Enthalten die Informationen der Übersicht.

MstInfoButton

Erscheint nur, wenn das Attribut MstStatus den Wert "I" hat, d.h. wenn eine Informationsmeldung eingetroffen ist. Hat als Hintergrundpixmap das Info-Icon.

MstStartWatch

Icon zum Aufblenden der Übersicht.

StatWorkArea

Workarea für die Stationsübersicht. Ist der Vater von Statzeile1 bis Statzeile3 und KompListbox.

StatZeile1, ..., StatZeile3

Enthalten die Informationen der Stationsübersicht.

KompListbox

Enthält eine Auflistung aller Komponenten der Station.

WerteWorkArea

Workareas für die Komponentenübersicht. Für jede Komponente muß eine solche Workarea existieren. Sie enthält die Attribute KompZeile1 bis KompZeile4 und WerteListBox.

KompZeile1, ..., KompZeile3:

Enthalten die Informationen über die momentan gewählte Komponente.

WerteListBox

Enthält eine Auflistung der Halbstundenmittelwerte (HMW) und 3-Stundenmittelwerte (MW3) für einen Tag.

MstMeldeWorkArea

Zeigt die Meldung an, die das Attribut MstMeldung vom Meßwertprozeß der Meßstation empfangen hat. Enthält die Attribute MstMeldeText und MstMeldeQuittung.

MstMeldeText

Zeigt den Text der Meldung an.

MstMeldeQuittung

Durch Anlicken kann der Benutzer den Empfang einer Meldung quittieren.

Zu einer Meßstation gehören u.A. folgende Regeln

```
RULE (ZeigeMesstelleDirekt
(PRIORITY ^Wert <prio >)
(OBJECT ^Name <messtelle >
^Class Messtelle ^Priority <prio >)
{ <click >
(TEXTCLICKED ^Object <messtelle >
^Name << MstRahmen
MstBalken >> )}
{ <area >
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstWorkArea
^Mode { Show })}
-->
(MODIFY <area > ^Mode Show))
```

Diese Regel triggert mit allen Meßstationen. Durch Anlicken des Rahmens oder des Balkens des Meßstationssymbols wird die zugehörige WorkArea sichtbar gemacht.

Eine analoge Regel existiert für die Abwahl des Menüs nach anlicken des Meßstationsbezeichners. Weitere Regeln, die alle analog der obigen Regel aufgebaut sind, lassen die Anlagenübersicht bzw. die Stationsübersicht erscheinen und verschwinden.

Folgende Regel beeinflusst die Farbe des Balkens einer Meßstation je nach Wert des angezeigten Meßwerts (in diesem Fall für die Komponente SO₂)

```

RULE (SO2Vorwarnung
(PRIORITY ^Wert <prio >)
(OBJECT ^Name <messtelle >
^Class Messtelle
^Priority <prio >)
(1) (ATTRIBUTE ^My_Obj <messtelle >
^Name MstWertSO2
^Value <wert >)
(ATTRIBUTE ^My_Obj <messtelle >
^Name LokaleKomp
^Value SO2)
(2) (ATTRIBUTE ^My_Obj SO2
^Name Vorwarnung
^Value { < <wert > })
(3) (ATTRIBUTE ^My_Obj SO2
^Name Alarmstufe1
^Value { > = <wert > })
(4) (GRAPHICS ^G_Object SO2
^G_Attr ZeigeVorwarnung
^Background <background >)
-->
(CALL KGT$SetBackground
<messtelle > MstBalken <background >)
)

```

In der Klausel (1) wird zunächst der Meßwert an die Variable <wert > gebunden. Anschließend erfolgt in den Klauseln (2) und (3) ein Vergleich, ob der Wert sich zwischen den Schwellwerten für eine Vorwarnung und die Alarmstufe 1 befindet. In Klausel (4) schließlich wird für die graphische Darstellung die Farbe übernommen, die auf der rechten Seite der Regel gesetzt wird. Solche Regeln lassen sich durch geschickte Wahl der Objektklassen auch auf alle Komponenten übertragen.

Die folgenden Regeln modifizieren das Aussehen der Meßstation sowie den Dialog. Falls eine besondere Meldung, z.B. vom Typ "Warning", eintrifft, soll erstens der Hintergrund des Meßstationssymbols sofort die Farbe ändern (hier: Gelb) und zweitens soll nach dem "Anlicken" des Meßstationssymbols die entsprechende Meldung gleichzeitig mit der Übersicht erscheinen.

Jede Meßstation hat zum Empfang einer Meldung zwei Attribute:

MstStatus und

MstMeldung.

MstStatus erhält in diesem Falle das Symbol "W" und **MstMeldung** den Wortlaut der Meldung. Die beiden folgenden Regeln leisten das oben Beschriebene:

```

RULE (ErhalteWMeldung
(PRIORITY ^Wert <prio > )
(OBJECT ^Name <messtelle >
^Class Messtelle
^Priority <prio > )
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstWorkArea
^Mode { < > Show } )
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstStatus
^Value W )
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstMeldung
^Value <text > )
-->
(CALL KGT$SetBackground
<messtelle > MstMeldeText Yellow )
(CALL KGT$SetBackground
<messtelle > MstRahmen Yellow )
(CALL KGT$ListBoxAddItem
Bediener Meldungspuffer <text > )
(CALL KGT$ListBoxAddItem
Bediener AktMeldepuffer <text > )
)

```

Diese Regel modifiziert den Hintergrund einer Meßstation und die Farbe der aufscheinenden Meldung. Darüberhinaus werden die Meldungen noch in den Meldungspuffer geschrieben. Die folgende Regel macht die Meldung sichtbar:


```

RULE (ZeigeWMeldung
(PRIORITY ^Wert <prio >)
(OBJECT ^Name <messtelle >
^Class Messtelle
^Priority <prio >)
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstWorkArea
^Mode Show )

{ <wrkarea >
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstMeldeWorkArea
^Mode { < > Show }) }
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstMeldung
^Value <text > )
(ATTRIBUTE ^My_Obj <messtelle >
^Name MstStatus
^Value W)

-->
(CALL KGT$SetLabel
<messtelle > MstMeldeText <text >)
(MODIFY <wrkarea >
^Mode Show )
)

```

Die Regeln für Meldungen anderer Stufen (Information, Error, Fatal Error) funktionieren analog.

Die vorangegangenen Regeln sollten einen kurzen Einblick in die Modellierung der Interaktion mit der Meßstation geben. Eine einzelne Komponente ist wie folgt modelliert (am Beispiel einer SO₂-Messung):

```

OBJECT   Name =      SO2
         Class =     Komponente
         Priority =   8

```

Dieses Objekt hat folgende nicht-graphische Attribute:

Entwarnung

Der Wert ist der untere Grenzwert für den Bereich Entwarnung.

Vorwarnung

Der Wert ist der untere Grenzwert für den Bereich Vorwarnung.

Alarmstufe1

Der Wert ist der untere Grenzwert für den Bereich Alarmstufe1.

Alarmstufe2

Der Wert ist der untere Grenzwert für den Bereich Alarmstufe2.

und die graphischen Attribute **KompWorkArea**, **Bezeichner**, **ZeigeEntwarnung**, **ZeigeVorwarnung**, **ZeigeAlarmstufe1** und **ZeigeAlarmstufe2**.

Diese Attribute bilden die Legende für die Anzeige der einzelnen Bereiche von SO₂. Die jeweilige Hintergrundfarbe der "Zeige..."-Attribute wird in das Meßstationsymbol übernommen. Wählt der Benutzer im Hauptmenü die Komponente SO₂, so erhält das Attribut **GlobalKomp** sowie alle Attribute **LokalKomp** der Meßstationen den Wert **SO2**. Diese Attribute sind für jede Komponente verschieden.

6. Diskussion

In der vorliegenden Arbeit wurde der Versuch unternommen, Anforderungen und Konzepte für Endbenutzerschnittstellen herauszuarbeiten, welche große, komplexe Datenmengen in komplexen, verteilten Rechnersystemen visualisieren sollen. Hieraus ergab sich insbesondere die Notwendigkeit der integrativen Funktion des Dialogsystems.

Die Architektur stellt ein Gesamtmodell eines Systems zur Visualisierung in komplexen Systemen dar. Sie wurde daher zu einer Synthese unterschiedlicher Techniken:

- Es wird konsequent ein verteilter, offener Ansatz gewählt.
- Objekte werden im Dialogsystem als Ganzes modelliert.
- Die interne Struktur der Objekte entspricht dem mentalen Modell von der Realität.
- Die Modellierung erfolgt unter Verwendung objekt-orientierter Techniken, wobei die Verhaltensweisen der Objekte durch Regeln ausgedrückt werden.
- Die Dialoge können unter Verwendung von Prozeßwissen an spezifische Situationen adaptiert werden.

Das eingeführte Kommunikationssystem ist von seiner Art her neu: auf eine Anfrage können je nach Anforderung beliebig oft Informationen ausgetauscht werden, sofern diese sich ändern.

Die Veränderung einer externen Information wird automatisch ausgetauscht und führt durch eine asynchrone Unterbrechung des Programms zu direkten Auswirkungen in der Benutzerschnittstelle. Hierzu ist keinerlei Programmierung des Datenaustauschs nötig. Man muß lediglich Applikationsregeln schreiben, die auf eine Veränderung des Wertes des Attributs die gewünschten Veränderungen in der Benutzerschnittstelle durchführen.

Das Kommunikationsmodell basiert desweiteren auf Signifikanzkriterien, welche den Datenaustausch steuern. Eine Anfrage an eine bestimmte Information ist also an eine Dienstgüte im Sinne offener Systeme gebunden. Dies führt zu einer erheblichen Reduktion der transportierten Information, insbesondere bei Analogwerten in technischen Anlagen, welche sich oft über längere Zeit nur unwesentlich, oft unterhalb der Meßgenauigkeit, ändern.

Im gesamten System gibt es also nur noch signifikante Ereignisse. Nur solche werden ausgetauscht und weiter verarbeitet.

Das Kommunikationsmodell wird dadurch zu einem mächtigen Instrument, daß der Begriff der externen Information (des externen Attributs) direkt in die Objektbeschreibung des Laufzeitsystems aufgenommen wurde.

Externe Attribute werden nicht durch Programmierung aktualisiert. Durch die Deklaration virtueller Verbindungen zwischen einzelnen Attributen von Objekten im Entwicklungssystem wird die Aktualisierung deklariert, nicht programmiert. Dies führt zu einer erheblichen Einsparung an Programmierfähigkeit beim Aufbau eines verteilten Systems.

Objekte können also aus beliebigen, verteilten Eigenschaften bestehen. Ein Aggregat kann einen externen Meßwert aus einer Prozeßschnittstelle besitzen und einen externen Diagnosewert aus einem Diagnosesystem. Damit kann man die Objekte so modellieren, wie sie dem mentalen Modell des Benutzers entsprechen, gleichgültig welches Attribut eines Objektes in welchem Teilsystem generiert wird.

Das Modell integriert desweiteren die technischen und graphischen Eigenschaften der Objekte in hochwertige, wiederverwendbare Einheiten, welche in einer Entwicklungs-Objektdatenbank gespeichert werden.

Das Verhältnis der Architektur zu UIMS ist zwiegespalten: einerseits kann man mit der Architektur Applikationskomponenten und Darstellungskomponenten durch den verteilten Ansatz wie bei UIMS voneinander trennen. Andererseits wird durch die integrierte Modellierung genau das Gegenteil angestrebt.

Dies hat Vor- und Nachteile. Ein Hauptvorteil ist die Tatsache, daß die Objekte als hochwertige Einheiten wiederverwendbar sind. Der Hauptnachteil mag darin liegen, daß man die Darstellungskomponente nicht wie bei UIMS in einfacher Weise austauschen kann, wenn man eine andere Optik wünscht.

Es ist denkbar, daß man diesen Nachteil dadurch beseitigen kann, daß durch entsprechende Unterklassen unterschiedliche Darstellungskomponenten realisiert werden. Eine abschließende Beurteilung der Vor- und Nachteile gegenüber "klassischen" UIMS-Ansätzen ist zum jetzigen Zeitpunkt noch nicht möglich.

Die entscheidenden Unterschiede zu den in der Literatur bekannten Ansätzen bei UIMS sind

- die Integration eines verteilten Objektsystems in das Dialogsystem
- die Verwendung von KI-Methodiken in Form von vorwärtsverkettenden Regeln zur Erstellung adaptiver Benutzerschnittstellen
- die Möglichkeit, auf einfache Weise auch verteilte Benutzeroberflächen zu erzeugen, die miteinander kommunizieren

Hierdurch wird das Dialogsystem zu einer verteilten Wissensbasis, die alleine durch die eingebrachten Objekte und Regeln lebt. Durch Einbringung neuer Objekte und Objektklassen werden die Limitierungen, die UIMS besitzen können, beseitigt.

Die Erfahrungen beim Aufbau des Systems an sich und beim Aufbau der ersten Anwendungen können folgendermaßen zusammengefaßt werden:

- Der regelbasierte Ansatz war beim Aufbau des verteilten Objektsystems von großem Vorteil. Das gesamte Kommunikationsmodell läßt sich mit weniger als 20 Regeln realisieren (je nach Anzahl der eingeführten Signifikanzkriterien). Hierzu gehört die Verwaltung sämtlicher Anfragen, Meldungen, Verbindungen und die Fehlerbehandlung. Die Regeln drücken vollständig transparent das Kommunikationsverhalten des verteilten Objektsystems aus.
- Ein Hauptproblem ist das der Synchronisation der eingehenden Meldungen mit der Wissensbasis. Nicht jede symbolische Umgebung unterstützt Synchronisationsmechanismen.
- Der Aufwand, der außerhalb der symbolischen Umgebung auftrat (Erstellung der untergeordneten Kommunikationsmechanismen und Anbindung an die Graphik) war höher als der Aufwand innerhalb der symbolischen Umgebung.
- Durch geeignete Wahl der Objektklassen und Regeln ist es möglich, mit wenigen Regeln das gesamte Aussehen einer Anwendung einheitlich zu gestalten.
- Regeln sind ein mächtiges Mittel zur Adaption, zur Dialogfilterung und zur optischen Filterung.
- Die Realisierung sehr komplexer adaptiver Dialoge ist - nach dem jetzigen Erkenntnisstand - problematisch.
- Das Laufzeitverhalten der Benutzeroberfläche ist vollständig zufriedenstellend.
- Das Laden einer komplexen Anwendung ist durch das Erzeugen einer Vielzahl von OPS5-Objekten und deren graphischer Merkmale (Objekte im Sinne von DECwindows/Intrinsics) langsam. Es kommen Ladezeiten von über einer Minute vor.
- Die Anwendungen benötigen große Mengen von dynamischem Speicher.

Abschließend sei angemerkt, daß der Entwurf von Anwendungen ohne eine komfortable Entwicklungsumgebung nicht praktikabel ist. Die derzeit noch laufenden Arbeiten widmen sich unter anderem einer solchen Entwicklungsumgebung.

Mittels dieser wird es in einfacher Art möglich sein, zunächst das Aussehen der Objektklassen und Objekte zu modellieren. Virtuelle Verbindungen zwischen Objekten werden einfach zwischen Objekten deklariert. Hierzu werden geringe Kenntnisse benötigt.

Bei der Modellierung der Verhaltensweisen der Objekte - also bei der Gestaltung der Regeln - kommt man allerdings ohne ein tieferes Verständnis des Gesamtkonzeptes und der symbolischen Verarbeitung nicht aus. Insofern ist das System in seiner derzeitigen Version nur den Spezialisten und nicht den Endanwendern zugänglich.

Es ist denkbar, daß man durch entsprechende graphische Werkzeuge diese Situation verbessern kann. Allerdings lassen die Erfahrungen des Projekts die Illusion mancher DV-Fachleute irrig erscheinen, die für die nächsten Jahre Umgebungen kommen sehen, die es gestatten, daß Endanwender ohne DV-Kenntnisse Anwendungen aufbauen können.

Zumindest für die Visualisierung von komplexen Systemen in komplexen DV-Systemen kann hiervon vorläufig nicht ausgegangen werden.

Literaturverzeichnis

Literatur zu Kapitel 1

- [1.1] IEEE, First IEEE Conference on Visualization, Visualization '90, San Francisco, IEEE Computer Society Press, 1990
- [1.2] Becker R.A. et. al., Dynamic Graphics for Network Visualization, in [1.1], pg. 93-96
- [1.3] Hanson A.J., Heng P.A., Kaplan B.C., Techniques for Visualizing Fermat's Last Theorem: A Case Study, in [1.1], pg. 97-106
- [1.4] Inselbert A., Dimsdale B., Parallel Coordinates: A Tool for Visualizing Multi-dimensional Geometry, in [1.1], pg. 361-378
- [1.5] Holfter B., Scherelis G., Einsatzmöglichkeiten eines geographischen Informationssystems bei der Umweltverträglichkeitsprüfung, in [1.7], pg. 355-362
- [1.6] Föllinger O., Regelungstechnik: Einführung in die Methoden und ihre Anwendungen, Hüthig, 1990
- [1.7] Pillmann W., Jaeschke A. (eds.), Informatik für den Umweltschutz, 5. Symposium, Wien, 1990, Informatik-Fachberichte 256, Springer, 1990

Literatur zu Kapitel 2

- [2.1] Jaeschke A., Page B. (eds.), Informatikanwendungen im Umweltbereich, 1. Symposium, Karlsruhe, 1986, KfK-Bericht KfK 4223, Kernforschungszentrum Karlsruhe, 1987
- [2.2] Jaeschke A., Page B. (eds.), Informatikanwendungen im Umweltbereich, 2. Symposium, Karlsruhe, 1987, Informatik-Fachberichte 170, Springer, 1987
- [2.3] Fachgespräch Informatikanwendungen im Umweltbereich, GI-Jahrestagung 1988, in Valk R. (ed.), GI - 18. Jahrestagung, Hamburg, 1988, Informatik-Fachberichte 187, Springer, 1988
- [2.4] Jaeschke A., Geiger W., Page B. (eds.), Informatik im Umweltschutz, 4. Symposium, Karlsruhe, 1989, Informatik-Fachberichte 228, Springer, 1989
- [2.5] Pillmann W., Jaeschke A. (eds.), Informatik für den Umweltschutz, 5. Symposium, Wien, 1990, Informatik-Fachberichte 256, Springer, 1990

- [2.6] Denzer R., Hagen H., Kutschke K.H., Visualisierung von Umweltdaten, Workshop, Rostock, 1990, Informatik-Fachberichte 274, Springer, 1991
- [2.7] Denzer R., User Interfaces für die Visualisierung von Umweltdaten - Anforderungen und Architektur, in [2.5], pg. 847-854
- [2.8] Denzer R., User Interface Management in Distributed Systems, in [2.6], pg. 83-97
- [2.9] Pfaff G. (ed.), User Interface Management Systems, Workshop, Seeheim, 1983, Springer, 1985
- [2.10] Cato J., Issues in Object Oriented Design of the User Interface, in: OOPS-30, "The next step", Meeting of the British Computer Society OOPS group on object-oriented programming, 1990, Strand Palace Hotel, London
- [2.11] Sachs P., ESCORT - an Expert System for Complex Operations in Real Time, Alvey Workshop on Deep Knowledge, IEE, London, 1985
- [2.12] Alty J.L., Mullins J., Dialogue Specification in the GRADIENT Dialogue System, in: Sutcliffe A., MacAulay L., People and Computers V, 5. Conference of the British Computer Society HCI Specialist Group, Nottingham, 1989, pg. 151-168, Cambridge University Press, 1989
- [2.13] Denzer R., Informatikeinsatz im prozeßnahen Bereich an einer Pilotanlage zur schadstoffarmen Müllverbrennung, in [2.4], pg. 329-337
- [2.14] Föllinger O., Lineare Abtastsysteme, 2. Auflage, Oldenbourg-Verlag, 1982
- [2.15] Washington R., Hayes-Roth B., Input Data Management in Real-Time AI Systems, in: Sridharan, H.S. (ed.), IJCAI '89, 11. International Joint Conferences on Artificial Intelligence, Detroit, 1989, Morgan Kaufmann Publishers, 1989
- [2.16] Pflügl M., Interprocess Communication in MARS, in: Kühn P.J. (ed.), Kommunikation in verteilten Systemen, ITG/GI-Fachtagung, Stuttgart 1989, Informatik-Fachberichte 205, Springer, 1989
- [2.17] Schimak G., Grundlagen, Komponenten und Konzeption eines computerunterstützten Umweltinformationssystems für das Bundesland Oberösterreich - UIS/OÖ -, Diplomarbeit, Universität Linz, 1990
- [2.18] Schimak G., Denzer R., Komplexe Inhalte eines Umweltinformationssystems, in [2.6], pg. 9-21

- [2.19] Dienes A., DIM, Daten- und Informationssystem für den Minister für Umwelt, Raumordnung und Landwirtschaft des Landes Nordrhein-Westfalen (MURL), in [2.4], pg. 203-208
- [2.20] Lessing H., Umweltinformationssysteme - Anforderungen und Möglichkeiten am Beispiel Niedersachsens, in [2.4], pg. 209-218
- [2.21] Kremers H. et. al., Arbeitskreis "Umweltdatenbanken" - Ziele und erste Ergebnisse, in [2.5], pg. 1-16
- [2.22] Hagen H., van Lengen R., Schreiber Th., Visualisierung von Umweltdaten, in [2.5], pg. 799-807
- [2.23] Groß M., Ein integriertes Visualisierungs- und Simulationssystem für den Umweltbereich, in [2.5], pg. 808-817
- [2.24] Weidemann R., Geiger W., XUMA - ein Assistent für die Beurteilung von Atlanten, in [2.4], pg. 385-394
- [2.25] Dollinger F., Strobl J. (eds.), Angewandte Geographische Informationstechnologie, Beiträge zum GIS-Symposium, Salzburg, 1989, Selbstverlag des Inst. für Geographie der Univ. Salzburg, 1989
- [2.26] Starke H., Automatische Meßnetze in Bayern, in [2.4], pg. 300-308
- [2.27] Grützner R., Simulation im Umweltschutz - Anwendung, Anforderung, Visualisierung, in: [2.6], pg. 29-37
- [2.28] Fedra K., Interactive Environmental Software: Integration, Simulation and Visualization, in [2.5], pg. 733-744
- [2.29] Pillmann W., Zobl Z., Entwicklungstendenzen in Waldökosystemen - Ergebnisse der Scannerbildanalyse auf einem Parallelrechner, in [2.5], pg. 286-295

Literatur zu Kapitel 3

- [3.1] Nehmer J., Softwaretechnik für verteilte Systeme, Springer, 1985
- [3.2] Wybraniec D., Multicast-Kommunikation in verteilten Systemen, Informatik-Fachberichte 242, Springer, 1990
- [3.3] Jablonski S., Datenverwaltung in verteilten Systemen, Informatik-Fachberichte 233, Springer, 1990

- [3.4] Görgen K. et. al., Grundlagen der Kommunikationstechnologie, Springer, 1985
- [3.5] Henshall J., Shaw S., OSI Explained, End-to-end Computer Communication Standards, John Wiley and Sons, 1988
- [3.6] Kühn P.J. (ed.), Kommunikation in verteilten Systemen, ITG/GI-Fachtagung, Stuttgart 1989, Informatik-Fachberichte 205, Springer, 1989
- [3.7] Gerner N., Spaniol O. (eds.), Kommunikation in Verteilten Systemen, GI/NTG-Fachtagung, Aachen 1987, Informatik-Fachberichte 130, Springer, 1987
- [3.8] Heger D., Krüger G., Spaniol O., Zorn W. (eds.), Kommunikation in Verteilten Systemen I, GI/NTG-Fachtagung, Karlsruhe 1985, Informatik-Fachberichte 95, Springer, 1985
- [3.9] Heger D., Krüger G., Spaniol O., Zorn W. (eds.), Kommunikation in Verteilten Systemen II, GI/NTG-Fachtagung, Karlsruhe 1985, Informatik-Fachberichte 111, Springer, 1985
- [3.10] Schröder-Preikschat W., Zimmer W. (eds.), Progress in Distributed Operating Systems and Distributed Systems Management, European Workshop, Berlin 1989, Lecture Notes in Computer Science 433, Springer, 1989
- [3.11] Nehmer J. (ed.), Experiences with Distributed Systems, International Workshop, Kaiserslautern 1987, Lecture Notes in Computer Science 309, Springer, 1987
- [3.12] Paul M., Siegert H.J. (eds.), Distributed Systems, An Advanced Course, München 1985, Lecture Notes in Computer Science 190, Springer, 1985
- [3.13] IEEE, The 6th International Conference on Distributed Computing Systems, Cambridge MA 1986, IEEE Computer Society Press, 1986
- [3.14] Burkhart H. (ed.), CONPAR 90, VAPP IV, Joint International Conference on Vector and Parallel Processing, Zurich 1990, Lecture Notes in Computer Science 457, Springer, 1990
- [3.15] Müller G., Blanc R.P. (eds.), Networking in Open Systems, International Seminar, Oberlech 1986, Lecture Notes in Computer Science 248, Springer, 1986
- [3.16] Hutchison D., Mariani J., Shepherd D., Local Area Networks: An Advanced Course, 1983, Lecture Notes in Computer Science 184, Springer, 1985

- [3.17] Lampson B.W., Paul M., Siegart H.J., Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer
- [3.18] van Rossum G., AIL - a Class-Oriented RPC Stub Generator for Amoeba, in [3.10], pg. 13-21
- [3.19] Horn C., Is Object Orientation a Good Thing for Distributed Systems?, in [3.10], pg. 60-74
- [3.20] Tschammer V. et. al., OAI - Concepts for Open Systems Cooperation, in [3.10], pg. 174-192
- [3.21] van Renesse R., Position paper, in [3.10], pg. 193-194
- [3.22] Nehmer J., Position paper, in [3.10], pg. 201
- [3.23] Almes G.T., The Eden System: A Technical Review, in: IEEE Transactions on Software Engineering, Vol. SE-11, No. 1, Jan. 1985, pg. 43-58
- [3.24] Kühn P. J., Kommunikation in verteilten Systemen - Einführung und Überblick, in [3.6], pg. 1-4
- [3.25] Schill A. et. al., Using the Object Paradigm for Distributed Application Development, in [3.6], pg. 84-98
- [3.26] Pflügl M. et. al., Interprocess Communication in MARS, in [3.6], pg. 189-202
- [3.27] Bever M., Zimmermann M., Data Translation in Heterogeneous Computer Networks, in [3.6], pg. 430-447
- [3.28] Schulz K.J., Objekt-Orientierte Konstruktion eines Portablen Kommunikations-Transport-Systems für Unix Netzwerk Applikationen, in [3.6], pg. 448-462
- [3.29] Ness A.J., Reim F., Adaptive Design and Management of Distributed Information Systems, in [3.6], pg. 804-817
- [3.30] Brössler P. et. al., Attributierte Kanäle: Ein neues Kommunikationskonzept, in [3.7], pg. 141-153
- [3.31] Härtig H. et. al., Distribution and Recovery in the BirliX Operating System, in [3.7], pg. 190-210

- [3.32] Zhou S., Zicari R., Object Management in Local Distributed Systems, in [3.7], pg. 202-215
- [3.33] Nehmer J., On the Adequate Support of Communication Interfaces in Distributed Systems, in [3.11], pg. 1-18
- [3.34] Mullender S. J., Process Management in a Distributed Operating System, in [3.11], pg. 38-51
- [3.35] Zahorjan J. et. al., Accomodating Heterogeneity, in [3.11], pg. 89-103
- [3.36] Schlichting R. D. et. al., Observations on Building Distributed Languages and Systems, in [3.11], pg. 271-291
- [3.37] Black A. et.al., Distribution and Abstract Types in Emerald, in: IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, Jan. 1987, pg. 65-75
- [3.38] Effelsberg W., Meurer H.W., Müller G. (eds.), Kommunikation in verteilten Systemen, ITG/GI-Fachtagung, Mannheim 1990, Informatik-Fachberichte 267, Springer, 1989
- [3.39] The 10th International Conference on Distributed Computing Systems, Proceedings, Paris, 1990, IEEE Computer Society Press, 1990
- [3.40] The 9th International Conference on Distributed Computing Systems, Proceedings, 1989, IEEE Computer Society Press, 1989
- [3.41] The 8th International Conference on Distributed Computing Systems, Proceedings, San Jose, 1988, IEEE Computer Society Press, 1988
- [3.42] The 7th International Conference on Distributed Computing Systems, Proceedings, Berlin, 1987, IEEE Computer Society Press, 1987
- [3.43] The 6th International Conference on Distributed Computing Systems, Proceedings, Cambridge, Massachusetts, 1986, IEEE Computer Society Press, 1986
- [3.44] The 5th International Conference on Distributed Computing Systems, Proceedings, Denver, 1985, IEEE Computer Society Press, 1985
- [3.45] Birell A.D., Secure Communication Using Remote Procedure Calls, ACM Transactions on Computer Systems, Vol. 3, No. 1, Feb. 1985, pg. 1-14
- [3.46] Bloch J. J., The Camelot Library: A C Language Extension for Programming a General Purpose Distributed Transaction System, in [3.40]

- [3.47] Walker E.F., Floyd R., Neves P., Asynchronous Remote Operation Execution In Distributed Systems, in [3.39], pg. 253-259
- [3.48] Barth G., Welsch C., Objektorientierte Programmierung, Informationstechnik it 30 (1988) 6, pg. 404-421
- [3.49] Pinson J.L., Wiener R.S., Applications of Object-Oriented Programming, Addison-Wesley, 1990
- [3.50] Winder R., Developing C++ Software, Wiley & Sons, 1991
- [3.51] Jones M., Rashid R., Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems, Proceedings OOPSLA '86, pg. 67-77, 1986
- [3.52] Nye A., O'Reilly T., The X Window System, Vol. 0-7, OSF/Motif Edition, O'Reilly & Associates, 1990
- [3.53] Wisskirchen P., Object-Oriented Graphics, Springer, 1990
- [3.54] Vandoni C.E., Duce D.A. (eds.), Eurographics '90, Proceedings, Montreux, North-Holland, 1990
- [3.55] Hansmann W., Hopgood F.R.A., Strasser W. (eds.), Eurographics '89, Proceedings, Hamburg, North-Holland, 1989
- [3.56] Requicha A.A.G. (ed.), Eurographics '86, Proceedings, Lisbon, North-Holland, 1987
- [3.57] Pierra G., An Object Oriented Approach To Ensure Portability Of CAD Standard Part Libraries, in [3.54], pg. 205-214
- [3.58] Fellner W.D., Kappe F., EDEN - An Editor Environment For Object-Oriented Graphics Editing, in [3.54], pg. 425-437
- [3.59] Faconti G.P., Paterno F., An Approach To The Formal Specification Of The Components Of An Interaction, in [3.54], pg. 481-494
- [3.60] Hill R.D., Herrmann M., The Structure of Tube, A Tool for Implementing Advanced User Interfaces, in [3.55], pg. 15-25
- [3.61] Breen D.E., Kühn V., Message-Based Object-Oriented Interaction Model, in [3.55], pg. 489-503

- [3.62] Prospero M.J., Messina L.A., Towards The Construction Of Graphical Interfaces On The Basis Of Geometric Models, in [3.56], pg. 173-183
- [3.63] Soloway E., Frye D., Sheppard S. (eds.), CHI '88, Proceedings, Human Factors in Computing Systems, Washington, 1988, Special Issue of the SIG-CHI Bulletin, ACM, 1988
- [3.64] Foley J. et. al., A Knowledge-Based User Interface Management System, in [3.63], pg. 67-72
- [3.65] Gargan R.A. et. al., Multimodal Response Planning: An Adaptive Rule Based Approach, in [3.63], pg. 229-235
- [3.66] Tyler S.W., SAUCI: A Knowledge-Based Interface Architecture, in [3.63], pg. 235-240
- [3.67] Pfaff G.E. (ed.), User Interface Management Systems, Springer-Verlag, 1985
- [3.68] Diaper D. (ed.), INTERACT '90, Proceedings, North-Holland, 1990
- [3.69] Roudaud B. et. al., SCENARIOO: A new generation UIMS, in [3.68], pg. 607-612
- [3.70] Rich E., KI-Einführung und Anwendungen, McGraw-Hill Book Company, Hamburg, 1988
- [3.71] Stoyan H., Programmiermethoden der Künstlichen Intelligenz, Studienreihe Informatik, Springer-Verlag, Heidelberg, 1988
- [3.72] Schnupp P., Nguyen Huu C.T., Expertensystem-Praktikum, Springer-Verlag, Heidelberg, 1987
- [3.73] Scheffe P., Künstliche Intelligenz, Überblick und Grundlagen, Reihe Informatik, Band 53, BI-Wissenschaftsverlag, Mannheim, 1987
- [3.74] Puppe F., Diagnostisches Problemlösen mit Expertensystemen, Informatik-Fachberichte 148, Springer-Verlag, Heidelberg, 1987
- [3.75] Schnupp P., PROLOG - Einführung in die Programmierpraxis, Hanser-Verlag, München, 1987
- [3.76] Stock M., AI Theory and Applications in Process Control and Management, McGraw-Hill Book Company, New York, 1989

- [3.77] Tzafestas S.G., Knowledge-Based System Diagnosis, Supervision and Control, Plenum Press, New York, 1989
- [3.78] Früchtenicht H.W. (Hrsg.), Technische Expertensysteme: Wissensrepräsentation und Schlußfolgerungsverfahren, Oldenbourg-Verlag, München, 1988
- [3.79] IEEE Conference on Artificial Intelligence Applications, Proceedings, Kissikkee, Florida, 1987
- [3.80] Proceedings of the 4. IEEE Conference on Artificial Intelligence Applications, San Diego, 1988
- [3.81] Proceedings of the 5. IEEE Conference on Artificial Intelligence Applications, Miami, 1989
- [3.82] Expert Systems 1985, Proceedings of the Technical Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, 1985
- [3.83] Applications of Artificial Intelligence in Engineering Problems, 1. International Conference, Proceedings, Southampton, 1986, Sprinegr-Verlag
- [3.84] Proceedings of the IASTED International Symposium on Applied Informatics, Grindelwald, 1987
- [3.85] European Conference on Artificial Intelligence, Proceedings, München, 1988
- [3.86] Proceedings of the 4. International Expert Systems Conference, Oxford, 1988
- [3.87] Knowledge Based Concepts and Artificial Intelligence Applications to Guidance and Control, Proceedings, Ottawa, 1987, AGARD Lectures Series 155
- [3.88] IEE Colloquium on The Use of Expert Systems in Control Engineering, London, 1987, IEE Digest No. 1987/27
- [3.89] IEE Colloquium on Expert Systems in Process Control, London, 1988, IEE
- [3.90] Chantler M.J., Real-Time Aspects of Expert Systems in Process Control, London, 1988, in [3.89], pg. 4/1-4/8
- [3.91] Doherty B.S., Sorin A.J., Knowledge Based Real-Time Learning Control System, in [3.84], pg. 18-21
- [3.92] Jones A.H., Porter B., Expert Systems for tuning PID controllers, in [3.89], pg. 1/1-1/4

- [3.93] McCluskey E.G., Thompson S., A self-tuning PID controller incorporating expert systems ideas, in [3.89], pg. 2/1-2/3
- [3.94] Cram R.S., Clarke B.R., Expert System Monitoring and Control of a Polymer Plant, in [3.89], pg. 6/1-6/6
- [3.95] Proceedings of the 1987 American Control Conference, Minneapolis, 1987
- [3.96] Handelman D.A., An Architecture for Real-Time Rule-Based Control, in [3.95], pg. 1636-1642
- [3.97] Rao M., Tsai J.J.-P., Jiang T.S., An Intelligent Decision Maker for Optimal Control, in Applied Artificial Intelligence, 1988, 2, pg. 285-305
- [3.98] Ambrosio D.A., AI in the Power Plants: PERF-EXS, a PERFORMANCE diagnostics EXPert System, in: Proc. of the 4. Conference on Artificial Intelligence Applications, San Diego, 1988, pg. 11-17
- [3.99] Leith D., Deans N.D., Thomson W.T., A Real-Time Expert System for Machine Condition Monitoring, in: International Conference on Software Engineering, Cirencester, UK, 1987, pg. 113-119
- [3.100] Chesler L. et. al., EXFAULT: A Generalized Expert Fault Diagnosis System, in: Proceedings of the 3. Annual Artificial Intelligence and Advanced Computer Technology Conference, Long Beach, 1987, pg. 573-582
- [3.101] Krishnamurthy M., Efstatiou H.J., An Intelligent Knowledge Based System for Diagnosis Based on Qualitative Reasoning, in 3.89, pg. 2/1-2/5
- [3.102] Gidwani K. K., Dalton W.S., Innovative Knowledge Engineering for Real-Time Expert Systems, in 4. IFAC/IFIP-Symposium, Graz, 1986
- [3.103] Paterson A., Sachs P., Turner M., ESCORT: The Application of Causal Knowledge to Real-Time Process Control, in: Expert Systems 1985, Cambridge, 1985, pg. 79-88
- [3.104] Paterson A.m Sachs P., Causal Reasoning in a Real-Time Expert System, in [3.77], pg. 217-229
- [3.105] Moore R.L. et. al., A Real-Time Expert System for Process Control, in: The First Conference on Artificial Intelligence Applications, IEEE, Denver, 1984, pg. 569-576
- [3.106] Moore R.L., Khanna R., Artificial Intelligence on the Factory Floor: A success story, in Advanced Manufacturing Systems Conference, Chicago, 1986,pg. 509-516

- [3.107] Sachs P.A. Paterson A.M., Turner M.H.M., Escort - an expert system for complex operations in real time, in: Expert Systems, Vol. 3, No. 1, 1986, pg. 22-28
- [3.108] Leitch R.R., Kraft R., A Real-Time Knowledge Based System for Product Quality Control, in: International Conference on Control 1988, Oxford, UK, IEE Conf. publ. No. 285, pg. 281-286
- [3.109] Brwon M.C., The Dynamic Rescheduler: Conquering the Changing Production Environment, in [3.80], pg. 175-180
- [3.110] Cavalloro P., Cividati E., INCA: An Expert System for Process Planning in PCB Assembly Line, in [3.80], pg. 170-174
- [3.111] Khanna R. Moore R.L., Expert Systems Involving Dynamic Data for Decisions, in: 2. International Expert Systems Conference, Oxford, UK, 1986
- [3.112] Bowen B.A., Real-Time Expert Systems: A Status Report, in: AGARD Lecture Series No. 166 (1987)
- [3.113] Früchtenicht H.W., Wittig T., Ein Ansatz für Echtzeit-Expertensysteme, in atp, Heft 2/1987, pg. 78-82
- [3.114] Alty J.L. et. al., Literature and User Survey of Issues Related to Man-Maschine Interfaces for Supervision and Control Systems, ESPRIT-Conference 1985, pg. 719-729
- [3.115] Elzer P. et. al., Expertensysteme und hochauflösende Graphik zur Unterstützung des Bedienpersonals in der Prozeßleittechnik, in: Prozeßrechner 1988
- [3.116] Alty J.L., Mullin J., Dialogue Specification in the GRADIENT Dialogue System, in: People and Computers V, Cambridge University Press, 1989
- [3.117] Collet M., Hälker-Küsters M., Tischedorf M., Darstellung der Ergebnisse einer Untersuchung über den Einsatz von Expertensystemen im Bereich Umwelt, in [2.5], pg. 167-176
- [3.118] Weidemann R., Geiger W., XUMA - Ein Assistent für die Beurteilung von Altlasten, in [2.4]pg. 385-394
- [3.119] Günther O., Scheuer K., Expertensysteme im Umweltschutz, in [2.5], pg. 177-186
- [3.120] Grützner R., Simulation im Umweltschutz - Anwendung, Anforderung, Visualisierung, in [2.6], pg. 29-37

Literatur zu Kapitel 4

- [4.1] Denzer R., Hagen H., An Object-Oriented Architecture for User Interface Management in Distributed Applications, Interner Report, Univ. Kaiserslautern, 1991
- [4.2] Pflügl M. et. al., Interprocess Communication in MARS, in [3.6], pg. 189-202
- [4.3] Denzer R., User Interface Management in Distributed Systems, in [2.6], pg. 83-97
- [4.4] Denzer R., Informatikeinsatz im prozeßnahen Bereich an einer Anlage zur schadstoffarmen Müllverbrennung, in [2.4], pg. 329-337
- [4.5] Denzer R., Ein verteiltes Objektsystem zur Führung und Visualisierung technischer Prozesse, Interner Bericht, Universität Kaiserslautern, 1990
- [4.6] Kira G., Entwicklung eines objektorientierten Graphiksystems zur Visualisierung komplexer Prozesse, Diplomarbeit, Universität Kaiserslautern, 1991
- [4.7] Reißfelder M., Konzeption und Erstellung einer Softwareschnittstelle zwischen einem verteilten Message-Passing-System und OPS5, Studienarbeit, Berufsakademie Mannheim, 1990
- [4.8] Reißfelder M., Entwicklung eines verteilten, echtzeitfähigen Objektsystems in OPS5, Diplomarbeit, Berufsakademie Mannheim, 1990
- [4.9] AG Computergraphik der Universität Kaiserslautern, Spezifikation eines auf Xtoolkit basierenden Programmpakets zur Entkopplung einer Wissensbasis von graphischen Ein-/Ausgabekomponenten, Universität Kaiserslautern, 1990
- [4.10] Henshall J., Shaw S., OSI Explained, End-to-end Computer Communication Standards, John Wiley and Sons, 1988
- [4.11] Denzer R., User Interfaces für die Visualisierung von Umweltdaten - Anforderungen und Architektur, in [2.5], pg. 847-854

Literatur zu Kapitel 5

- [5.1] Denzer R., Informatikeinsatz im prozeßnahen Bereich an einer Anlage zur schadstoffarmen Müllverbrennung, in [2.4], pg. 329-337

- [5.2] Denzer R., Ein verteiltes Objektsystem zur Führung und Visualisierung technischer Prozesse, Interner Bericht, Universität Kaiserslautern, 1990
- [5.3] Denzer R., Interner Bericht, Kernforschungszentrum Karlsruhe, 1990
- [5.4] Denzer R., Interner Bericht, Kernforschungszentrum Karlsruhe, 1989
- [5.5] Denzer R., Hagen H., Kira G., Koob F., Using Process Knowledge for Adaptive User Interfaces, Bericht, Universität Kaiserslautern, 1991
- [5.6] Dittrich G., Kerpe R., TAMARA - Versuchsanlage zur schadstoffarmen Müllverbrennung, GVC-Kongress, Baden-Baden, 1989
- [5.7] Kira G., Entwicklung eines objektorientierten Graphiksystems zur Visualisierung komplexer Prozesse, Diplomarbeit, Universität Kaiserslautern, 1991