Nietsch, Thomas; Rinschede, Matthias; Rautenstrauch, Claus

Working Paper

Werkzeuggestützte Individualisierung des objektorientierten Leitstands ooL

Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 24

Provided in Cooperation with:
University of Münster, Department of Information Systems

Suggested Citation: Nietsch, Thomas; Rinschede, Matthias; Rautenstrauch, Claus (1993) : Werkzeuggestützte Individualisierung des objektorientierten Leitstands ooL, Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 24, Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, Münster

This Version is available at:
https://hdl.handle.net/10419/59346

Arbeitsberichte des Instituts für Wirtschaftsinformatik
Herausgeber: Prof. Dr. J. Becker, Prof. Dr. H. L. Grob, Prof. Dr. K. Kurbel,
Prof. Dr. U. Müller-Funk, Prof. Dr. R. Unland, Prof. Dr. G. Vossen

Arbeitsbericht Nr. 24

Werkzeuggestützte Individualisierung des objektorientierten Leitstands ooL

Michael Nietsch, Matthias Rinschede, Claus Rautenstrauch

Institut für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster,
Grevener Str. 91, 48159 Münster, Tel. (0251) 83-9750, Fax (0251) 83-9754
Dezember 1993

Inhalt

Zusammenfassung

Die Einführung elektronischer Leitstände wird in zahlreichen Unternehmen durch die Notwendigkeit, die Software an unternehmensindividuelle Bedürfnisse anzupassen, z.T. erheblich erschwert und verteuert. Die Nutzung objektorientierter Softwaretechnologie ist eine Möglichkeit zur Verringerung des Anpassungsaufwands.

Dieser Beitrag beschreibt die Individualisierungsmöglichkeiten des objektorientierten Leitstands "ooL". Anhand der Architektur des Systems werden die Anpassungen bezüglich der *Oberfläche*, des *Objektmodells* und der *Datenbank* beschrieben und anhand von Beispielen veranschaulicht.

1 Anpaßbarkeit und Individualisierung

Softwaresysteme sind komplexe technische Systeme, die im Gegensatz zu anderen vergleichbar komplexen Systemen permanent Änderungen unterworfen sind. Ursachen hierfür sind:

- die zumindest vordergründig einfache Änderbarkeit von Software,
- der technologische Fortschritt und
- der Modellcharakter der Software.

Änderbarkeit

Software ist auch nach der Fertigstellung (der ersten Version) prinzipiell unbeschränkt änderbar. Änderungen am Quellcode einer Applikation können grundsätzlich ohne großen technischen Aufwand durchgeführt werden. Auf der Code-Ebene können globale Zusammenhänge allerdings nur schwer erkannt werden, so daß die Gefahr besteht, daß lokale ("kleine") Änderungen unerwünschte Effekte auf das globale System nach sich ziehen. Daher sollten globale, auf der Entwurfsebene beschriebene Systemstrukturen auch im Programmcode direkt abgebildet sein, damit die Tragweite von Änderungen auch auf Codeebene erkennbar bleibt.

Technologischer Fortschritt

Die Anpassung an neue Standards (SQL3, X11, OSF-Motif etc.) und der Einsatz neuer leistungsfähigerer Rechner und Basissoftware (Betriebssysteme, Datenbanksysteme, graphische Systeme etc.) erfordern ebenfalls Änderungen der Softwaresysteme. Veraltete, nicht mehr gewartete Hardware erzwingt Portierungen auf modernere Plattformen. Technologische Fortschritte führen zudem häufig zu Systemerweiterungen. So haben z.B. Transputer bildverarbeitende Systeme entscheidend verbessert[1)] oder multimediale Systemerweiterungen neue Anwendungsbereiche bei der Fertigungssteuerung erschlossen[2)].

Modellcharakter der Software

Softwaresysteme sind Realisierungen eines Modells der jeweiligen Problemlösung. Sie sind im betrieblichen Bereich zweckbezogene, partielle Abbilder der Aufbau- und Ablauforganisation. Organisatorische Änderungen, Änderungen in der Ablauforganisation und Verfahrensänderungen wirken sich direkt auf die Anforderungen an das Softwaresystem aus[3)]. So verändert sich

---

1) Vgl. Grebe, Baumann (1992), S. 127 ff.
2) Vgl. Kurbel (1992).
3) Vgl. Thurner (1990), S. 13.

z.B. bei der Bildung von Dispositionszentren, d.h. Bereichen mit einer einheitlichen Steuerungsphilosophie, in einem Fertigungsunternehmen die Funktionalität des Fertigungssteuerungssystems in den verschiedenen Dispositionsbereichen, indem sich zunehmend dispositive Aufgabenstellungen in die teilautonomen Bereiche verschieben.

Die bei der Entwicklung gewählte Systemstruktur ist maßgeblich für die Änderbarkeit des Systems. Eine Zielsetzung beim Systementwurf ist die Ausrichtung der Struktur an möglichst fixen bzw. längerfristig stabilen Informationseinheiten. Hier kommen Daten, Funktionen oder Objekte in Frage. Daten(strukturen) sind gegenüber den Funktionen deutlich weniger Änderungen unterworfen, sagen aber auch wenig über dynamische Eigenschaften eines Systems aus. Letztere werden üblicherweise in Form von Funktionen beschrieben. Objekte bieten hier eine Alternative, indem sie Daten und Funktionen kapseln[4)] und die künstliche Trennung von Daten und Funktionen aufheben[5)].

Häufig sind Schwächen im Design und Programmierfehler (schlechte Dokumentation, viele Seiteneffekte, uneinheitliche Notation etc.) dafür verantwortlich, daß Softwaresysteme nur mit erheblichem Aufwand geändert werden können[6)]. Werden darüber hinaus Änderungen unsachgemäß vorgenommen ("gestrickt"), führt dies letztendlich zu nicht mehr änderbaren Softwaresystemen. Derartige Systeme müssen dann entweder reimplementiert oder aufwendig saniert werden.

Um diesen Problemen vorzubeugen, muß bei der Entwicklung komplexer Systeme gesteigerter Wert auf die Änderbarkeit gelegt werden[7)]. Dies gilt vor allem für Systeme, die bereits bei der Installation weitreichende Änderungen erfahren, indem sie an individuelle Anforderungen angepaßt werden.

Individualisierbarkeit bezeichnet hier eine spezielle Form der Anpaßbarkeit, bei der auch eine Veränderung grundlegender Funktionen eines Anwendungssystems mit vertretbarem Aufwand möglich ist.

Softwaresysteme können im Hinblick auf ihre Anpaßbarkeit bzw. Individualisierbarkeit in zwei Kategorien eingeteilt werden:

---

4) Vgl. zu objektorientierten Konzepten Nierstrasz (1988), S. 3 ff.; Booch (1991); Meyer (1990); Rumbaugh u.a. (1991) u.a.m.
5) Vgl. Hesse (1990), S. 51 ff.
6) Vgl. Jung (1993).
7) Änderbarkeit gehört zu den wichtigsten Softwarequalitätsmerkmalen; vgl. z.B. Kurbel (1983), S. 111 ff.

- Passen sich Systeme automatisch veränderten Randbedingungen an, spricht man von *adaptiven* Systemen.

- Demgegenüber stehen die *adaptiblen* Systeme[8], die durch den Endanwender oder entsprechend geschultes Personal an Individualanforderungen angepaßt werden können. Anpassungen, die nach der Systemfreigabe durch Anwender durchgeführt werden, verstärken die Identifizierung des Anwenders mit dem System und entlasten das eigentliche Entwicklerteam. Vor allem im Bereich der graphischen Oberflächen wird dieser Ansatz verfolgt und mit zahlreichen Tools unterstützt. Anpassungen werden nach End-User- und Super-User-Anpassungen sowie Anpassungen durch den technischen Spezialisten klassifiziert.

Der objektorientierte Leitstand ooL wurde als adaptibles System konzipiert und implementiert. Werkzeuge helfen dem Endbenutzer, Änderungen an der Benutzeroberfläche, dem Objektmodell und der Datenhaltung durchzuführen. Je nach Komplexität der durchzuführenden Anpassungen muß der Benutzer unterschiedlich qualifiziert bzw. mit dem System vertraut sein. Das Spektrum reicht von einem gelegentlichen Benutzer bis hin zum technischen Spezialisten, der auch komplexe Änderungen im Objektmodell durchführen kann.

2 Architektur von ooL

2.1 Entwicklungsumgebung

Der ganzheitlich objektorientierten und multimedialen Architektur des Leitstands ooL liegt ein objektorientiertes Hardware- und Softwarekonzept zugrunde. ooL wurde unter dem multimedia-geeigneten und Unix-basierten Betriebssystem NeXTStep 3.1 auf einer NeXT-Cube-Workstation entwickelt. Die Wahl fiel auf diese Hardware-Plattform, da sie hervorragende ergonomische Eigenschaften aufweist, ein größtenteils objektorientiertes Betriebssystem besitzt und von Haus aus mit Multimediafähigkeiten ausgestattet ist. Die Verarbeitungsfähigkeit von Echtzeit-Videos wird durch eine zusätzliche Multimediakarte (Dimension-Board) erreicht. Zur Konfiguration des Entwicklungsrechners gehören weiterhin ein hochauflösender Bildschirm, optische Platten, ein Videorecorder sowie Audioperipherie (Lautsprecher, Mikrophon).

---

[8] Zu adaptiblen und adaptiven Systemen vgl. Haaks (1992), S. 58 ff.

2.2 Softwarearchitektur

Die Architektur des Leitstands ist für das Verständnis der nachfolgenden Aussagen über die Anpaßbarkeit des Systems von grundlegender Bedeutung. Hierbei wird eine Zweiteilung in Objektmodell und Datenbanktechnologie vorgenommen. Zu Beginn der Implementierungsarbeiten stand kein anforderungsgerechtes objektorientiertes Datenbanksystem auf der gewählten Systemplattform zur Verfügung. Darüber hinaus sollte die Integrationsfähigkeit zu konventionellen PPS-Systemen gewahrt werden, so daß das aktive relationale Datenbanksystem Sybase für die Datenverwaltung eingesetzt wurde. Daher konnte das objektorientierte Paradigma nur partiell in die Datenhaltung umgesetzt werden.

2.2.1 Die Bedeutung des Objektmodells

Das Objektmodell ist ein Modell des zu entwickelnden Systems. Die Gestaltung des Objektmodells bedarf der Erfahrung und der Urteilskraft der Entwickler und ist das Ergebnis einer intellektuellen Leistung im Rahmen der objektorientierten Softwareentwicklung. Das Objektmodell entsteht in einem evolutionären Prozeß, der von der Systemanalyse über den Entwurf bis zur Implementierung geht und darüber hinaus Wartung und Anpassung umfassen kann. Die Begriffswelt der Problemanalyse bleibt während der gesamten Entwicklung weitestgehend unverändert bestehen. Die Isomorphie zwischen dem Objektmodell und der Implementierung erhöht im Vergleich zu konventionellen Systementwicklungen die Verständlichkeit des Codes, wodurch der Wartungs- und Individualisierungsaufwand reduziert wird.

2.2.2 Einsatz des aktiven relationalen Datenbanksystems Sybase

Sybase bietet, wie mittlerweile auch Systeme anderer Anbieter, die Möglichkeit, strukturierte Objekte mit applikationsorientiertem Verhalten zu verwalten. Das heißt, die Datenbank ist in der Lage, neben dem reinen Datenmanagement auch Algorithmen zu verwalten und auszuführen. Hierzu stehen folgende Mechanismen und Konstrukte zur Verfügung:

- Eine Programmiersprache, welche die Verbindung von SQL mit prozeduralen Sprachkonstrukten ermöglicht und damit den bisher bestehenden "impedance mismatch" zwischen den Konzepten der Datenbankanfragesprache und den traditionellen Programmiersprachen überwindet. Mit Hilfe dieser Konstrukte lassen sich Stored procedures für die Entwicklung komplexer Datenbankzugriffe erstellen und in der Datenbank abspeichern.

- Ein Triggerkonzept, das es ermöglicht, auf Änderungen der Daten ereignisgesteuert zu reagieren, um bspw. Mechanismen zur Integritätssicherung anzustoßen.

- Defaults, Rules und benutzerdefinierte Datentypen.

Werden alle Datenbankzugriffe über Stored procedures realisiert und erfolgt die Integritätssicherung über datenbankeigene Mechanismen, die ereignisorientiert ausgeführt werden, läßt sich die Datenbank vollständig vom Anwendungsprogramm abkapseln. Die Kommunikation zwischen Programm und Datenbank erfolgt ausschließlich über den Aufruf von Stored procedures, und datenbanksystem-spezifische Aufrufe werden vollständig aus dem Code des Anwendungsprogramms eliminiert. Bei einer solchen schema-transparenten Verbindung zwischen Anwendung und Datenbank ist dann die Realisierung der Stored procedures für die Applikation nicht mehr von Interesse, sondern wird von der Datenbank übernommen[9]. Sollte aus dem semantischen Umfeld der Applikation heraus eine Änderung - bspw. eine Erweiterung um Datenelemente oder Nebenbedingungen - notwendig werden, so wird diese zentral in der Datenbank durchgeführt. Die Applikation, die die Prozedur verwendet, muß nicht modifiziert werden.

Konsequenzen für die Anwendungsarchitektur

Bot sich für fensterorientierte und datenbankbasierte Anwendungen bislang eine dreischichtige Anwendungsarchitektur an, so ist heute die Aufteilung einer Anwendung in fünf Ebenen möglich (siehe Abbildung 1). Der Bereich unterhalb der gestrichelten Linie in der Abbildung kennzeichnet den Teil eines Systems, der vom Datenbanksystem verwaltet wird.

Die Anwendungslogik muß dabei in *gebundene Funktionen*, *Stored procedures* und *Integritätsregeln* aufgeteilt werden. Gebundene Funktionen sind diejenigen Funktionen, die bestimmten Anwendungen eindeutig zugeordnet sind oder aus technischen Gründen nicht als Stored procedures implementiert werden können. Diese Funktionen sind daher in den Programmcode von Anwendungssystemen eingebunden. Demgegenüber werden *globale Funktionen*, als Stored procedures implementiert. Beispiel für eine globale Funktion, ist ein Kalkulationsmodul, das sowohl von Programmen der Vor- als auch der Nachkalkulation benutzt werden kann. Integritätsregeln werden als Trigger oder Constraints implementiert. Auch wenn Trigger und Constraints strenggenommen globale Funktionen darstellen, unterscheiden sie sich von globalen Funktionen dadurch, daß sie ereignisorientiert (Ereignisse beziehen sich hier auf Veränderungen des Datenbestands) und nicht durch expliziten Aufruf ausgeführt werden.

---

9) Vgl. McGoveran (1989), S. 29.

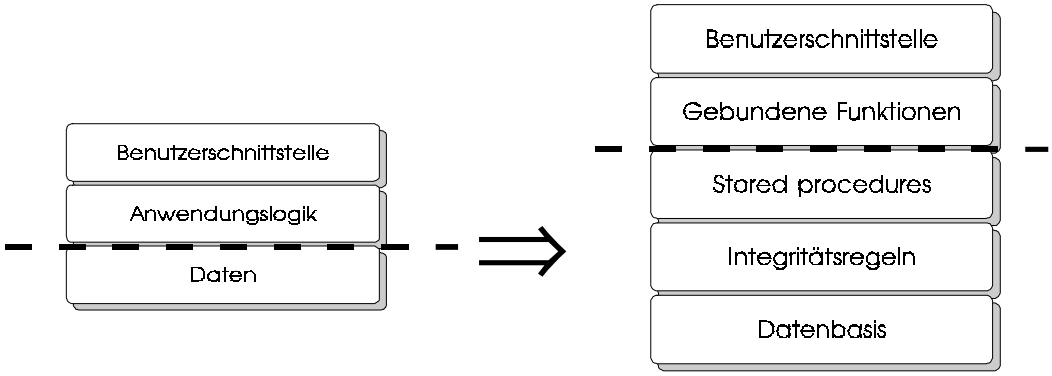

Abb. 1: Drei- und fünfschichtige Softwarearchitektur

Die konsequente Realisierung eines Application server[10)] macht die Datenbank komplexer, während die gebundenen Funktionen der Applikation in den Hintergrund treten. Gleichzeitig ist die Datenbank aber auch enger mit der Applikation verbunden. Bei konsequenter Weiterführung entfallen gebundene Funktionen fast komplett, und die Oberfläche kann als Fenster zur Datenbank betrachtet werden.

Sind an den Daten bzw. den darunterliegenden Datenbankschemata Anpassungen durchzuführen, so werden diese Änderungen durch Funktionen ausgeführt, die in der Datenbank gespeichert sind. Daten und zugehörige Operationen werden auf diese Weise gegenüber Anwendungsprogrammen gekapselt. Darüber hinaus bietet diese Architektur noch eine Reihe weiterer Vorteile, die im folgenden dargelegt sind:

- Die Datenbankschnittstelle wird trotz der engeren Kopplung von Datenbank und Applikation entlastet, weil hier statt (großer) Datenmengen als Ergebnisse relationaler Anfragen nur noch Resultatparameter von Stored procedures übertragen werden. Liegt die Datenbank in einer verteilten Umgebung auf einem anderen Netzknoten als das aufrufende Programm, so wird zusätzlich das Netzwerk entlastet. Beide Gesichtspunkte führen zu einer verbesserten Performance.

- Bei der Portierung von Softwaresystemen treten weniger Probleme auf, da man sich auf die Portabilität des Datenbanksystems abstützen kann.

- Auch die Entwicklung verteilter Systeme läßt sich einfacher durchführen, weil die Datenbank Mechanismen wie z.B. das Zwei-Phasen-Commit-Protokoll zur Verfügung stellt.

[10)] Eine Datenbank mit Stored procedures, Triggern, Constraints und Daten wird auch *Application server* genannt.

Der Einsatz einer aktiven Datenbank hat aber auch zumindest einen gravierenden Nachteil:

- Mit zunehmender Anzahl an Triggern und Stored procedures steigt auch die Zahl der Interdependenzen zwischen ihnen. Damit ist immer schwieriger nachzuhalten, welche Datenfelder von welchen Triggern und welchen Stored procedures beeinflußt werden. Um diese Komplexität zu beherrschen, muß die Datenbankprogrammierung explizit Softwareentwicklungsprinzipien berücksichtigen.

2.3 Implementierung von ooL

Die Architektur von ooL entspricht der beschriebenen 5-Schicht-Architektur. Neben Sybase für die Entwicklung des Application server wurden der NeXT User Interface Builder für die Entwicklung der Benutzeroberfläche und der Database Interface Builder für die Realisierung der Datenbankschnittstelle eingesetzt. Die gebundenen Funktionen wurden in Objective C implementiert. Die Darstellung des Objektmodells würde den Rahmen dieses Beitrags sprengen[11)].

3 ooL-Anpassungswerkzeug

Aufgrund der zuvor beschriebenen Architektur ist es nicht ausreichend, sich bei Anpassungen auf das Objektmodell des Leitstands zu beschränken. Vielmehr müssen auch Anpassungen in der Datenbank und bei der Benutzeroberfläche möglich sein und durch entsprechende Tools unterstützt werden.

Diesen Anforderungen entsprechend ist auch das Anpassungswerkzeug des Leitstands konzipiert, mit dem der Anwender seine Individualanforderung definiert und umsetzt. Für jeden Bereich befindet sich im ooL-Werkzeugkasten ein entsprechendes Tool. Durch Aktivierung der in Abbildung 2 dargestellten Buttons wird es gestartet.

---

11) Vgl. hierzu Nietsch u.a. (1992).

Abb. 2: ooL-Anpassungswerkzeuge

Einschränkend muß erwähnt werden, daß die Werkzeugunterstützung nicht in allen drei Bereichen gleichermaßen möglich ist. Oberflächen sind gute Kandidaten für eine Werkzeugunterstützung, da sie i.d.R. weitgehend unabhängig von der Applikation entwickelt werden können. Algorithmische Anpassungen und Datenbank-Anpassungen sind dagegen gewissen Einschränkungen unterworfen.

3.1 Oberflächenanpassungen bei ooL

Die Architektur des Leitstands ooL erlaubt dem Benutzer die individuelle Anpassung der Oberfläche an seine persönlichen Bedürfnisse. Anpassungen können auf drei verschiedenen Ebenen durchgeführt werden.

Einfache Anpassungen erfolgen zur Laufzeit mit menügesteuerten Konfigurationsmöglichkeiten, wie man sie z.B. von Endbenutzerwerkzeugen kennt. Auf dieser Ebene können intuitiv Anpassungen wie etwa Zeitachsenskalierungen, Statuslegenden, Arbeitsgangbeschriftungen etc. von Laien, die vorher noch nicht mit dem Leitstand ooL gearbeitet haben, durchgeführt werden. Abbildung 3 zeigt exemplarisch ein Konfigurationspanel für die Zeitachse, mit dem die Skalierung der Zeitachse sowie das in der Plantafel angezeigte Zeitintervall interaktiv verändert werden können.

Abb. 3: Konfigurationspanel für die Zeitachse

Anpassungen, die über die menügesteuerten Konfigurationsmöglichkeiten hinausgehen, können von eingearbeiteten ooL-Benutzern mit Hilfe des ooL-Oberflächentoolkits auf der Basis des User-Interface-Builder von NeXT durchgeführt werden. Diese Anpassungen werden nicht zur Programmlaufzeit durchgeführt. Ziel ist es, das Layout, d.h. Lage und Form der einzelnen Oberflächenobjekte, individuell anpassen zu können. Ist es aus Sicht des Anwenders bspw. sinnvoll, Objekte wie Zeitachse, Betriebsmittelleiste, Informationsfelder für Arbeitsgänge etc. an anderer Stelle im Applikationsfenster erscheinen zu lassen, als dies in der Grundeinstellung vorgesehen ist, können die graphischen Objekte im ooL-Oberflächentoolkit mit der Maus gegriffen und an der gewünschten Stelle positioniert werden.

In der dritten und weitreichendsten Anpassungsebene wird fortgeschrittenen Benutzern die Möglichkeit gegeben, nicht nur das Layout bestehender Oberflächen zu modifizieren, sondern auch völlig neue Oberflächenkomponenten zu gestalten. Hierzu stellt das ooL-Oberflächentoolkit in vier Paletten eine Vielzahl von Oberflächenobjekten zur Verfügung, die sich interaktiv in den Leitstand integrieren lassen. Abbildung 4 zeigt beispielhaft zwei Paletten für Menüobjekte und allgemeine graphische Objekte wie Push-Buttons, Textfelder etc. Mit Hilfe der Paletten können innerhalb kürzester Zeit neue Fenster in den Leitstand eingebunden oder bestehende Fenster um zusätzliche graphische Komponenten erweitert werden, ohne daß Änderungen am

Objective-C- oder Display Postscript-Code notwendig werden. Im Vergleich zu konventionellen Methoden lassen sich Änderungen damit bedeutend effizienter durchführen.

Abb. 4: Zwei Paletten aus dem ooL-Oberflächentoolkit

Jedem graphischen Objekt ist ein individueller Inspector zugeordnet, mit dem die Eigenschaften des Objekts festgelegt und zwischen verschiedenen Objekten Verbindungen hergestellt werden können. Abbildung 5 zeigt den Push-Button-Inspector, mit dessen Hilfe über eine einfache Auswahlliste die graphische Repräsentationsform des Push-Buttons - wie "Umrandet", "Inaktiv" etc. - festgelegt werden kann. Weiterhin können in dem Push-Button-Inspector Attribute wie bspw. Titel, Icon, Tastaturschlüssel und Sound zu einem Button wie in einem Formular eingetragen werden. Jede Einstellung der Attribute hat zur Laufzeit des Programms eine Objective-C-Nachricht an das entsprechende Objekt zur Folge, die das Objekt dann an die gewünschten Eigenschaften anpaßt.

Während die graphischen Gestaltungsmöglichkeiten für Buttons durch die kombinatorischen Variationsmöglichkeiten der Auswahlliste begrenzt sind, gibt es für die Titel-, Icon- und Soundvergabe keinerlei Beschränkungen. Der Titel kann interaktiv eingegeben und verändert werden. Sound und Icons müssen lediglich als Image im dargestellten ooL-Ressourcenmanager (vgl. Abbildung 6) verfügbar sein.

Der Ressourcenmanager stellt im oberen Teil vier "Aktenkoffer-Icons" zur Auswahl. Der erste Koffer beinhaltet die Oberflächenschnittstelle zum Objektmodell, der zweite Koffer die verfüg-

baren Icons und der dritte Koffer die im Leitstand bisher vorhandenen Sounds. Auf den letzten Koffer wird an späterer Stelle noch näher eingegangen.

Abb. 5: Push-Button-Inspector

Durch Doppelklick mit der Maus kann der jeweils gewünschte Koffer geöffnet und die verfügbaren Elemente angezeigt werden. Abbildung 6 zeigt oben links den geöffneten ersten Koffer "Objects", in dem sich der leistungsfähigste Teil des Oberflächentoolkit verbirgt: die Schnittstelle zwischen Objekten, die auf der Benutzeroberfläche angezeigt werden ("Oberflächenobjekte"), und Objekten, in denen die Anwendungslogik codiert ist ("logische Objekte").

Abb. 6: ooL-Ressourcenmanager

Im unteren Teilfenster sind neun von insgesamt 52 bisher verfügbaren Objekten abgebildet. Davon sind sechs Objekte Instanzen der Klasse "Window-Panel". Das durch das große "A" repräsentierte Objekt ist eine Instanz der Klasse "Font-Manager". Die durch die beiden Kugeln dargestellten Objekte sind Objekte der Leitstandlogik. Bei dem Objekt oben links handelt es sich um die "Auswahlkontrolle", die die Betriebsmittelselektion beim Starten der Plantafel überwacht. Das Objekt "AVComponent" stellt die funktionalen Eigenschaften des Arbeitsvorrats zur Verfügung, dessen Oberfläche im rein graphischen Objekt "Arbeitsvorrat" definiert ist. Oberfläche und Logik sind demnach streng voneinander getrennt. Trotzdem besteht zwischen beiden Objekten eine enge Kopplung, die durch sogenannte "Connections" erreicht wird. Durch Connections lassen sich zur Laufzeit des Programms zwischen beliebigen Instanzen graphischer oder funktionaler Klassen Nachrichtenflüsse auslösen und dadurch Aktionen in einem anderen Objekt anstoßen.

Beispiel: Anpassung des Hauptmenüs

Im folgenden soll die Anpassung der Leitstandoberfläche und der Funktionalität anhand eines einfachen Beispiels erörtert werden. Ziel der Anpassung sei es, über das Hauptmenü des Leitstands einen direkten, menügesteuerten Durchgriff auf den Arbeitsvorrat zu ermöglichen, ohne daß - wie bisher - vorher die elektronische Plantafel gestartet werden muß. Um dieses Ziel

zu erreichen, muß das Hauptmenü um einen weiteren Menüpunkt erweitert werden. Außerdem muß dem Window-Panel, in dem der Arbeitsvorrat angezeigt wird, mitgeteilt werden, daß sich das Arbeitsvorratsfenster öffnen soll, wenn der neue Menüpunkt angewählt wird.

Die Erweiterung des Hauptmenüs geschieht, indem - wie aus Abbildung 7 ersichtlich - aus der Menüpalette mit der Maus der Menüpunkt "Item" entnommen und über das Hauptmenü gezogen wird. Das Hauptmenü wird automatisch um den Menüpunkt erweitert. Zur Positionierung des Menüpunkts in der Menüleiste lassen sich die Menüpunkte ebenfalls mit der Maus verschieben. Als nächstes muß dem Menüpunkt ein sinnvoller Titel gegeben werden. Durch Doppelklick mit der Maus wird der Menüpunkt im Oberflächentool editierbar, und es kann der Menüpunkttitel "Arbeitsvorrat" eingegeben werden.

Abb. 7: Erweiterung des Hauptmenüs

Im nächsten Schritt muß der Menüpunkt noch mit einer Aktion hinterlegt werden. Hierzu wird mit Hilfe der Maus zwischen dem Quellobjekt, dem Menüpunkt "Arbeitsvorrat", und dem Zielobjekt, der Panel-Instanz "Arbeitsvorrat", eine Connection erzeugt. Die Connection wird durch eine Linie sichtbar (vgl. Abbildung 8). In dem hier betrachteten Beispiel soll bei Aufruf des Menüpunkts "Arbeitsvorrat" eine Nachricht an das Arbeitsvorratspanel geschickt werden, damit sich dieses öffnet. Deshalb muß im Connection-Inspector die Nachricht "makeKeyAndOrderFront" ausgewählt werden. Der Connection Inspector erscheint automatisch beim Erzeugen einer Connection und stellt nur die Methoden zur Wahl, die vom Empfänger der Nachricht zur Laufzeit auch verstanden werden. Das Arbeitsvorratspanel wiederum wird zur

Laufzeit eine Nachricht an das funktionale Arbeitsvorratsobjekt "AVComponent" schicken, damit es den Arbeitsvorrat im Arbeitsvorratspanel anzeigt.

Abb. 8: Erzeugen einer Connection

3.2 Anpassung der gebundenen Funktionen auf Basis des Objektmodells

Der Entwurf eines Objektmodells entspricht dem Systementwurf in einem konventionellen Anwendungssystem. Innerhalb des Modells können Teilmodelle für die Oberfläche, die Datenhaltung und die gebundenen Funktionen des Leitstands identifiziert werden. Im weiteren werden die Möglichkeiten zur Anpassung des Objektmodells näher beschrieben.

Voraussetzung für die Änderbarkeit von Objektmodellen ist, daß das Objektmodell so konzipiert ist, daß Änderungen auf minimale Ausschnitte des Systems beschränkt bleiben. Die hierzu notwendigen softwaretechnischen Voraussetzungen werden durch die objektorientierte Sprache bereitgestellt. Auf der Ebene der Klassen- bzw. Objektbeziehungen wird die Änderungslokalität

durch die Clusterung von Klassen unterstützt. Cluster sind Mengen von Klassen, die aus einer bestimmten Sicht logisch zusammengehören.

Bei der objektorientierten Modellentwicklung spielen die Vererbungsbeziehungen eine wesentliche Rolle. Durch die Vererbung werden aufbauend auf bereits existierenden Klassen neue Klassen definiert, die Strukturen und Verhalten der bestehenden Klassen übernehmen. Dabei müssen lediglich die Unterschiede zwischen der neuen und der vorhandenen Klasse durch "programming by difference" explizit beschrieben werden[12]. In Bezug auf die Individualisierbarkeit ist die Unterscheidung zwischen Typ- und Modulvererbung von Bedeutung. Während die Typvererbung Vererbungshierarchien auf der Basis der Generalisierung und Spezialisierung von Typen aufbaut, wird bei der Modul-Vererbung die Hierarchie auf der Basis von Code-Redundanzen entwickelt[13]. Beide Vererbungsarten müssen zum besseren Verständnis des Objektmodells klar voneinander getrennt werden.

Eine wesentliche Hilfe bei der Modellanpassung ist die Dokumentation des Modells. Komplexe Strukturen lassen sich angemessener in Form von Graphen als durch eine Auflistung aller ihrer Elemente erklären. Deshalb wurde zum besseren Verständnis ein Browser und ein Dokumentationswerkzeug entwickelt. Mit ihnen werden Komponenten (Klassen, Objekte) als Knoten und Vererbungsbeziehungen (is a, used) als Kanten dargestellt. Die Werkzeuge werden im folgenden kurz erklärt.

3.2.1 ooL-Klassenbrowser

Zum Verständnis einer Klasse ist es hilfreich, die Entwicklungsgeschichte, wie sie in der dazugehörenden Vererbungshierarchie dokumentiert ist, zu kennen. Mit Hilfe des ooL-Browsers können die durch Vererbung implizierten Veränderungen der Klassen schrittweise nachvollzogen werden. Abbildung 9 zeigt den Vererbungspfad für die Klasse AgGraph. Selbstdefinierte Klassen sind durch die fettere Schrift von den Systemklassen zu unterscheiden. Ausgehend von der vordefinierten Klasse View wird AgGraph über die Klasse OOLView und OOLGraph gebildet. Mit Hilfe des Scrollbar kann die Vererbungshierarchie bis zur der Ausgangsklasse Object verfolgt und die jeweils gemachten Erweiterungen vom Anwender nachvollzogen werden.

---

[12] Vgl. Booch (1991), S. 514.
[13] Vgl. Adolf u.a. (1993), S. 40 ff.

Abb. 9: Der ooL-Klassenbrowser

In dem mit Class-Inspector überschriebenen Fenster werden die Attribute (outlets), die externe Referenzen darstellen, und Methoden (actions) der Klasse beschrieben. Im Sinne des Information hiding wird lediglich das Protokoll (d.h. die öffentliche Schnittstelle) der Klasse angezeigt. Interne Strukturen und Methoden bleiben dem Betrachter verborgen.

3.2.2 ooL-Dokumentationswerkzeug

Das Werkzeug ooL-Dokumentation-Manager ist als Hypertext- bzw. Hypermediasystem realisiert und basiert auf der NeXTApplikation "Diagram!". Mit Hilfe dieses Tools kann der Anwender durch die Objektwelt des Leitstands navigieren. Die Knoten des Netzwerks bilden die Objekte der Leitstandanwendung. Benutzt-Beziehungen zwischen den Objekten werden durch die Verbindungen zwischen den Knoten visualisiert. Über die Knoten ist der Zugriff auf detailliertere Beschreibungen bis hin zum Programmcode möglich. Die Information kann den einzelnen Objekten in Form von verfeinerten Objektmodellen, Texten, Bildern oder gesprochener Sprache hinterlegt werden. Der Informationstyp wird über ein entsprechendes Symbol am jeweiligen Objekt deutlich gemacht. Abbildung 10 zeigt einen kleinen Ausschnitt der ooL-Dokumentation für das Arbeitsgangobjekt.

Abb. 10: Dokumentationsausschnitt im ooL-Dokumentation-Manager

Objekte werden in gestrichelten und schattierten Kästen dargestellt. Verbindungen zwischen den Kästen visualisieren Benutzt-Beziehungen. Über den Doppelpfeil können zusätzliche Dokumente, bspw. weitere Diagramme, Klasseninformationen etc., geöffnet werden. Der kleine Pfeil an den Objekten verweist auf das jeweilige Objektdiagramm des gekennzeichneten Objekts, die durch einfaches Anklicken des Symbols mit der Maus erreicht wird. In Abbildung 10 sind beim AgGraph- und Ag-Objekt zusätzlich gesprochene Erläuterungen hinterlegt. In diesen Fällen erscheint automatisch das Lippensymbol an den Objekten. Erklärungen zum dargestellten Diagrammausschnitt können vom Anwender über den im Fenster oben rechts dargestellten Hilfe-Button abgerufen werden. Im Leitstand werden graphische und logische Objekte voneinander getrennt. Die Klasse AgGraph beinhaltet alle graphischen Attribute, wie Form, Farbe, Ausmaß

etc. Sie steht mit der logischen Klasse Ag in einer Benutzt-Beziehung. Die Klasse Ag enthält die fachbezogenen Informationen zu einem Arbeitsgang wie Dauer, frühester Start, zugehöriger Fertigungsauftrag etc.

Für die Änderungen des Objektmodells ist das Dokumentationstool von grundlegender Bedeutung, da damit Änderungen in den Systemzusammenhang eingeordnet werden können. Es hilft dem Durchführenden, den Änderungsbereich zu verstehen, einzugrenzen und den Aufwand der Änderung abzuschätzen. Die konkreten Änderungen werden mit dem beim Klassenbrowser beschriebenen Klasseninspektor durchgeführt. Durch einen Doppelklick auf den Namen der zu ändernden Klasse werden die Klassendokumente (Schnittstellenbeschreibung und Implementierung) geöffnet. Nach erfolgter Änderung wird aus der Entwicklungsumgebung heraus die Übersetzung angestoßen und das System neu gebunden.

3.3 Datenbankanpassungen bei ooL

3.3.1 Aufbau des Datenbanktools

Wie in Kapitel 2 beschrieben, nimmt die Datenbank in der Architektur des Leitstands eine besondere Rolle ein. Deshalb wurde auch für den Datenbankbereich ein Anpassungstool in den ooL-Werkzeugkasten integriert.

Das ooL-Datenbanktoolkit basiert auf dem Entity-Relationship-Diagramm des Leitstands ooL. Die Entitäten, Attribute und Beziehungen werden dem Anwender graphisch veranschaulicht. Das Datenbankschema kann mit Hilfe des ooL-Datenbanktoolkits vom Anwender interaktiv modifiziert und ergänzt werden. Abbildung 11 zeigt den Einstieg in das Datenbankanpassungstool.

Im Entityfenster kann das zugrunde liegende Datenmodell um Entitäten erweitert werden. Wird eine bereits bestehende Entität selektiert, kann diese modifiziert, entfernt oder für weitere Zugriffe versteckt werden.

Wird ein Attribut selektiert, erscheint der in Abbildung 12 auf der linken Seite dargestellte Attribut-Inspector. Mit Hilfe dieses Panels können Eigenschaften des selektierten Attributs wie bspw. Name, Datentyp etc. verändert werden.

Abb. 11: Einstieg in das ooL-Datenbankanpassungstool

Wird im Attributfenster eine Beziehung selektiert, erscheint der in Abbildung 12 auf der rechten Seite dargestellte Relationship Inspector. Der Relationship-Inspector erleichtert die Erstellung und Modifikation von Beziehungen zwischen zwei Objekten.

Abb. 12: Attribut- und Relationship-Inspector

3.3.2 Integration von Oberflächen- und Datenbanktoolkit

Oberflächen- und Datenbanktoolkit sind einheitlich konzipiert und wurden zu einem integrierten Anpassungstoolkit für Oberfläche und Datenbank zusammengefaßt. Die einzelnen Komponenten der Datenbank wurden mit einer graphischen Oberfläche ausgestattet, die über zwei zusätzliche Paletten im ooL-Oberflächentoolkit enthalten sind.

Bei der Entwicklung neuer Leitstandkomponenten, die sich direkt auf die Datenbank abstützen, können - wie oben erwähnt - einzelne Bausteine von den Datenbank-Paletten ausgewählt und bspw. in einem neuen Fenster zusammen mit Oberflächenobjekten arrangiert werden. Die Relationen der Datenbank stellen sich für die Applikation als Klassen mit einfachen Zugriffsoperationen dar, deren Klassenattribute den Relationenattributen entsprechen. Der Typ eines Klassenattributs entspricht dem Datentyp, der zuvor im Attribut-Inspector festgelegt wurde.

Abbildung 13 zeigt die Rückmeldemaske für Arbeitsgänge, die aus Oberflächen- und Datenbankobjekten zusammengesetzt ist. Durch Betätigung des OK-Buttons werden die zuvor eingegebenen Daten in die Datenbank geschrieben. Diese Maske wurde mit Hilfe des kombinierten

Toolkits erstellt, indem die Eingabefelder der Maske mit dem Attribut Rückmeldemenge und die Push-Buttons "Fertig" und "teilweise rückgemeldet" mit dem Attribut Status der Relation "Arbeitsgang" verknüpft wurden. Die eindeutige Identifikation des betreffenden Arbeitsgangs erfolgt bei Aufruf der Rückmeldemaske durch die Arbeitsgang- und Fertigungsauftragnummer. Beim Drücken des OK-Buttons werden die eingetragenen Werte in die Relation Arbeitsgang eingefügt. Bei Betätigen der Status-Buttons wird der Status des Arbeitsgangs in der Datenbank aktualisiert.

Abb. 13: Mit dem ooL-Werkzeugkasten erstellte Rückmeldemaske für Arbeitsgänge

Auch wenn die oben beschriebene Anpassung bzw. Erweiterung des Leitstands wesentlich einfacher als bei konventioneller Softwaretechnik durchzuführen ist, muß an dieser Stelle dennoch darauf hingewiesen werden, daß auch das ooL-Datenbanktoolkit nicht jede Art von Anpassung einheitlich unterstützt. Während Erweiterungen unproblematisch sind, können Modifikationen und Löschungen zu Problemen führen, die im Vorfeld nicht ersichtlich sind und die vom Datenbanktoolkit auch nicht abgefangen werden. Der Grund liegt darin, daß von nahezu beliebiger Stelle aus auf die Datenbank zugegriffen werden kann. Hierdurch wird zwar eine Erweiterung sehr komfortabel ermöglicht; der Überblick, welche Komponenten auf welche Relationen zugreifen, geht aber verloren. Eine Modifikation oder Löschung einzelner Relationen kann deshalb zu unerwarteten Reaktionen führen.

Die Erstellung eines automatischen Dokumentationstools, das die Änderungen bzw. Erweiterungen am Leitstand vollständig automatisch protokolliert, ist bislang nicht möglich, da Änderungen an der Datenbank durch den Endanwender losgelöst von der objektorientierten Entwicklungsumgebung durchgeführt werden können. Datenbankänderungen müssen daher explizit dokumentiert werden, andernfalls führen Änderungen zu unvollständiger und möglicherweise fehlerhafter Dokumentation. Um Dokumentationfehler von vorneherein zu vermeiden, wurde darauf verzichtet, die Zugriffe des Leitstands auf die Datenbank für den Endanwender zu

dokumentieren, obwohl es mit Hilfe des ooL-Dokumentation-Managers leicht möglich gewesen wäre.

Auch Änderungen der abgelegten Stored procedures sind nur beschränkt möglich. Solange die Schnittstelle der bestehenden Stored procedures nicht verändert wird, sind Änderungen innerhalb einer Prozedur unproblematisch. Ebenso unproblematisch ist es, neue Stored procedures hinzuzufügen. Veränderungen des Prozedurkopfs oder Löschungen ganzer Prozeduren können hingegen weitreichende Veränderungen der Datenbankfunktionalität bewirken und sind ausschließllich Datenbankexperten vorbehalten.

4 Ausblick

Die drei Anpassungsbereiche der objektorientierten Leitstandsarchitektur (Oberfläche, Objektmodell, Datenbank) unterscheiden sich hinsichtlich der erreichten Individualisierbarkeit voneinander. Während Anpassungen der Oberfläche und des Objektmodells gut bis sehr gut unterstützt werden, ist die Individualisierbarkeit im Bereich der Datenbank durch die relationale Struktur eingeschränkt. Besonders für die ausgelagerten Funktionen müßten die Strukturierungsmöglichkeiten innerhalb der Datenbank erweitert werden. Die Kombination eines objektorientierten Leitstands mit einer relationalen Datenbank darf daher nur als Zwischenlösung auf dem Weg zu einer ganzheitlich objektorientierten Architektur angesehen werden. Trotz des Einsatzes der aktiven Datenbank bleibt die semantische Lücke zwischen den flachen Strukturen einer relationalen Datenbank und den komplexen Strukturen des Objektmodells bestehen. Mit einer objektorientierten Datenbank kann diese Lücke geschlossen werden, wodurch nicht nur die Transparenz und Verständlichkeit, sondern auch die Individualisierbarkeit weiter verbessert wird.

Literatur

Adolf, M., Fillhardt, H.: Der Vererbungsbegriff in der Objektorientierung; HMD 30 (1993) 170, S. 35-46.

Booch, G.: Object Oriented Design with Applications; Redwood City CA,1991.

Grebe, R., Baumann, M. (Hrsg.): Proceedings zum Transputer-Anwender-Treffen (TAT ´92); Aachen 1992.

Haaks, D.: Anpaßbare Informationssysteme; Göttingen u. a. 1992.

Hesse, W.: Objektorientierte Anwendungsmodellierung - ein Weg zur (Re-)Strukturierung von Software-Anwendungssystemen; in: Thurner, R. (Hrsg.): Reengineering - Ein integrales Wartungskonzept zum Schutz von Software-Investitionen; Halbergmoos 1990, S. 45-64.

Jung, R.: Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht, Arbeitsbericht Nr. 16 des Instituts für Wirtschaftsinformatik, Münster 1993.

Kurbel, K.: Produktionsplanung- und steuerung - Methodische Grundlagen von PPS-Systemen und Erweiterungen; München, Wien 1993.

Kurbel, K.: Multimedia-Unterstützung für die Fertigungssteuerung; ZwF 87 (1992) 12, S. 664-668.

Kurbel, K.: Software Engineering im Produktionsbereich; Wiesbaden 1983.

McGoveran, D.: The Power of Stored Procedures; Database Programming & Design 2 (1989) 9, S. 29-43.

Meyer, B.: Objektorientierte Softwareentwicklung; München 1990.

Nierstrasz, O.: A Survey of Object-Oriented Concepts; in: Kimm, W., Lochovsky, F. (Hrsg.): Object-Oriented Concepts, Databases and Applications; Reading u.a. 1989, S. 3-21.

Nietsch, M., Rinschede, M., Rautenstrauch, C.: Konzeption und Entwicklung eines Objektmodells für einen individualisierbaren Leitstand; in: Görke, W., Rininsland, H., Syrbe, M. (Hrsg): 22. GI-Jahrestagung, Information als Produktionsfaktor; Proceedings, Berlin u. a. 1992, S. 597-606.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modelling and Design; Englewood Cliffs 1991.

Thurner, R.: Beherrschung des Technologie-Wechsels mit Reengineering; in: Thurner, R. (Hrsg.): Reengineering - Ein integrales Wartungskonzept zum Schutz von Software-Investitionen; Halbergmoos 1990, S. 11-20.

Arbeitsberichte des Instituts für Wirtschaftsinformatik

Nr. 1 Bolte, Ch., Kurbel, K., Moazzami, M., Pietsch, W.: Erfahrungen bei der Entwicklung eines Informationssystems auf RDBMS- und 4GL-Basis; Februar 1991.

Nr. 2 Kurbel, K.: Das technologische Umfeld der Informationsverarbeitung - Ein subjektiver 'State of the Art'-Report über Hardware, Software und Paradigmen; März 1991.

Nr. 3 Kurbel, K.: CA-Techniken und CIM; Mai 1991.

Nr. 4 Nietsch, M., Nietsch, T., Rautenstrauch, C., Rinschede, M., Siedentopf, J.: Anforderungen mittelständischer Industriebetriebe an einen elektronischen Leitstand - Ergebnisse einer Untersuchung bei zwölf Untenehmen; Juli 1991.

Nr. 5 Becker, J., Prischmann, M.: Konnektionistische Modelle - Grundlagen und Konzepte; September 1991.

Nr. 6 Grob, H.L.: Ein produktivitätsorientierter Ansatz zur Evaluierung von Beratungserfolgen; September 1991.

Nr. 7 Becker, J.: CIM und Logistik; Oktober 1991.

Nr. 8 Burgholz, M., Kurbel, K., Nietsch, Th., Rautenstrauch, C.: Erfahrungen bei der Entwicklung und Portierung eines elektronischen Leitstands; Januar 1992.

Nr. 9 Becker, J., Prischmann, M.: Anwendung konnektionistischer Systeme; Februar 1992.

Nr. 10 Becker, J.: Computer Integrated Manufacturing aus Sicht der Betriebswirtschaftslehre und der Wirtschaftsinformatik; April 1992.

Nr. 11 Kurbel, K., Dornhoff, P.: A System for Case-Based Effort Estimation for Software-Development Projects; Juli 1992.

Nr. 12 Dornhoff, P.: Aufwandsplanung zur Unterstützung des Managements von Softwareentwicklungsprojekten; August 1992.

Nr. 13 Eicker, S., Schnieder, T.: Reengineering; August 1992.

Nr. 14 Erkelenz, F.: KVD2 - Ein integriertes wissensbasiertes Modul zur Bemessung von Krankenhausverweildauern - Problemstellung, Konzeption und Realisierung; Dezember 1992.

Nr. 15 Horster, B., Schneider, B., Siedentopf, J.: Kriterien zur Auswahl konnektionistischer Verfahren für betriebliche Probleme; März 1993.

Nr. 16 Jung, R.: Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht; Juli 1993.

Nr. 17 Miller, C., Weiland, R.: Der Übergang von proprietären zu offenen Systemen aus Sicht der Transaktionskostentheorie; Juli 1993.

Nr. 18 Becker, J., Rosemann, M.: Design for Logistics - Ein Beispiel für die logistikgerechte Gestaltung des Computer Integrated Manufacturing; Juli 1993.

Nr. 19 Becker, J.; Rosemann, M.: Informationswirtschaftliche Integrationsschwerpunkte innerhalb der logistischen Subsysteme - Ein Beitrag zu einem produktionsübergreifenden Verständnis von CIM; Juli 1993.

Nr. 20 Becker, J.: Neue Verfahren der entwurfs- und konstruktionsbegleitenden Kalkulation und ihre Grenzen in der praktischen Anwendung; Juli 1993.

Nr. 21 Becker, J., Prischmann, M.: VESKONN - Prototypische Umsetzung eines modularen Konzepts zur Konstruktionsunterstützung mit konnektionstischen Methoden; November 1993.

Nr. 22 Schneider, B.: Neuronale Netze für betriebliche Anwendungen: Anwendungspotentiale und existierende Systeme; November 1993.