Multiperspektivische Unternehmensmodellierung

Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung

Ulrich Frank

Oldenbourg 1994
ISBN 3-486-22922-2

Vorwort

Betriebliche Informationssysteme sollten einerseits fundierten Gestaltungsprinzipien der Informatik genügen, andererseits sollten sie die Geschäftsabläufe in einem Unternehmen in möglichst wirtschaftlicher Weise unterstützen. Seit geraumer Zeit ist es Konsens, daß der Entwurf solcher Systeme angemessene konzeptuelle Modelle empfiehlt. In den zurückliegenden Jahren haben objektorientierte Modellierungsansätze in der angewandten Informatik eine zunehmende Bedeutung gewonnen. Gleichzeitig gab es in der Betriebswirtschaftslehre Bemühungen, die Potentiale, die moderne Informationstechnologie für die strategische Planung und die organisatorische Gestaltung eröffnet, sowie die damit verbundenen Herausforderungen näher zu untersuchen. Hier sei an Schlagworte wie "papierloses Büro", "lean management", "elektronische Märkte", "Computer Integrated Business" und dergleichen erinnert.

Im Jahre 1989 wurde in der Forschungsstelle für Wirtschaftsinformatik der GMD damit begonnen, einen Ansatz zu entwerfen, der die verschiedenen Aspekte, die für den Entwurf, die Implementierung und die Nutzung betrieblicher Informationssysteme zu berücksichtigen sind, miteinander verbinden sollte. Dabei wurde sehr früh deutlich, daß ein solcher Ansatz *inter*disziplinär sein muß: Die in den verschiedenen Disziplinen - vor allem in der Betriebswirtschaftslehre und in der Informatik - durchzuführenden Arbeiten sollten nicht isoliert erfolgen, sondern in ständigem wechselseitigem Austausch. Darüber hinaus hatten wir sehr bald erkannt, daß eine gemeinsame Orientierung wichtig ist, die für alle beteiligten Kollegen - ob sie sich nun eher den Wirtschaftswissenschaften oder der Informatik zuordneten - eine tragfähige, sinnstiftende Funktion erfüllt. Es zeigte sich, daß Integration und Wiederverwendbarkeit - beide in einem umfassenden Sinn verstanden - eine solche Orientierung lieferten.

Schon nach kurzer Zeit zeichneten sich vielversprechende Konzepte ab - entsprechend groß war unsere Motivitation. Leider begann gerade zu jener Zeit eine Restrukturierung der GMD, die zu einer Auflösung des Projektteams führte. Glücklicherweise gab es im Institut für angewandte Informationstechnologie der GMD großes Interesse an dem Thema, so daß ich dort die begonnene Arbeit fortsetzen konnte. Auf diese Weise entstand ein theoretischer Bezugsrahmen, in dem neben fortschrittlichen softwaretechnischen Konzepten auch die Anforderungen an eine dem Technikeinsatz angemessene Gestaltung des organisatorischen Kontextes Berücksichtigung findet. Der Bezugsrahmen zeigt auf, wie die Wirtschaftlichkeit des Entwurfs, der Implementierung und der Nutzung betrieblicher Informationssysteme wesentlich verbessert werden kann.

Angesichts der Weite und Komplexität des Themas wäre es vermessen, von einer abgeschlossenen Theorie zu sprechen. Ich hoffe allerdings, daß die Darstellung deutlich macht, wie der Bezugsrahmen weiter ausgestaltet und ausgefüllt werden

kann. Die Diskussionen, die sich an die bisherigen Präsentationen des Bezugsrahmens und der darauf aufbauenden Entwurfsumgebung anschlossen, bestärken mich in dieser Hoffnung. In diesem Zusammenhang ist es mir eine Freude, den Kollegen in der GMD, mit denen ich in den letzten Jahren zusammengearbeitet habe, herzlich zu danken. Sie haben mir mit ihrer konstruktiven Kritik und ihrem fundierten Wissen sehr geholfen. Zudem waren sie mir eine wichtige Quelle der Motivation, indem sie mich - jenseits mancher Differenzen im Detail - in meiner Begeisterung für den hier vorgestellten Ansatz immer wieder gestärkt haben.

Das vorliegende Buch wurde von der wirtschaftswissenschaftlichen Fakultät der Universität Marburg als Habilitationsschrift angenommen. Den beteiligten Professoren möchte ich für die freundliche Aufnahme in der Fakultät danken - besonders Ulrich Hasenkamp, der mich in meinem Vorhaben von Anbeginn an nachhaltig unterstützt hat.

Ulrich Frank Köln im Dezember 1993

Zusammenfassung

In der vorliegenden Schrift wird ein Ansatz zum Entwurf objektorientierter Unternehmensmodelle entwickelt. Er ermöglicht, durch die Verwendung der gleichen anwendungsnahen Konstrukte für die verschiedenen Phasen der Systementwicklung, die vielfältigen Friktionen zwischen Spezifikation und Implementierung erheblich zu reduzieren. Dabei werden nicht nur softwaretechnische, sondern auch betriebswirtschaftliche Anforderungen berücksichtigt. Dazu werden verschiedene Abstraktionsebenen vorgeschlagen, die eine Konzeptualisierung wichtiger Perspektiven auf das Unternehmen sowie eine Integration derselben gestatten. Auf diese Weise kann etwa der Fokus auf Aspekte der strategischen Planung, auf die Organisation von Geschäftsprozessen oder auf die Spezifikation der Klassen eines Objektmodells gerichtet werden.

Die auf dem vorgestellten Modellierungsansatz basierende Entwicklungsumgebung wird detailliert beschrieben. Zu ihren Vorzügen gehört u.a. eine Benutzerschnittstelle, die ein Unternehmensmodell aus unterschiedlichen Perspektiven und variierenden Detaillierungsgraden zu betrachten gestattet. Zudem erlaubt sie schnelles Prototyping und beinhaltet Verfahren, die die Analyse und Simulation organisatorischer Gestaltungsalternativen unterstützen.

Inhaltsverzeichnis

I. Einleitung

"Could it be that we are putting fifth generation technology in second generation organizations?"

Charles M. Savage

"To most people, including a surprising number who program computers, software engineering is a mystery."

Allen Macro und John N. Buxton

Der in den zurückliegenden Jahrzehnten zu verzeichnende Fortschritt in der Informationstechnologie äußert sich vordergründig in einer drastisch gestiegenen Leistungsfähigkeit von Hardware und einer erheblich ausgeweiteten Funktionsbandbreite bei gleichzeitig wesentlich verbesserter Ergonomie von Software. Daneben ist eine Fülle von Methoden, Konzepten und Werkzeugen entstanden, die darauf gerichtet sind, die häufig noch unbefriedigende Wirtschaftlichkeit der Entwicklung, Nutzung und Pflege computergestützter Informationssysteme zu fördern. Hier ist an Analyse- und Entwurfsmethoden, Werkzeuge zur Programmentwicklung und -pflege und - von herausragender Bedeutung - an Datenbankmanagement-Systeme zu denken. Solche Ansätze sind zumeist - mehr oder weniger explizit - zwei komplementären Zielen verpflichtet: Einerseits soll der Prozeß der Software-Entwicklung und -pflege rationalisiert werden, andererseits soll die Qualität von Software und allgemein von Informationssystemen gefördert werden. Seit einigen Jahren werden in diesem Bereich Ansätze diskutiert, die eine weitere erhebliche Wirtschaftlichkeitssteigerung versprechen. Sie ranken sich, auf unterschiedlichen Detaillierungsebenen und mehr oder weniger eng miteinander assoziiert, um Begriffe wie Wiederverwendbarkeit, Objektorientierung, Integration und - Unternehmensmodellierung.

Die Modellierung von Unternehmen ist seit langem ein bedeutendes Instrument betriebswirtschaftlicher Forschung, das einem weiten Spektrum unterschiedlicher Ziele dient. Im Kontext der vorliegenden Arbeit ist der Begriff Unternehmensmodellierung eingeschränkt auf seine Verwendung im Rahmen der Gestaltung und organisatorischen Implementierung betrieblicher Informationssysteme.

1. Unternehmensmodelle: Motive, Ziele und Herausforderungen

In vielen Unternehmen gibt es mittlerweile langjährige Erfahrungen mit der Gestaltung, Nutzung und Pflege computergestützter Informationssysteme. Einer wachsenden Zahl von Mitarbeitern ist der Umgang mit solchen Systemen - sei es als Entwickler oder Anwender - zur alltäglichen Routine geworden. Dennoch

kann kaum von einer zufriedenstellenden Situation gesprochen werden. So sind auch zwanzig Jahre nach Verkündung der sogenannten Software-Krise in allen Phasen des Software-Lebenszyklus Schwierigkeiten zu verzeichnen:

- Während der Analyse
  - werden die Anforderungen häufig nicht vollständig und authentisch erfaßt.
  - vollzieht sich die Kommunikation zwischen den Beteiligten (Entwickler, Anwender, Führungskräfte) nicht auf einer einheitlichen Sprachebene.
  - werden die Ergebnisse so präsentiert, daß den Anforderungen an den Softwareentwurf nicht in wünschenswerter Weise genügt wird.
- Für das Design und die Implementierung gilt:
  - Die Software-Architektur berücksichtigt Anforderungen an Wartbarkeit und Anpaßbarkeit nicht in wünschenswerter Weise.
  - Der implementierte Code ist in der Regel durch ein geringes Maß an Portabilität gekennzeichnet.
  - Die Planung und Durchführung von DV-Projekten ist mit einem hohen Maß an Unsicherheit über den zeitlichen Ablauf und den Ressourcenbedarf verbunden.
  - Die Qualifikation der Systementwickler ist häufig unzureichend.
  - Der Übergang von Design zu Implementierung ist sehr aufwendig und häufig mit dem Verlust von Information verbunden.
  - Die Dokumentation ist nicht vollständig und unverständlich. Es gibt häufig keine Richtlinien für eine einheitliche Dokumentation.
  - Der sogenannte Anwendungsstau ist immer noch weit verbreitet.[1]

Es liegt auf der Hand, daß die skizzierten Unzulänglichkeiten erhebliche betriebswirtschaftliche Konsequenzen haben: Sie stehen für einen erhöhten Aufwand der Software-Entwicklung und -pflege sowie für unzulänglich gestaltete Systeme, deren organisatorische Einbindung zu wünschen übrig läßt. Eine wesentliche Ursache für diese Schwierigkeiten ist darin zu sehen, daß die am Prozeß der Systementwicklung Beteiligten unterschiedliche *Perspektiven* auf das Problem haben: Während die einen (software-) technische Aspekte im Vordergrund sehen, ist die Wahrnehmung anderer wesentlich durch den Blickwinkel der eigenen Aufgabe geprägt. Solche Perspektivendifferenzen sind schwer zu überwinden, da die technischen Aspekte der Datenverarbeitung eine hohe Komplexität aufweisen, für die sich im Zeitverlauf eine entsprechend aufwendige Fachterminologie entwickelt hat.

Es ist seit langem allgemein anerkannt, daß eine *methodengestützte Modellierung*

1. So verweisen Mercurio et al. (1990, S. 170) auf eine Studie, nach der der durchschnittliche Rückstau in der Anwendungsentwicklung knapp vier Jahre beträgt.

von Anwendungsbereichen ein probates Mittel zur Überwindung der dargestellten Schwierigkeiten darstellt. Von besonderer Bedeutung ist dabei die sogenannte konzeptuelle Modellierung. Sie betont die Abstraktion von technischen Gegebenheiten (wie Hardware oder Programmiersprachen) und stellt stattdessen Konstrukte bereit, die Konzepte abzubilden gestatten, die im jeweiligen Anwendungsgebiet gebräuchlich sind. Gleichzeitig stellt sie für die Beschreibung der Konstrukte sowie für deren Verknüpfung ein formales Gerüst bereit, das einerseits ein systematisches Vorgehen fördert, andererseits die Transformation der Modelle in implementierungsnähere Repräsentationen unterstützt. Konzeptuelle Modelle stellen damit ein Medium dar, das die Verständigung zwischen Anwendern und Entwicklern zu verbessern verspricht. Darüber hinaus tragen sie zur Vermeidung von Friktionen zwischen den verschiedenen Phasen der Software-Entwicklung bei (was sicherlich besonders für die Transformation von Datenmodellen in entsprechende Schemata von Datenbankmanagement-Systemen gilt). Konzeptuelle Modelle sind in der Regel datenorientiert, das heißt, die Konstrukte, aus denen sie erstellt werden, werden (im wesentlichen) auf Datenstrukturen abgebildet, Funktionen bleiben unberücksichtigt. Dahinter verbirgt sich einerseits ein softwaretechnisches Prinzip, wonach der Entwurf von Programmen auf einem Datenmodell gründen soll. Andererseits - damit zusammenhängend - wird vielfach die Ansicht vertreten, daß Datenstrukturen im Zeitverlauf stabiler sind als Funktionen.

Der wesentliche Anreiz konzeptueller Modellierung liegt weniger in der isolierten Erstellung von Modellen für einzelne Anwendungen. Durch die anwendungsübergreifende Modellierung von Realitätsbereichen entstehen Datenmodelle, die von mehreren Anwendungen benutzt werden können. Auf diese Weise werden die Möglichkeiten der Inter-Anwendungskommunikation sowie der gemeinsamen Nutzung von Daten gefördert - kurz: die Integration von Informationssystemen. Angesichts des Ziels, möglichst alle Anwendungen eines betrieblichen Informationssystems zu integrieren, liegt es nahe, ein Datenmodell für das gesamte Unternehmen anzustreben. Entsprechende Ansätze einer unternehmensweiten Datenmodellierung werden denn auch seit einigen Jahren diskutiert (Vetter 1987, Scheer 1988 a).

Auch wenn eine unternehmensweite konzeptuelle Datenmodellierung eine erhebliche Verbesserung der gegenwärtigen Situation verspricht, kann doch nicht übersehen werden, daß Informationssysteme nicht nur aus Daten bestehen, sondern zu einem wesentlichen Teil aus Funktionen bzw. Prozeduren. Auch sie könnten anwendungsübergreifend modelliert werden. Auf diese Weise würde sich die Chance, einmal implementierte Prozeduren mehrfach wiederzuverwenden, erheblich verbessern. Diese Überlegungen waren das wesentliche Motiv für Ansätze, unternehmensweite Datenmodelle um unternehmensweite Funktions-

modelle zu ergänzen. Als Ausdruck dieser umfassenderen Sicht wird seit einiger Zeit der Begriff Unternehmensmodell verwendet. Die wenigen dedizierten Ansätze zur Unternehmensmodellierung lassen allerdings einen Anspruch erkennen, der deutlich weiter geht. Diese Ansätze präsentieren sich nicht in einheitlicher Form. Die folgenden Erweiterungen werden in unterschiedlicher Weise akzentuiert.

Ausweitung des Blickfelds

Während konzeptuelle Modellierung traditionell auf die Teile eines Unternehmens fokussiert ist, die auf ein computergestütztes Informationssystem abgebildet werden sollen, geht die Unternehmensmodellierung davon aus, daß eine solche Sicht verkürzt ist. Dazu wird darauf hingewiesen, daß Informationssysteme Geschäftsprozesse unterstützen sollten oder allgemeiner: einen wirksamen Beitrag zur Erreichung der Unternehmensziele leisten sollten. Das setzt ein angemessenes Verständnis der betriebswirtschaftlichen Zusammenhänge voraus. Gerade hier gibt es bei vielen DV-Experten erhebliche Defizite (Lederer/Mendelow 1987). Um diese Zusammenhänge sowie ihre Beziehung zu Daten- und Funktionsmodellen zu verdeutlichen, werden weitere Teilmodelle bzw. Abstraktionsebenen eingeführt. Scheer (1991) schlägt eine sogenannte Organisationsschicht vor, in der Organisationsstruktur, Benutzerklassen und Geschäftsprozesse abzubilden sind. Auch Kosanke und Vlietstra (1989, S. 663) betonen nachdrücklich eine umfassende Sicht: "... CIM-OSA embraces the complete enterprise operation, and describes it as an integral system."

Während die Betrachtung des Gesamtunternehmens einerseits als Voraussetzung für die angemessene Gestaltung von Informationssystemen angesehen wird, betonen manche Autoren einen weiteren Aspekt: Eine stärkere Einbeziehung von Führungskräften durch die Abbildung von Konzepten, die vom Management als wesentlich angesehen werden. Damit soll realen Entscheidungsprozessen und -kriterien Rechnung getragen werden. In diesem Sinne unterstreicht Katz (1990, S. 512) die Bedeutung von "dollars spent and dollars saved". Auch für ein ESPRIT-Projekt, das die CIM-orientierte Modellierung von Industriebetrieben anstrebt, wird die Berücksichtigung dieser Ebene als wesentlich für die erfolgreiche Vermarktung von CIM-Systemen herausgestellt (Kosanke/Vlietstra 1989). Darüber hinaus drückt sich darin mitunter eine Kritik daran aus, daß zur Darstellung von Informationssystemen gewöhnlich vorrangig technische Konzepte verwendet werden, also die Sichtweisen und damit Interessen der meisten Anwender vernachlässigt werden. Müller-Merbach (1989, S. 1042) fordert in diesem Zusammenhang "komprehensive Informationssysteme", die von "mündigen Anwendern" mitgestaltet werden.

In einigen Ansätzen wird zudem ein Aspekt betont, der in der Betriebswirtschaftslehre eine lange Tradition hat:

Die Verwendung von Unternehmensmodellen zur Durchführung von Organisationanalysen und zur Unterstützung der organisatorischen Gestaltung

Konzeptuelle Modellierung ist zumeist daran orientiert, einen Realitätsausschnitt mit Hilfe bestimmter Darstellungsmittel - in der Regel unter Vernachlässigung einzelner Aspekte - möglichst authentisch zu beschreiben. Die Gestaltung und Einführung neuer Informationssysteme ist aber häufig mit einer Revision aktueller Organisationsstrukturen und Geschäftsprozesse verbunden. Die Modellierung dieser Zusammenhänge veranschaulicht Gestaltungsalternativen und liefert Kriterien zu deren Evaluation. Bei hinreichender Formalisierung sind zudem Simulationen möglich. Nach Katz (1990, S. 512) ist die Durchführung solcher Analysen im Vorfeld der Einführung neuer Informationssysteme für viele Kunden eines großen Anbieters wesentliches Motiv für den Entwurf von Unternehmensmodellen gewesen. Häufig können die Potentiale komplexer informationstechnologischer Systeme erst durch angemessene organisatorische Änderungen wirksam ausgeschöpft werden. Anders formuliert: Es gibt gute Gründe dafür, das (ohnehin zumeist fiktive) Leitbild der Anpassung von Informationstechnik an bestehende Organisationsformen durch das Leitbild gegenseitiger Anpassung zu ersetzen. Baethge/Overbeck (1986, S. 23) sprechen in diesem Zusammenhang - allerdings aus anderem Blickwinkel - von “systemischer Rationalisierung”:

> “Mit dem Übergang zu Formen systemischer Rationalisierung werden sowohl die Ziele für den Einbezug von Funktionsbereichen in den Prozeß der Technisierung und Automatisierung als auch die Perspektiven für die prozeßhafte Durchsetzung von Konzepten zur Veränderung von Arbeitsabläufen und Bearbeitungsweisen erheblich erweitert.”

Im Zusammenhang mit solchen langfristigen Änderungsprozessen wird mitunter eine weitere Zielsetzung der Unternehmensmodellierung genannt:

Die Unterstützung der strategischen Planung

Hier ist einerseits an die Planung von Informationssystemen zu denken. Das Bemühen um Investitionssicherung legt es nahe, Systeme auf zukünftige Anforderungsänderungen vorzubereiten. Dazu ist es hilfreich, strategische Optionen, ihre Auswirkungen auf betroffene Unternehmensbereiche und vor allem auf das Informationssystem abzubilden. Andererseits kann eben der Einsatz von Informationstechnik selbst neue strategische Potentiale schaffen. Es gibt also gute Gründe für einen Planungsansatz, der durch Modelle gestützt ist, die entsprechend umfangreich sind.[1]

Ein Merkmal schließlich ist besonders charakteristisch für Unternehmensmodelle im Sinne unserer Betrachtung. Zugleich liefert es mehr als alle anderen Anlaß für kontroverse Diskussionen:

1. Vgl. dazu Katz (1990), Eckert (1991) und die Ausführungen unter IV.4.

Der Anspruch auf generelle Gültigkeit

Ein Unternehmensmodell soll danach nicht allein der Abbildung eines spezifischen Unternehmens dienen, sondern vielmehr so generell sein, daß es die Anforderungen einer größeren Zahl von Unternehmen abdeckt - also beispielsweise ein Modell für alle Unternehmen einer bestimmten Branche. In diesem Sinne will Scheer (1991, S. 3) ein von ihm entworfenes Modell "als allgemeingültigen Vorschlag" verstanden wissen. In anderen Beiträgen wird von "reference architecture", "generic design" (ESPRIT-Consortium Amice 1989) oder "Enterprise Business Process Reference Model" (IBM 1990) gesprochen.

Generelle Unternehmensmodelle als Grundlage für den Entwurf und die organisatorische Einbindung betrieblicher Informationssysteme bieten eine ökonomisch ausgesprochen reizvolle Vision. So versprechen sie die vielfache Wiederverwendung von Organisationsentwürfen, Informationssystem-Architekturen und nicht zuletzt von Software. Sorgfältig und kompetent entworfene generelle Konzepte würden damit einem spezifischen Unternehmen die Chance bieten, leistungsfähigere Lösungen zu erhalten, die gleichzeitig mit geringeren Kosten verbunden wären. Darüber hinaus schafft die Verbreitung von Referenz-Architekturen hervorragende Voraussetzung für die zwischenbetriebliche Integration von Informationssystemen.

Die an solche Verheißungen geknüpften Erwartungen werden allerdings durch eine Reihe kritischer Einwände gedämpft. Aus epistemologischer Sicht kann darauf hingewiesen werden, daß die Suche nach generellen Konzepten ein im Wortsinn selbstverständliches Anliegen wissenschaftlicher Untersuchungen sei. Daraus könnte zweierlei geschlossen werden. Erstens: Die Betonung einer entsprechenden *Orientierung* ist müßig. Zweitens: Wenn diese Orientierung allerdings zu einem Forschungs*ziel* wird, gibt es ernsthafte Zweifel an der Machbarkeit. Die Geschichte der Wirtschaftswissenschaften im allgemeinen, die der Betriebswirtschaftslehre im besonderen zeigt, daß generelle Beschreibungen von Unternehmen entweder an der Varianz (und Kontingenz) realer Erscheinungsformen scheitern oder aber durch einschränkende Abstraktionen und ceteris paribus Klauseln den Anwendungsbezug verlieren.[1] Ein solcher Einwand ist gewiß ernst zu nehmen, bedeutet aber nicht notwendig, daß generelle Unternehmensmodelle eine naive Fiktion sind. So müssen Unternehmensmodelle und die auf ihnen basierenden Architekturen und Programme zwar einen hohen Anwendungsbezug aufweisen, sie sind aber nicht allumfassende Modelle oder gar Theorien der Unternehmung. Vielmehr kann der Fokus auf solche Sachverhalte eingeschränkt werden, die eine formale Beschreibung erlauben - ihnen kommt bei der Abbil-

1. Die Diskussion über Sinn und Zweck von Modellen in den Wirtschaftswissenschaften hat eine lange Tradition. Vgl. dazu Albert (1960), Bretzke (1980) sowie die Beiträge in Schmidt/Schor (1987).

dung auf Informationssysteme besondere Bedeutung zu. Es ist zudem wichtig, daß Unternehmensmodellierung in dem hier betrachteten Kontext weniger auf Erklärung als auf Gestaltung gerichtet ist. Auch wenn erfolgreiche Gestaltung ohne sorgfältige Erfassung realer Gegebenheiten kaum möglich ist, so kann doch die dabei festgestellte Varianz durch die Einführung eines (akzeptierten) Referenzmodells erheblich reduziert werden. Tatsächlich gibt es eine Reihe von Belegen für einen solchen Wirkungszusammenhang. Hier ist beispielweise an das betriebliche Rechnungswesen oder mehrfach verwendbare Anwendungssoftware (häufig irreführend Standardsoftware genannt) zu denken. Darüber hinaus muß ein generelles Modell nicht von allen Anwendern in gleicher Form übernommen werden: Durch Modifikationen können allzu große Abweichungen von unternehmensspezifischen Anforderungen ausgeglichen werden. Darüber hinaus kann Varianz natürlich mit dem Kreis der Unternehmen reduziert werden, für die ein gemeinsames Modell gestaltet wird. Selbst wenn ein Modell nur für die Unternehmen eines Teils einer Branche brauchbar ist, können mit der Einführung eines solchen Modells (und der darauf basierenden Informationssystem-Architektur) erhebliche Einsparungen verbunden sein.

Aber selbst wenn generelle Unternehmensmodelle (mit welcher Reichweite auch immer) realisiert werden können, sind sie - so könnte ein weiterer Einwand lauten - überhaupt wünschenswert? Dabei ist vor allem an Auswirkungen auf den Wettbewerb zu denken. So werden gerade betriebliche Informationssysteme häufig als wichtiges Differenzierungsinstrument gesehen[1], mitunter gar als “competetive weapons” (Wiseman 1985). Andererseits stellt sich die Frage, ob (standardisierte) Informationssystem-Architekturen nicht den Wettbewerb unter den Anbietern von Informationstechnik und damit den technischen Fortschritt behindern. Auch wenn Einwände dieser Art einer gewissen Plausiblität nicht entbehren, sind sie doch m.E. wenig überzeugend. Das liegt vor allem daran, daß isoliert *eine* mögliche Auswirkung auf den Wettbewerb betrachtet wird, während tatsächlich auch andere Wirkungen denkbar sind. So kann ein strategischer Vorteil (oder auch: die Vermeidung eines Nachteils) natürlich auch gerade dadurch entstehen, daß ein vielfach verwendetes Modell adaptiert wird: einerseits durch die damit unter Umständen verbundenen Kostensenkungen, andererseits durch die verbesserten Möglichkeiten zwischenbetrieblicher Integration.

Auch der technische Fortschritt kann durch die Einführung von Referenzmodellen positiv beeinflußt werden. Die Einigung auf solche Modelle würde die Möglichkeit bieten, Entwicklungen in Angriff zu nehmen, die für einen einzelnen Anbieter zu aufwendig oder zu risikoreich sind. Wenn daneben verschiedene Anbieter modellkonforme eigene Implementierungen vermarkten, ist ein besonders wirksamer Wettbewerb zu erwarten, da auf diese Weise die Vergleichbarkeit

1. So in Martiny/Lutz (1989) oder Fischbacher (1986). Vgl. dazu auch IV.4.

verbessert wird und die Herstellerabhängigkeit der Anwender reduziert wird. Insgesamt zeigen die skizzierten Einwände und Erwiderungen, daß eine Diskussion um die Auswirkungen von generellen Unternehmensmodellen auf den Wettbewerb reizvoll sein kann, zum gegenwärtigen Zeitpunkt allerdings gewiß noch verfrüht ist, da sie nur spekulativ geführt werden kann.

Den Anreizen, die mit dem Entwurf von Unternehmensmodellen (wie weit ihr Gültigkeitsbereich auch immer sein mag) verbunden sind, stehen zur Zeit noch erhebliche Herausforderungen gegenüber. Die Modellierung von Unternehmen erfordert die Wahl geeigneter Abstraktionsebenen: Welche Aspekte des Unternehmens sind in welchem Detaillierungsgrad abzubilden? Dazu ist sorgfältig abzuwägen, welchen Zielen ein Unternehmensmodell vorrangig dienen soll. Angesichts der Komplexität des Gegenstands verwundert es wenig, daß erste publizierte Entwürfe (IBM 1990, Scheer 1991, ESPRIT Consortium AMICE 1989) durch voneinander abweichende Differenzierungen gekennzeichnet sind. Diese Vielfalt ist nicht per se ein Problem. Schließlich mag sie ein notwendiger Reflex auf unterschiedliche Anforderungen sein. Unbefriedigend ist aber in jedem Fall der Umstand, daß die Unterschiede sich zum Teil in subtiler Form präsentieren (vgl. dazu die Analyse in III.5).

Es gibt also Bedarf an einem Bezugsrahmen oder wenigstens an brauchbaren Kriterien zur Evaluation von Unternehmensmodellen. Ähnliches gilt für den Einsatz einer Entwurfsmethodologie bzw. - damit zusammenhängend - der Konzeptualisierung der Konstrukte, die zur Modellbildung verwendet werden. In diesem Zusammenhang ist gegenwärtig vor allem von Bedeutung, ob ein objektorientierter oder ein daten-/funktionsorientierter Ansatz gewählt wird. Denkt man an Wiederverwendbarkeit ist zu klären, in welcher Form ein Unternehmensmodell präsentiert werden soll: als natürlichsprachliche oder formale Spezifikation, als konzeptuelles Modell und/oder in Form kompilierbaren Codes? In diesem Kontext stellt sich zudem die Frage, wer Unternehmensmodelle entwickelt, wie sie verbreitet werden könnten und welche Rolle die betroffenen wissenschaftlichen Disziplinen - wie die Betriebswirtschaftslehre, die Informatik und nicht zuletzt die Wirtschaftsinformatik - dabei wahrnehmen könnten.

2. Gegenstand und Gang der Untersuchung

Der kurze Überblick über den Gegenstand dessen, was unter Unternehmensmodellierung verstanden wird, macht deutlich, daß hier ein weites interdisziplinäres Forschungsfeld abgesteckt ist. Damit wird dem Umstand Rechnung getragen, daß die Gestaltung von Informationssystemen im Unternehmensgesamtzusammenhang zu sehen ist. Das heißt nicht nur, daß der Entwurf von Informationssystemen ein fundiertes Verständnis der Unternehmensziele und -abläufe voraus-

setzt[1], sondern auch, daß Informationstechnik selbst neue organisatorische Gestaltungsspielräume und strategische Optionen schafft. Eine gemeinsame Betrachtung der interdependenten Bereiche scheint deshalb bessere Resultate zu versprechen als die Betonung einer bestimmten Sicht. Ähnliches gilt für die wissenschaftliche Forschung: Interdisziplinäre Zusammenarbeit verspricht die synergetische Verknüpfung fortschrittlicher Konzepte der Betriebswirtschaftslehre, der Wirtschaftsinformatik und der Informatik - um die wichtigsten zu nennen.

Ein Forschungsvorhaben, das Unternehmensmodellierung zum Gegenstand hat, sieht sich damit vor einem bekannten, hier aber mit besonderer Prägnanz auftretenden, Dilemma: Auf der einen Seite steht das nicht nur wohlbegründete, sondern den Ansatz konstituierende Bekenntnis zu einer umfassenden Perspektive, auf der anderen Seite die Gewißheit, die gesamte Bandbreite kaum abdecken zu können. In der vorliegenden Schrift wird diesem Umstand durch drei komplementäre Ansätze Rechnung getragen. Auf einer abstrakteren, methodologischen Betrachtungsebene wird die Gesamtsicht und das Zusammenspiel der wesentlichen Teilsichten analysiert. Um einen darauf aufbauenden konkreten Ansatz zur Unternehmensmodellierung im Detail entwickeln zu können, wird der Betrachtungsfokus eingeengt. Schließlich wird ein Vorschlag zur schrittweisen Evolution von Unternehmensmodellen skizziert, der darauf zielt, das dargestellte Dilemma zu überwinden. Zum Vorgehen im einzelnen:

Unseren Ausgangspunkt bildet die Betrachtung von Zielen der Gestaltung und organisatorischen Einführung betrieblicher Informationssysteme. Dabei wird sich zeigen, daß zwei Orientierungen eine herausragenden Bedeutung zukommt: Integration und Wiederverwendbarkeit. Beide Begriffe werden mit vielfältigen Intentionen verwendet. Sie sind nicht nur auf die Gestaltung von Informationssystemen im engeren Sinne anwendbar, sondern weisen auch andere - für eine Theorie der Unternehmensmodellierung bedeutsame - Dimensionen auf. Wir werden diese Dimensionen näher betrachten und untersuchen, durch welche Maßnahmen Integration und Wiederverwendbarkeit gefördert werden können. Dabei wird sich ein Kriterium ergeben, das es einerseits gestattet, verschiedene Intensitätsgrade der Begriffe zu differenzieren, und das andererseits einen Zielkonflikt offenbart, der in ähnlicher Form in der Wissenschaftstheorie seit langem bekannt ist. Auf diese Weise lassen sich - auf einem noch abstrakten Niveau - erste Kriterien für die Beurteilung von Unternehmensmodellen formulieren.

Vor diesem Hintergrund erfolgt die Untersuchung bisheriger Ansätze zur Unternehmensmodellierung bzw. wichtiger Vorläufer. Dazu werden wir zunächst wesentliche Evolutionsstufen der konzeptuellen Modellierung betrachten -

1. Exemplarisch dafür zeigt eine neuere Studie von Lederer/Mendelow (1987), daß DV-Leiter häufig mit Organisationszielen und den Vorstellungen der Unternehmensleitung nicht hinreichend vertraut sind.

ergänzt um die Berücksichtigung wichtiger Einflüsse aus den Bereichen Programmiersprachen und Künstliche Intelligenz. Anschließend werden wir drei dedizierte Ansätze zur Unternehmensmodellierung betrachten. Auf dieser Grundlage wird ein Bezugsrahmen für die Gestaltung von Unternehmensmodellen geschaffen, der auf einem sozialwissenschaftlich geprägten Bild der Unternehmung gründet. Er ist wesentlich eingeschränkt auf den Büro- und Verwaltungsbereich - Eigenarten z.B. des Produktionsbereichs werden also nicht explizit behandelt - und beinhaltet die Beschreibung verschiedener Abstraktionsebenen - hier auch Perspektiven genannt - und ihrer Interdependenzen. Vor diesem Hintergrund werden eine Entwurfsmethodik und eine daran orientierte objektorientierte Entwicklungsumgebung entworfen. Sie besteht aus drei integrierten Teilwerkzeugen, die die Beschreibung statischer Objektmodelle, von Vorgangsmodellen und - zur Einbeziehung einer globaleren Sicht - von Unternehmenstrategien erlauben. Die wesentlichen Unterschiede gegenüber gegenwärtig verbreiteten Analyse- und Entwurfsmethoden sowie den damit korrespondierenden CASE-Werkzeugen sind dabei:

- Durch die Verwendung der gleichen anwendungsnahen Konstrukte für Analyse, Entwurf und Implementierung können die vielfältigen Friktionen zwischen Spezifikation und Implementierung erheblich reduziert werden.
- Eine konsequente Objektorientierung unterstützt die semantische Modellierung von Informationen, die in der konzeptuellen Datenmodellierung in der Regel unberücksichtigt bleiben, deren Bedeutung allerdings tendenziell zunimmt: Dokumente, Grafiken, Bilder und dergleichen.
- Die Integration eines Benutzerinteraktionsmodells erlaubt die schnelle Realisation lauffähiger Prototypen.
- Die softwaretechnischen Vorzüge des Systems sind kombiniert mit einer Benutzerschnittstelle, die ein Unternehmensmodell aus unterschiedlichen *Perspektiven* und in variierenden Detaillierungsgraden zu betrachten gestattet. Dazu gehören Repräsentationsformen, die auch für Anwender ohne Informatik-Hintergrund geeignet erscheinen.
- Die Möglichkeit, Entwurf und Implementierung auf einem hohen semantischen Niveau durchzuführen, verspricht eine erheblich gesteigerte Integrität.
- Das System beinhaltet Verfahren zur Analyse alternativer Formen der Gestaltung von Vorgängen und der Formulierung von Unternehmensstrategien.

Das vorgeschlagene Konzept multiperspektivischer Modellierung und die einen konkreten Ausschnitt desselben widerspiegelnde Entwicklungsumgebung dienen auch dazu, zu veranschaulichen, wie sich die Evolution von Unternehmensmodellen zu generellen Referenzmodellen vollziehen könnte. Dabei wird besonderes Augenmerk auf die zu beteiligenden wissenschaftlichen Disziplinen gelegt - nicht zuletzt auf die (Moderatoren-) Rolle der Wirtschaftsinformatik.

II. Integration und Wiederverwendbarkeit als wesentliche Orientierungen

"Das ganze Problem zwischen Theorie und Wirklichkeit schrumpft auf die Frage nach Übersetzung bzw. Abbildung einer Sprache in eine andere Sprache zusammen."

Dieter G. Schneider

"The fact is that all semantics compromise reuse."

Ian Graham

Aus der Sicht der Unternehmensleitung ist es naheliegend, die Gestaltung, Nutzung und Pflege von Informationssystemen vorrangig auf ein Ziel auszurichten: Wirtschaftlichkeit. Aber so schwierig es schon ist, eine solche Wirtschaftlichkeit ex post zu messen - schließlich sind dazu auch angemessene Bewertungen des Nutzens von Informationen nötig, so unzureichend ist Wirtschaftlichkeit als operationale Orientierung für die Gestaltung von Informationssystemen. Im Laufe der Zeit hat sich allerdings eine Reihe von Maßnahmen und zugehörigen Evaluationskriterien herausgebildet, die jeweils - bei dichotomischer Betrachtung - vorrangig Nutzen- oder Kostenaspekte behandeln. Ansätze zur Förderung des Nutzens von Informationssystemen sind hauptsächlich auf die Unterstützung organisatorischer Prozesse und einzelner Arbeitsplätze gerichtet. Hier ist unter anderem an Informations- und Kommunikationsbedarfsanalysen sowie an Maßnahmen zur Verbesserung der Ergonomie von Benutzerschnittstellen zu denken. Auf der anderen Seite ist vor allem das Software-Engineering darauf konzentriert, die Kosten der Software-Entwicklung und -wartung zu senken - indem einerseits der Entwicklungsprozeß rationalisiert wird und andererseits, durchaus damit zusammenhängend, Kriterien für die Qualität von Software vorgegeben werden.

Zur Begründung zweckmäßiger Gestaltungsorientierungen werden wir im folgenden nicht versuchen, die den genannten Ansätzen eigenen Zielsetzungen zu rekonstruieren. Stattdessen gehen wir zunächst davon aus, daß Integration und Wiederverwendbarkeit solche Orientierungen sein könnten. Die Untersuchung dieser Konstrukte wird dann zeigen, daß - bei entsprechend weiter und differenzierter Begriffswahl - Integration und Wiederverwendbarkeit andere speziellere Gestaltungsrichtlinien im wesentlichen abdecken. Dabei wird deutlich werden, daß beide auf das Engste miteinander verknüpft sind - gewissermaßen zwei Seiten einer Medaille. Es wird sich allerdings auch zeigen, daß eine solche doppelte Orientierung allein kein Königsweg ist: Es handelt sich hier um komplexe Sachverhalte, deren Beziehung zu Wirtschaftlichkeit nicht nur komplementär sein muß.

1. Integration

Gleichgültig, ob man wissenschaftliche oder an Anwender gerichtete Veröffentlichungen betrachtet, Forschungsprogramme oder Verlautbarungen von Anbietern: Kaum ein anderer Begriff wird so oft genannt wie Integration, um eine wünschenswerte Eigenschaft von Informationssystemen zu artikulieren[1]. Auch wenn ein solch inflationärer Gebrauch des Begriffs mit einer Vielfalt unterschiedlicher Assoziationen einhergeht, wurde seine Bedeutung dadurch nicht diskreditiert. Vor dem Hintergrund des Anspruchs, Informationssysteme nicht isoliert, sondern als Unternehmensbestandteil zu betrachten, empfiehlt es sich, auch andere in diesem Zusammenhang bedeutsame Dimensionen von Integration zu berücksichtigen. Nach einer Darstellung dieser Dimensionen werden wir den Blickwinkel zunächst auf die Integration von Informationssystemen einengen, um zu den wesentlichen Merkmalen und Voraussetzungen zu gelangen. Die anschließende Betrachtung der Verwendung des Integrationsbegriffs in der Organisationstheorie wird nicht nur essentielle Gemeinsamkeiten sichtbar machen und uns erste Aufschlüsse für die Gestaltung von Unternehmensmodellen liefern, sondern auch Indizien für mögliche negative Wirkungen von Integration aufzeigen.

1.1 Dimensionen von Integration

Nach alltagsweltlichem Verständnis ist der Begriff Integration[2] unter anderem verbunden mit Sinngehalten wie “Einfügung von Teilen in ein Ganzes”, “Überwindung, Ausräumung von Friktionen”, aber auch “Vermittlung” und “Harmonie”. Wenn man im Hinblick auf Probleme und Anforderungen der Entwicklung, Wartung und Nutzung von Informationssystemen an dieses begriffliche Raster anknüpft, lassen sich verschiedene Dimensionen von Integration unterscheiden. In der folgenden Übersicht[3] werden diese Dimensionen kurz beschrieben, um erste Hinweise auf charakteristische Merkmale (oder wohl eher: wichtige Fragestellungen) zu erhalten.

Die Verbindung der Komponenten eines computergestützten Informationssystems

Die Bedeutung von Integration in dieser Dimension wird in der Praxis vor allem durch die Evolution von DV-Systemen in der Vergangenheit und die daduch ver-

1. Damit soll nicht darüber hinweg getäuscht werden, daß Integration manchem DV-Leiter als Reizwort erscheint. Dabei handelt es sich allerdings häufig um eine Reaktion auf Probleme beim Versuch, Integration zu realisieren.
2. Es ist an dieser Stelle nicht nötig, zwischen dem Prozeß des Integrierens und Integration als Zustand eines Systems zu unterscheiden.
3. Es ist damit gewiß kein Anspruch auf Vollständigkeit erhoben. Je nach Differenzierungsgrad der Betrachtung lassen sich mehr oder weniger Dimensionen identifizieren. Zudem gibt es durchaus Überschneidungen, das Verhältnis der Dimensionen ist also nicht immer orthogonal.

ursachten Unzulänglichkeiten geprägt: Einzelne Komponenten (in der Frühzeit der DV vor allem Anwendungssoftware, mit der Verbreitung von Mikrocomputern auch Betriebssysteme und Hardware) wurden ohne hinreichende Berücksichtigung bereits existierender Komponenten installiert. Integrationsfördernde Maßnahmen sind zumeist durch das Ziel der Datenintegration geprägt: Verschiedene und unter Umständen verteilte Anwendungen nutzen gemeinsame Datenbestände - mit den bekannten Vorteilen für die Systemkonsistenz. Abstrahiert man von diesem speziellen Integrationsaspekt läßt sich Integration wesentlich kennzeichnen durch die Aufgabe, Kooperationsfähigkeit zwischen verschiedenen Komponenten eines Informationssystems herzustellen. Die Frage nach Formen und Qualitäten von Kooperation wird in II.1.2.2 eingehend behandelt.

Die Überwindung von Friktionen zwischen verschiedenen Phasen des Lebenszyklus von Informationssystemen

Im Laufe ihrer Entstehung werden Informationssysteme mit Hilfe unterschiedlicher Repräsentationsformen beschrieben. Tendenziell ist der Prozeß von der Analyse bis zur Implementierung durch einen wachsenden Formalisierungsgrad gekennzeichnet. Die Übergänge von einer Stufe zu einer anderen (beispielsweise von einem Pflichtenheft zum Entwurf) sind dabei zumeist mit einer zweifachen Veränderung des jeweils repräsentierten Informationsgehalts, der Semantik, verbunden: Einerseits wird von bestimmten Sachverhalten der vorhergehenden Stufe abstrahiert - diese Information geht verloren - andererseits kann mit fortschreitender Formalisierung auch eine restriktivere Interpretation und damit das Hinzufügen von Semantik verbunden sein. Damit ist - auf einem abstrakten Niveau - die Problematik der Unterstützung des Entwicklungsprozesses skizziert: Die Transformationen zwischen den Phasen sind häufig nicht vollständig automatisierbar. Auf der einen Seite steht die Schwierigkeit, informale Beschreibungen in formale zu überführen, auf der anderen Seite das Problem, daß der Verlust von Semantik entweder in späteren Stufen wieder mühsam rekonstruiert werden muß oder aber die operationale Integrität des Informationssystems verringert.

Seit langem zielen Software-Entwicklungswerkzeuge darauf, die dargestellten Schwierigkeiten zu mindern. Während solche Werkzeuge früher vor allem auf einzelne Repräsentationsformen bzw. Transformationsstufen gerichtet waren, sind jüngere Ansätze, zumeist als Computer Aided Software Engineering (CASE) bezeichnet, dadurch gekennzeichnet, eine phasenübergreifende Unterstützung zu bieten. An dieser Stelle sei nur kurz darauf hingewiesen, wie mit Hilfe von CASE die verschiedenen Phasen besser integriert werden sollen. Einerseits zielt man darauf, Konstrukte bereitzustellen, die - wenn auch in unterschiedlichem Detaillierungs- und Formalisierungsgrad - in allen Phasen verwendet werden können. Auf der anderen Seite sind Maßnahmen zu sehen, die die refe-

rentielle Integrität zwischen diesen Konstrukten auf den verschiedenen Ebenen sichern sollen: Veränderungen an einer Stelle (etwa an einem Datenmodell) sollten dananch entsprechende Modifikationen an davon betroffenen anderen Stellen hervorrufen (beispielsweise an einem Datenbankschema), zumindest aber sollte der Entwickler von den Auswirkungen einer Veränderung unterrichtet werden. Ein weiterer (nur teilweise durch Werkzeuge gestützter) Aspekt ist die *Koordination* der Arbeit der Entwickler. Hier ist unter anderem an die Vermeidung von Doppelarbeit und die Einführung von *Konventionen* zu denken.

Das Zusammenführen verschiedener Rollen und Perspektiven

In einer Phase ist die angemessene Interpretation von einfließender Information besonders schwierig: Während der Analyse werden fragmentarische, zum Teil widersprüchliche Anforderungen in mehrdeutiger Weise formuliert. Für den anschließenden Entwurf sind sie zu einer möglichst konsistenten Gesamtbeschreibung zusammenzuführen. Dabei ist einerseits zu berücksichtigen, daß die Beteiligten die Unternehmung - oder deutlicher: den für sie bedeutenden Ausschnitt - in unterschiedlicher Weise konzeptualisieren, andererseits daran, daß mit der Gestaltung von Informationssystemen variierende Ziele und Interessen verbunden sind. Wollnik (1986, S. 13 ff.) spricht hier von *Perspektivendifferenzen*. Bei der Vermittlung zwischen den verschiedenen Perspektiven spielen *Moderatoren* eine zentrale Rolle.

Auch Prototyping hat hier eine wichtige Funktion. Es zielt darauf, auch denen, die mit Informationstechnik nicht vertraut sind, eine anschauliche *Vorstellung* möglicher Nutzungsformen zu bieten, um ihnen eine wirksame Beteiligung zu ermöglichen. Anders formuliert: Mit einem Prototypen wird ein *Medium* geschaffen, das es erlaubt, über eine gemeinsame Orientierung zu reden. Ähnliche Ziele sind mit Darstellungstechniken zur Anleitung und Dokumentation der Anforderungsanalyse verbunden. Für unser Thema ist ein Ansatz besonders erwähnenswert: Die “Soft Systems Methodology” (Checkland 1981, 1987) ist darauf gerichtet, die Wahrnehmungswelt der Anwender mit Hilfe sogenannter “rich pictures” zu erfassen und zu veranschaulichen. Dabei wird die Bedeutung solcher Darstellungen als Medium nachdrücklich betont: “Die “weiche” Tradition sieht Systemmodelle also als relevant für die Auseinandersetzung mit der Wirklichkeit, nicht als Modelle der Wirklichkeit an.” (Checkland 1987, S. 130).

In diesem Zusammenhang ist nicht zuletzt die Vermittlung zwischen strategischen Zielen, organisatorischer Gestaltung und Systementwicklung von großer Bedeutung. Der Umstand, daß die verschiedenen Beteiligten hinsichtlich der Konzeptualisierung wesentlicher Aspekte unterschiedliche Abstraktionsebenen und Evaluationskriterien (eben verschiedene Perspektiven) pflegen, ist mit erheblichen betriebswirtschaftlichen Konsequenzen verbunden: Hier ist einerseits an die langfristige Sicherung von Investitionen in Informationstechnik zu

denken, andererseits an die Schaffung strategischer Potentiale durch den Einsatz geeigneter Informationssysteme (vgl. dazu beispielhaft Venkatraman 1991).

Die Einbindung von Informationssystemen in organisatorische Handlungsabläufe

Hier handelt es sich um einen zentralen Aspekt der zuvor skizzierten Dimension. Er ist hier deshalb getrennt aufgeführt, weil er nicht nur Wirkungszusammenhänge zwischen bestehenden Handlungskomplexen und zu gestaltender Informationstechnik betont, sondern auch die organisatorische Gestaltung mit einbezieht. Das legt die gemeinsame Betrachtung beider Bereiche in einer Weise nahe, die Gestaltungsoptionen unter besonderer Berücksichtigung des Einsatzes von Informationstechnik verdeutlicht. Ein ebenso anschauliches wie aktuelles Beispiel für derartige Herausforderungen ist der Entwurf von Produktionsanlagen, die nicht nur durch einen hohen Grad an Automatisierung gekennzeichnet sind, sondern auch durch eine enge Verzahnung mit vor- und nachgelagerten sowie parallelen betrieblichen Aktivitäten - verbunden mit Schlagworten wie “concurrent engineering”, “lean production” oder “Computer Integrated Manufacturing”. Im Büro- und Verwaltungsbereich beinhaltet organisatorische Integration - vor allem, wenn man an dispositive Tätigkeiten denkt - einen weiteren Aspekt:

D*ie Abstimmung von DV-Arbeitsplätzen mit individuellen Arbeitsstilen*

Dazu gehört beispielsweise die Berücksichtigung begrifflicher Assoziationen und präferierten Repräsentationsformen. Hier gibt es zu den Themen Management Information Systems (MIS) und Decision Support Systems (DSS) eine Vielzahl von Arbeiten, die allerdings die euphorischen Hoffnungen nicht erfüllen konnten. Dabei ist jedoch einerseits zu berücksichtigen, daß zur Blütezeit dieser Ansätze in den siebziger und frühen achtziger Jahre (Davis 1973, Mason/Mitroff 1973, Keen/Morton 1978) die Informationstechnik weit von der heutigen Leistungsfähigkeit entfernt war (hier ist vor allem an Konzepte und Techniken zur Verbesserung der Ergonomie zu denken), andererseits scheint der Anspruch, “kognitive Stile” von Nutzern in funktionaler Weise zu formalisieren, ohnehin allzu gewagt.[1]

Die zwischenbetriebliche Verbindung von Informationssystemen[2]

Sie verspricht nicht nur die Reduktion von (Transaktions-) Kosten, sondern auch eine Erhöhung der Reaktionsfähigkeit im Sinne einer größeren Kundennähe und damit letztlich: eine verbesserte Wettbewerbsfähigkeit. Hier ist an Schlagworte

1. Eine konstruktiv gewendete Kritik findet sich in Frank (1988), S. 145 ff. Dabei ist zu berücksichtigen, daß beide Begriffe - vor allem aber DSS - auch heute noch Verwendung finden. Die mit neueren Systemen verbundenen Ansprüche und deren softwaretechnische Realisierung haben sich allerdings erheblich gewandelt.
2. Ein Überblick über verschiedene Formen der Integration findet sich in Mertens (1985).

wie Electronic Data Interchange (EDI), Outsourcing, Just-in-Time oder allgemeiner an die überbetriebliche Koordination von Abläufen zu denken.

Die Förderung des Dialogs zwischen den beteiligten Disziplinen
Betriebliche Informationssysteme sind Gegenstand verschiedener wissenschaftlicher Disziplinen. Da die dabei jeweils vorherrschenden Untersuchungsschwerpunkte inhaltlich zusammenhängen, verspricht eine verbesserte Zusammenarbeit Synergie-Effekte. Dieser Aspekt erhält zusätzliches Gewicht dadurch, daß es auch innerhalb einzelner Disziplinen erhebliche Perspektivendifferenzen gibt. So ist die anwendungsorientierte Informatik durch eine Fülle unterschiedlicher Ansätze gekennzeichnet, die zwar selten paradigmatischen Charakter haben (etwa im Sinne der Definition von Kuhn 1969), aber durchaus mit eigenständiger, häufig noch zu profilierender Methode und Zielsetzung antreten. Um einige Beispiele zu nennen: wissenbasierte Ansätze in ihren unterschiedlichen Ausprägungen, objektorientierte Ansätze oder Arbeiten, die unter dem Etikett "Computer Supported Cooperative Work" durchgeführt werden. Bei näherer Betrachtung zeigen sich häufig vielfältige Überschneidungen.

Das - aus wissenschaftssoziologischer Sicht bekannte - Bemühen um Differenzierung und die damit einhergehende Betonung eigener Terminologien, Metaphern und Ausrichtungen machen es dem einzelnen Forscher allerdings sehr schwierig, von Arbeiten in verwandten Gebieten zu profitieren - von wirksamer Kooperation ganz zu schweigen. Die Wirtschaftsinformatik sieht sich hier von einer außerordentlichen Heterogentität herausgefordert: Für ihren Untersuchungsgegenstand sind sowohl diverse Forschungsansätze der Informatik als auch betriebswirtschaftliche Arbeiten von Bedeutung. Darüber hinaus sieht sie sich - in Anknüpfung an das (weithin als solches akzeptierte) Selbstverständnis der Betriebswirtschaftslehre - realen Problemen in der Praxis verpflichtet. Die häufig festzustellenden erheblichen Verständigungsprobleme zwischen Theorie und Praxis kennzeichnen die letzte Dimension in unserer - nicht auf Vollständigkeit beharrenden - Liste:

Die Vermittlung zwischen Forschung und Anwendungspraxis
Hier ist neben der Überwindung von Sprachbarrieren auch an divergierende Interessen und unterschiedliche Randbedingungen zu denken.

1.2 Integrierte Informationssysteme

Die Vielzahl der Dimensionen und die mit ihnen verbundene Komplexität machen es unmöglich, im Detail auf jeweilige Integrationsstrategien einzugehen. Dennoch ist es für unsere Untersuchung wesentlich, die verschiedenen Verheißungen von Integration im Kontext betrieblicher Informationssysteme - sowie ihre Interdependenzen - nicht aus den Augen zu verlieren. Um nähere Auf-

schlüsse über Merkmale und Voraussetzungen von Integration zu erhalten, betrachten wir im folgenden die Frage, welche Anforderungen an die Integration der Komponenten eines Informationssystems zu stellen sind.

1.2.1 Anforderungen

Wenn man die Komponenten eines Informationssystems dadurch kennzeichnet, daß sie Funktionen oder allgemeiner Dienste und Daten bereitstellen, liegt es nahe, eine Trennung in Daten- und Funktionsintegration vorzunehmen - was tatsächlich auch oft getan wird. Wir werden im folgenden nicht völlig von diesen beiden Aspekten abstrahieren, sie allerdings ihrer hohen Interdependenz wegen gemeinsam behandeln.[1] Zunächst lassen sich drei Anforderungen nennen, die unmittelbar mit der Integration der Komponenten verbunden sind.

Verfügbarkeit

Integration nimmt tendenziell in dem Maße zu, in dem einzelne Komponenten auf Dienste und/oder Daten anderer Komponenten zugreifen können - unabhängig davon, wo sie physisch innerhalb des Systems angesiedelt sind. Eine wichtige Voraussetzung dafür ist die kommunikationstechnische Verbindung der Komponenten. Zur Realisation solcher Verbindungen kann zur Zeit auf eine Reihe von Protokollen und darauf basierenden Produkten zurückgegriffen werden - auch wenn die Leistungsfähigkeit dieser Protokolle, wie sich noch zeigen wird, zu wünschen übrig läßt. Es ist denn auch ein anderer Aspekt, der die wesentliche Hürde für Verfügbarkeit im oben skizzierten Sinn markiert[2]. In der Regel gibt es nur sehr eingeschränkte Möglichkeiten, Funktionen, die einzelne Programme bieten, von anderen Programmen aus zu nutzen. So kann man typischerweise die vielfältigen Funktionen, die ein Textverarbeitungssystem zur Formatierung von Dokumenten (für einen Drucker oder für die Darstellung am Bildschirm) bietet, nur über die diesem System eigene Benutzerschnittstelle nutzen. Verfügbarkeit in diesem Sinne ist eng verknüpft mit der seit einigen Jahren populären Forderung nach offenen Systemen. Auf die Spitze getrieben bedeutet sie, daß alle Funktionen (und Daten) eines Programms, die potentiell von anderen Programmen benötigt werden, über geeignete Schnittstellen verfügbar gemacht werden. Es wird sich noch zeigen, ob und - wenn ja - unter welchen Voraussetzungen eine solche Realisierung von Offenheit sinnvoll ist.

Transparenz

Die Möglichkeit, auf Daten und/oder Funktionen zugreifen zu können, ist allein häufig nicht befriedigend. Darüber hinaus ist es wünschenswert, von Details, die

1. Damit erfolgt allerdings kein Vorgriff auf objektorientierte Ansätze.
2. Der Begriff "Verfügbarkeit" wird hier also mit anderer Intention verwendet als im Kontext "Ausfallsicherheit".

für den jeweiligen Verwendungszweck unerheblich sind, abstrahieren zu können. Beispiele für solche Details sind Adressen von Rechnern, auf denen angeforderte Daten oder Dienste abliegen, die interne Darstellung von Daten oder der betriebssystemspezifische Zugriff auf ein Dateisystem. In der Informatik wird in diesem Zusammenhang von Transparenz gesprochen: Diejenigen Teile eines Systems, die die Übersichtlichkeit einer bestimmten Abstraktionsebene beeinträchtigen, sollten transparent im Sinne von durchsichtig und damit: unsichtbar sein.

Semantische Angemessenheit

Hier handelt es sich um eine Eigenschaft, die eng mit Transparenz zusammenhängt, allerdings aus entgegengesetztem Blickwinkel formuliert ist. Während die Forderung nach Transparenz die Vernachlässigung des Unwichtigen betont, fokussiert die Forderung nach semantischer Angemessenheit auf die Frage nach dem jeweils wünschenswerten Bedeutungsgehalt der Funktionen oder Daten (oder allgemeiner: der Schnittstellen), die integriert werden sollen. Folgende Forderung scheint hier sinnvoll: Eine Schnittstelle sollte für die betroffenen Komponenten mit soviel Semantik[1] ausgestattet sein, wie jede Komponente für die weitere Verarbeitung der auszutauschenden Daten benötigen könnte. Anders formuliert: Keine Komponente sollte genötigt sein, die Semantik von Daten mühsam und unter Risiko zu rekonstruieren. Je mehr ein System dieser Forderung genügt, desto höher ist sein Integrationsniveau. So ist es beispielsweise in fortschrittlichen Systemumgebungen möglich, die in einem Tabellenkalkulationsprogramm erstellte grafische Darstellung einer Umsatzstatistik über einen Systempuffer in ein Textverarbeitungssystem zu laden. Schon auf der Ebene des Tabellenkalkulationsprogramms wird in der Regel nicht abgebildet, daß es sich bei den zu verarbeitenden Zahlen um Umsätze handelt - wodurch eine darauf zielende Suche zu einem späteren Zeitpunkt erheblich erschwert wird. Vielmehr beschränkt sich die Semantik auf gegebenenfalls arithmetisch verknüpfte Werte in einer Tabelle, aus denen beispielsweise ein Balkendiagramm erzeugt wird. Bei der Übertragung in ein Textverarbeitungssystem geht dieser Bedeutungsgehalt verloren. Es ist deshalb nicht mehr möglich, weitere Operationen durchzuführen, die eine entsprechende Semantik voraussetzen - also zum Beispiel die Berechnung und grafische Darstellung eines gleitenden Durchschnitts.

Die dargestellte Charakterisierung integrierter Informationssysteme ist eng an die Qualität der Kommunikation zwischen den Komponenten geknüpft. Eine Beschränkung auf diesen Aspekt ist jedoch zu eng, wenn Integration als wesentliche Orientierung für die Gestaltung von Informationssystemen dienen soll: Es gibt noch andere wichtige Kriterien zur Beurteilung der Qualität eines Informati-

1. Zur näheren Charakterisierung des Begriffs Semantik vgl. II.1.2.2.

onssytems, die zudem mit den drei dargestellten Merkmalen in Zusammenhang stehen mögen. Um zu einer unserem Blickwinkel angemessenen Erweiterung bzw. Ergänzung des Integrationsbegriff zu gelangen, werden Wir werden im folgenden Kriterien zur Beurteilung von Software betrachten. Als Orientierung dienen uns dabei entsprechende Richtlinien der ISO/IEC-Norm 9126 (“Information technology - Software product evaluation - Quality characteristics and guidelines for their use”) und von DIN 66 285 (“Anwendungssoftware - Gütebedingungen und Prüfbestimmungen”). Dabei geht es einerseits darum, das Verhältnis solcher Kriterien zu Integration zu betrachten - um ggfs. eine angemessene Erweiterung bzw. Ergänzung unserer vorläufigen Begriffsabgrenzung zu finden, andererseits darum, Hinweise auf weitere konkrete softwaretechnische Maßnahmen zur Förderung von Integration zu ermitteln.

Die folgende Übersicht zeigt die wichtigsten Kriterien. Auffällig ist dabei, daß Integration nicht explizit erwähnt wird:

- Correctness
- Reliability
- Efficiency
- Usability
- Maintainability
- Flexibility
- Reusability
- Portability
- Interoperability
- Integrity

Solche Kriterien, die mit der formalen Korrektheit, der technischen Zuverlässigkeit oder der Performance zusammenhängen (wie “Correctness”, “Reliability” und “Efficiency”), sind für unsere Untersuchung nicht von unmittelbarer Bedeutung. Andere Kriterien wie “Interoperability” oder “Usability” überschneiden sich weitgehend mit den oben dargestellten Merkmalen, enthalten also implizit Integrationsbedingungen. “Reusability” ist Gegenstand von Abschnitt II.2. Damit bleiben Portierbarkeit, Flexibilität, Wartbarkeit und Integrität. Sieht man zunächst von Integrität ab, sind die anderen drei Kriterien durch zwei gemeinsame Anforderungen gekennzeichnet: Anschaulichkeit (des Codes und der Dokumentation) sowie Modularität. Modularität aber steht in unmittelbarem Zusammenhang mit Integration: Integration wie auch Modularität implizieren das Vorhandensein von Komponenten, die irgendwie miteinander kommunizieren müssen. Beide Begriffe werden häufig mit unterschiedlicher Intention verwendet. Auf der einen Seite steht das Zusammenfügen von Vorhandenem im Vordergrund, auf der anderen Seite das Gestalten der Komponenten. Der letzte

Aspekt bleibt in der oben durchgeführten Begriffsbetrachtung weitgehend unberücksichtigt, obwohl beim Entwurf von Informationssystemen beide Aspekte kaum zu trennen sind. Das Merkmal Modularität ist mit einer generellen Richtlinie für die Unterteilung des Gesamtsystems in Module verbunden, die denn auch eine wichtige Erweiterung des Integrationsbegriffs darstellt. Sie besagt, daß die einzelnen Module möglichst unabhängig voneinander sein sollten und jedes für sich exklusiv mit potentiellen Änderungen des abgebildeten Realitätsausschnitts korrespondieren sollte.

Integrität bzw. Konsistenz stellt eine der wichtigsten Anforderungen an Informationssysteme dar. Die Betrachtung der oben genannten Normen und Richtlinien ist hier wenig aufschlußreich: Auch wenn Zugriffskontrolle ohne Zweifel wichtig ist, beinhaltet sie doch allenfalls ein schwaches konstruktives Prinzip. Dessen ungeachtet ist seit langem bekannt, daß der Umgang mit Redundanz von wesentlicher Bedeutung für die Integrität eines Systems ist. Dabei heißt Redundanz traditionell vor allem Datenredundanz. Je mehr sie vermieden wird, desto höher die Chance, daß ein System im Zeitverlauf konsistent bleibt.[1] Ein Blick auf die oben erwähnte Anforderung Modularität macht deutlich, daß auch Funktionsredundanz vermieden werden sollte: Einzelne Änderungen in der Anwendungsdomäne sollten eben möglichst wenige Module betreffen - was die Wartbarkeit und damit die Integrität im Zeitverauf fördert. Eine wesentliche Maßnahme zur Vermeidung von Redundanz - sowohl bei Daten als auch bei Funktionen - ist Generalisierung. Sie wird in den folgenden Abschnitten eingehend behandelt.

Wir können also zunächst feststellen, daß integrierte Informationssysteme durch folgende Eigenschaften gekennzeichnet sind:

- *Verfügbarkeit*
- *Transparenz*
- *Semantische Angemessenheit*
- *Modularität*
- *Minimale Redundanz*

1.2.2 Semantische Referenzsysteme und Integrationsstufen

Die bisherige Erörterung konzentrierte sich vor allem auf Anforderungen, die mit dem (softwaretechnischen) Entwurf integrierter Informationssysteme verbunden sind. Diese Anforderungen legen zwei Fragen nahe:

1. Um präsizer zu sein, könnte man hier auch von Vermeidung unkontrollierter Redundanz sprechen. Schließlich gibt es durchaus Formen *kontrollierter* Redundanz, die Integrität fördert: Kopien, die der Erhöhung des Sicherheitsniveaus dienen.

1. Welches sind die methodologischen/konstruktiven Maßnahmen, die zur Erfüllung der Anforderungen beitragen? Oder anders formuliert: Gibt es jenseits der genannten Anforderungen noch grundsätzlichere Voraussetzungen für Integration?
2. Wie der Integrationsbegriff selbst sind auch die Anforderungen in einer Fülle unterschiedlicher Ausprägungen denkbar. Ein Ansatz, der auf die Förderung von Integration zielt, sollte Maße oder wenigstens Hinweise darauf beinhalten, wie zwischen verschiedenen Integrationsgraden unterschieden werden kann. Wie könnten entsprechende Kriterien aussehen?

Ein Ansatzpunkt für die Beantwortung der ersten Frage wurde im vorherigen Abschnitt genannt: Das Bemühen um Generalisierung wird gemeinhin als wichtigste Voraussetzung zur Vermeidung von Redundanz (und damit eben zur Förderung von Integration) angesehen. Generalisierung ist ein mächtiges heuristisches Prinzip. Es ist natürlich nicht nur für den Entwurf von Informationssystemen von Bedeutung. Darüber hinaus ist es ein geradezu konstitutives Merkmal wissenschaftlicher Forschung schlechthin. Auch wenn es für den weiteren Verlauf unserer Untersuchung wichtig ist, diesen Zusammenhang nicht aus den Augen zu verlieren, beschränken wir uns zunächst auf den engeren Kontext von Informationssystemen. Generalisierung ist hier mit zwei Anforderungen verknüpft. Die eine betrifft die unmittelbare Parallele zur Forschung und zielt auf das Erkennen genereller Konzepte in einem Realitätsausschnitt. Die andere zielt darauf, solche Konzepte konstruktiv umzusetzen.

Das Erfassen genereller Konzepte vollzieht sich wesentlich unter zwei Randbedingungen, die nicht unabhängig voneinander sind: Zum einen sind die für die Repräsentation der Konzepte verfügbaren Konstrukte zu berücksichtigen, zum anderen sind dabei die Richtlinien zur Bildung von Modulen von Bedeutung. Die Leistungsfähigkeit softwaretechnischer Konstrukte (vor allem: Daten, Funktionen und Objekte) wird in III.2.4 und III.3.5 näher betrachtet. Module können (wie auch immer sie softwaretechnisch realisiert sind) als Abbildung genereller Konzepte betrachtet werden. Bei der Betrachtung der Sachverhalte eines Realitätsausschnitts ist demnach nicht nur nach solchen Merkmalen zu suchen, die für eine Menge realer Ausprägungen gleich sind. Gleichzeitig ist darauf zu achten, daß bei Änderungen im Zeitverlauf möglichst alle Exemplare eines Konzepts in gleicher Weise betroffen sind. Und schließlich: Durch solche Änderungen sollte die Gültigkeit anderer Konzepte (und damit die Brauchbarkeit der entsprechenden Module) möglichst nicht beeinträchtigt werden. Es liegt auf der Hand, daß nicht zuletzt das Erkennen und die Beschreibbarkeit von Ähnlichkeiten von erheblicher Bedeutung für das Gelingen dieses Unterfangens ist. Für die Erfassung von geeigneten Konzepten in diesem Sinne bleibt im wesentlichen der Verweis auf Evaluationsstrategien wie kritischer Diskurs und empirische Überprü-

fung - angesichts der Komplexität einer solchen Aufgabe ein wenig befriedigender Hinweis.

Die Implementierung genereller Konzepte (etwa in Form von Datenstrukturen und Funktionen) ist grundlegende Voraussetzung für die Schaffung integrierter Informationssysteme. Allerdings sind dazu noch einige Anforderungen zu berücksichtigen. So sind die Konzepte (genauer: die sie implentierenden formalen Konstrukte) für alle zu integrierenden Komponenten des Systems mit einem eindeutigen Namen zu kennzeichnen. Außerdem muß die Semantik der Konstrukte systemweit in eindeutiger Weise festgelegt sein. Auf diese Weise entsteht ein *gemeinsames semantisches Referenzsystem*. Sein Beitrag zur Integration ist offensichtlich: Komponenten, die miteinander kommunizieren oder kooperieren wollen, können dies mit geringerem Aufwand und zugleich auf einem höheren Sicherheitsniveau tun, wenn sie dazu in eindeutiger Weise auf Konstrukte verweisen können, deren Bedeutung systemweit bekannt ist. Im Hinblick auf die Integrität eines Systems im Zeitverlauf ist es wichtig, daß solche semantischen Referenzsysteme mit einem hohen Maß an Verbindlichkeit und Verläßlichkeit eingeführt werden. Verbindlichkeit betrifft dabei auch die Systementwickler: Sie sind angehalten, wenn möglich die gemeinsamen Referenzen zu nutzen.

Semantische Referenzsysteme können in unterschiedlicher Form realisiert werden. Datentypen oder die Funktionen eines Betriebssystems können ebenso Elemente eines semantischen Referenzsystems sein wie die Relationstypen eines Datenbankmanagement-Systems oder eine Compound Document Architecture (CDA), die den Aufbau von Dokumenten beschreibt. Wie sollten solche Elemente gestaltet sein, um die Integration eines Informationssystems besonders zu fördern? Es ist naheliegend, davon auszugehen, daß das Integrationsniveau eines Informationssystems mit der Semantik seiner Elemente zunimmt. In dem vorliegenden Kontext können wir Semantik dabei im Sinne des logischen Positivismus definieren. Danach nimmt die Semantik einer Repräsentation mit einer abnehmenden Zahl zulässiger Interpretationen zu.[1] Eine Instanz eines elementaren Datentyps (wie eine Festpunktzahl) kann eine Fülle ganz unterschiedlicher realer Sachverhalte repräsentieren - entsprechend gering ist sein Bedeutungsgehalt. Demgegenüber ist die Zahl möglicher Interpretationen einer größeren Datenstruktur - wie etwa eines Artikelstammsatzes - wesentlich geringer.

Die Beispiele machen deutlich, daß sich Semantik in der Regel allenfalls komparativ erfassen läßt. Doch nicht genug mit dieser Einschränkung: Bei näherer Betrachtung zeigt sich, daß die Forderung nach möglichst viel Semantik für die

1. Aus der Sicht des Kritischen Rationalismus ergibt sich - allen Unterschieden zu positivistischen Positionen zum Trotz - ein ähnliches Bild. So steigt für Popper der Informationsgehalt einer Theorie mit der Zahl der denkmöglichen Fälle, die sie ausschließt.

Elemente eines Referenzsystems weiterer Differenzierung bedarf. Wie bereits in II.1.2.1 angedeutet, ist ein Anreichern mit Semantik nur sinnvoll, wenn es pragmatisch relevant ist. Um ein Beispiel zu geben: Während es für ein Textverarbeitungsprogramm für mögliche weitere Verarbeitungen relevant sein kann, daß eine Zeichenkette als Telefonnummer gekennzeichnet ist, ist dies für einen Drukker, der ein Dokument mit dieser Zeichenkette ausdruckt, völlig unerheblich. Gleichzeitig sind für den Drucker andere Details von Bedeutung, die im Textverarbeitungsprogramm keine Rolle spielen.[1] Diesem Umstand wird bekanntlich dadurch Rechnung getragen, daß ein Informationssystem in verschiedene Schichten unterteilt wird. Dabei sollen die Schichten - wie die bereits erwähnten Module - möglichst unabhängig voneinander sein und mit je unterschiedlichen Randbedingungen korrelieren.[2] Ein integriertes Informationssystem weist also nicht ein einziges semantisches Referenzsystem auf, vielmehr hat jede Schicht ein eigenes. Die Elemente eines schichtspezifischen Referenzsystems sollten der Forderung nach semantischer Angemessenheit genügen.

Im Hinblick auf das Integrationsniveau ist es weiterhin wichtig, daß ein semantisches Referenzsystem möglichst vollständig ist. Vollständigkeit bedeutet im Extremfall, daß sämtliche Gemeinsamkeiten von zwei oder mehr Komponenten als Bezugspunkt in das Referenzsystem aufgenommen werden. Ein weiterer Indikator für Integration ist der Nutzungsumfang. Hier sollte gelten, daß die Komponenten einer Schicht untereinander nur solche Nachrichten austauschen, die allein auf das schichtspezifische Referenzsystem verweisen. Sonst wird - wie es gängige Praxis ist - gegen das Prinzip der Unabhängigkeit der Schichten verstoßen. So greifen Anwendungssysteme häufig auf Funktionen des jeweiligen Betriebssystems zu. Solche Verweise können inhaltlich durchaus angemessen sein, sie sollten allerdings indirekt erfolgen - also auf ein entsprechendes Element des semantischen Referenzsystems der jeweiligen Schicht, das dann die Schnittstelle zu den Elementen einer anderern Schicht realisiert.

Der Diskussion der verschiedenen Faktoren, die das Integrationsniveau eines Informationssystems bestimmen, macht deutlich, daß es sich hier um ein hochkomplexes Problem handelt. Die dargestellten Maßnahmen zur Förderung von Integration dienen dazu, grundlegende Zusammenhänge zu verdeutlichen. Sie liefern zudem eine erste - abstrakte - Festlegung dessen, was man unter der *Architektur integrierter Informationssysteme* verstehen könnte: Ein Gerüst aus semantischen Referenzsystemen (deren Entwurf und Verwendung an den skizzierten Anforderungen orientiert ist), die möglichst unabhängig voneinander sein

1. Brodie (1984, S. 39) spricht in einem ähnlichen Zusammenhang von “semantic relativism”.

2. Anders formuliert: Die durch den Begriff Schicht nahegelegte Vorstellung einer hierarchischen Ordnung ist nicht denknotwendig.

sollten. Aus dieser abstrakten Charakterisierung ergeben sich für unsere Betrachtung zwei wichtige Konsequenzen:

- Es gibt erhebliche Potentiale zur weiteren Untersuchung und Gestaltung integrierter Informationssysteme.
- Die Komplexität der damit verbundenen Herausforderungen übersteigt die (wirtschaftlichen wie fachlichen) Möglichkeiten vieler Entwickler.

1.3 Der Integrationsbegriff in der Organisationstheorie

Die erfolgreiche Implementierung von Informationssystemen ist gewiß nicht allein eine (software-) technische Herausforderung. Die in II.1.2.1 dargestellten Dimensionen von Integration machen deutlich, daß es sich dabei auch um eine organisatorische Aufgabe handelt. Dieser Umstand legt es nahe, einen Blick auf die betriebswirtschaftliche Organisationstheorie zu werfen. Es würde den Rahmen unserer Untersuchung sprengen, wenn dabei sämtliche organisationstheoretischen Arbeiten, die in irgendeiner Form die Implementierung von Informationssystemen berühren, berücksichtigt werden sollten. So gibt es allein im deutschsprachigen Raum eine Fülle empirischer Untersuchungen - ganz zu schweigen von der nordamerikanischen Forschung im Bereich Information Systems. Ähnliches gilt für Methoden zur Unterstützung der Organisationsanalyse. Für unsere Untersuchung ist es an dieser Stelle vorrangig, weitere Aufschlüsse über das Phänomen Integration zu erhalten, das wir bisher vor allem in einem softwaretechnischen Kontext betrachtet haben. Das erlaubt die Beschränkung auf konzeptionelle Arbeiten, die die Grundlagen der Organisationstheorie reflektieren: Integration ist ein wesentliches Merkmal von Organisationen. So kann der Prozeß des Organisierens durchaus als Integration der Interessen und Ziele der Beteiligten angesehen werden. Terminologisch spiegelt sich dieser Stellenwert in der Organisationstheorie allerdings nicht wider. Stattdessen wird - mit leicht geändertem Duktus - zumeist der Begriff Koordination verwendet.

1.3.1 Koordination

Die mit der Arbeitsteilung verbundene mehr oder weniger differenzierende Spezialisierung von Handlungskomplexen in Unternehmen macht eine zielgerichtete Integration bzw. Koordination nötig. Die Untersuchung und Entwicklung geeigneter Koordinationsinstrumente oder -verfahren ist denn auch wesentlicher Bestandteil der betriebswirtschaftlichen Organisationstheorie. Die klassischen Koordinationsmechanismen sind Regelsysteme. In diesem Sinne spricht Grochla (1982, S. 105) von der "Schaffung eines einheitlichen Planungs- und Kontrollsystems" und - an anderer Stelle[1] - von "personenbezogenen" und "maschinenbezo-

1. Grochla (1978, S. 12 ff.)

genen" Regeln. Solche Regelwerke - zu denen auch Anreizsysteme gehören - konstituieren die sogenannte formale Organisationsstruktur.[1] Die zunehmende Betrachtung von Organisationen als soziale Systeme hat zusammen mit dem Wandel von Werten - in Unternehmen wie in der Forschung - eine stärkere Hinwendung zu nicht-formalen Koordinationsmechanismen bewirkt. Damit sind zwei wichtige Konsequenzen verbunden. So wird einerseits die an ingenieurwissenschaftlichen Idealen orientierte Gestaltungsmetapher der frühen Organisationslehre relativiert: Eine Organisation ergibt sich nicht allein durch Verordnung, sie entwickelt und artikuliert sich zu einem erheblichen Teil mittels sozialer Prozesse. Damit einher ging ein Wandel der Interpretation sogenannter informaler Organisation. Wurde sie zunächst als notwendiges oder gar abzuschaffendes Übel betrachtet, wurde später ihr wesentlicher Beitrag zur Funktionsfähigkeit von Organisationen erkannt. Andererseits wird in jüngerer Zeit - in der Organisationstheorie und vor allem in der Beratungspraxis - versucht, informale Organisation zu instrumentalisieren. Prominentestes Beispiel dafür ist das immer noch große Interesse an der sogenannten Unternehmenskultur.

Formale wie informale Koordinationsinstrumente gibt es in einer großen Vielzahl, auf die hier nicht weiter eingegangen werden soll. Für unsere Betrachtung stellt sich vor allem die Frage, in welchem Verhältnis der Koordinations- bzw. Integrationsbegriff der betriebswirschaftlichen Organisationstheorie zu dem oben entwickelten Begriff von Integration als Merkmal von Informationssystemen steht. Die wesentliche Voraussetzung zur Schaffung integrierter Informationssysteme ist die Etablierung gemeinsamer semantischer Referenzsysteme. Die Parallele zu Regelsystemen ist dabei offensichtlich. Auch informale Koordinationsmechanismen wie Kultur sind von den Organisationsmitgliedern geteilte Orientierungsmuster oder Sinnsysteme. So kennzeichnen Deal/Kennedy (1983, S. 501) Kultur als "shared values, heroes and heroines, rituals and ceremonies". Noch deutlicher wird dieser Zusammenhang durch die Definition, die Luhmann (1984, S. 224) vorschlägt: "... eine Art Vorrat möglicher Themen, die für rasche und rasch verständliche Aufnahme in konkreten kommunikativen Prozessen bereitstehen." Habermas (1981, Bd. 2, S. 217) - um die Liste prominenter Zeugen fortzusetzen - betont ebenfalls, in instrumentalistischer Wendung, die Funktion gemeinsamer Referenzsysteme: Er sieht soziale Integration vor allem durch die Vermittlung "intersubjektiv anerkannter Geltungsansprüche" gefördert.

Organisieren zielt also letztlich auch - wie die Integration von Informationssystemen - auf die Schaffung semantischer Referenzsysteme. In diesem Sinne kennzeichnet Weick (1985, S. 11) den Prozeß des Organisierens als "... durch Konsens gültig gemachte Grammatik für die Reduktion von Mehrdeutigkeit ...". Je nach

1. Für Kieser/Kubicek (1983, S. 16) manifestiert sich die Organisationsstruktur in Regeln zur Arbeitsteilung und Koordination.

Größe und Diversifikationsgrad eines Unternehmens gibt es mehr oder weniger solcher Referenzssysteme. Sie artikulieren sich unter anderem in Fachterminologien einzelner Berufsgruppen und/oder in verschiedenen Ziel- und Wertesystemen.

Welche Bedeutung hat die skizzierte Kohärenz für unsere Untersuchung? Es sind vor allem zwei - miteinander zusammenhängende - Implikationen, die zu berücksichtigen sind. So setzt die organisatorische Integration von Informationssystemen voraus, daß es eine Menge von Begriffen gibt, die korrespondierende Sachverhalte (wie Aufgaben, Objekte, Vorgänge und dergleichen) in der Organisation und im Informationssystem bezeichnen. Je größer diese Menge gemeinsamer semantischer Referenzen, desto enger ist ceteris paribus die organisatorische Einbindung des Informationssystems. Eine solche Zuordnung wird allerdings dadurch erschwert, daß integrationsrelevante Sachverhalte in Organisationen häufig mehrdeutig sind. So kann häufig die Preispolitik eines Unternehmens nicht oder nur mit nicht vertretbarem Aufwand vollständig formal beschrieben werden. Je größer die Zahl der Fälle, denen das formale Verfahren der Preisgestaltung nicht genügt, desto schlechter tendenziell die organisatorische Integration. Ein Teil dieser Ambiguität wird in der Regel durch Reorganisationsmaßnahmen, die die Einführung von Informationssystemen erst ermöglichen, reduziert.[1] Darüber hinaus gibt es Mehrdeutigkeiten, die eine wichtige Funktion zur Bewältigung von Komplexität haben und deshalb nicht vermieden werden können. Dieser Umstand erhält noch dadurch zusätzliches Gewicht, daß die Einführung eines Informationssystems organisatorische Realität verändert, was von denjenigen, die mit solchen Systemen als Entwickler oder Nutzer umgehen, in unterschiedlicher Weise wahrgenommen und konzeptualisiert wird: Sie betrachten das Informationssystem und sein tatsächliches oder gewünschtes Wechselspiel mit organisatorischen Handlungskomplexen aus unterschiedlichen *Perspektiven*. In diesem Sinne Wollnik (1986, S. 61):

> "Pragmatische Relevanzen und Erfahrungen steuern die selektiven Bestimmungen. Diese relativieren das Verständnis computergestützter Informationssysteme auf einen jeweils besonderen Standort, den man im Verhältnis zu diesen Systemen einnimmt: eben einen Standort, an dem man im Zusammenhang mit computergestützten Informationssystemen gewisse Handlungsabsichten verfolgt und gewisse Erfahrungen typischerweise machen kann."

Daraus folgt, daß ein semantisches Referenzsystem, das darauf zielt, die organisatorische Integration von Informationssystemen zu fördern, die relevanten Perspektiven berücksichtigen sollte. Die im Referenzsystem verwendeten Konzepte sollten also geeignet sein, zwischen korrespondierenden Konzepten der einzel-

1. Kieser/Kubicek (1983, S. 309) unterscheiden in diesem Zusammenhang zwischen der "Formalisierung des Informationsflusses zwischen Computer und Benutzern" und einer "Strukturformalisierung".

nen Perspektiven zu vermitteln. Ein Mittel dazu ist eine anschauliche Darstellung dieser Konzepte. Anschaulich bedeutet dabei, daß Darstellungsformen gewählt werden, die in anderen Perspektiven bekannt sind und angemessene Assoziationen fördern. Bei allzu großen Perspektivendifferenzen ist an einen anderen Weg zu denken: Das Bemühen um verschiedene Referenzsysteme, die jeweils einige wenig divergierende Perspektiven integrieren (z.B. die Sicht des Vertriebs und des Marketings). Um zwischen solchen partiellen Referenzsystemen zu vermitteln, können einzelne ihrer Konzepte miteinander assoziiert werden. Es erfolgt also keine Integration durch Generalisierung über Konzepte, sondern eine mehr oder weniger enge Kopplung.

Darüber hinaus ergibt sich die Konsequenz, daß die Etablierung organisatorisch integrierter Informationssysteme als evolutorischer Prozeß anzulegen ist. Vordergründig ist diese Feststellung wenig überraschend: Die Erfassung von Anforderungen, ihre Umsetzung sowie anschließende Korrekturen werden in der Regel als Prozeß betrachtet. Wenn man jedoch organisatorische Integration von Informationssystemen weiter denkt, geht es nicht allein um einen technisch orientierten Entwurfsprozeß, zu dessen Randbedingungen das Bemühen um eine möglichst authentische Erfassung sozialer Realität gehört. Vielmehr ist diese Realität selbst - oder besser: die sie konstituierenden Wahrnehmungen und Konzeptualisierungen - Gegenstand eines Prozesses, in dem durch Verhandlung und Lernen die Differenzen zwischen diesen Perspektiven reduziert werden. Mit anderen Worten: Die organisatorische Integration von Informationssystemen sollte nicht allein verstanden werden als ein Problem der angemessenen Anpassung von Informationstechnik an bestehende Realität, sondern als ein sozialer Prozeß, der auf eine kollektive Revision gängiger Konzeptualisierungen unter veränderten Randbedingungen zielt - also auf organisatorischen Wandel.

1.3.2 Mögliche dysfunktionale Effekte von Integration

Nach gängiger Vorstellung in der Organisationstheorie ist Koordination kein Selbstzweck. Sie ist vielmehr notwendige Konsequenz der Spezialisierung, die wiederum ein Reflex auf die heterogenen von außen an das Unternehmen herangetragenen Erwartungen sowie die Komplexität von Leistungserstellungsprozessen ist. Da Koordination bzw. Integration das Zusammenwirken einzelner Handlungen auf die Erreichung der Unternehmensziele hin ausrichten soll, stellt sich die Frage, wie sie zu gestalten ist, um dies in bestmöglicher Weise zu tun. Die frühe Organisationstheorie und die sie begleitende Praxis waren durch den Glauben an die Realisierbarkeit einer optimalen Organisation geprägt. Zu erreichen glaubte man sie durch vollständige Regelung von Handlungen (bei einem hohen Maß an Arbeitsteilung) und mit ihr einhergehender Kontrolle - mit anderen Worten: völlig formalisierte Organisation. Dadurch sollten die durch individuelle

Handlungsspielräume möglichen Unabwägbarkeiten ausgeschlossen werden. Eine solche Sicht spiegelt sich besonders deutlich in den Empfehlungen der ingenieurwissenschaftlich geprägten Organisationslehre angelsächsischer Provenienz[1] wie auch in der Organisationsform Bürokratie. Die in den sechziger Jahren einsetzende sozialwissenschaftliche Orientierung gab die Vision der optimalen Organisation nicht auf, allein der methodische Ansatz war modifiziert. So sollten - vereinfacht dargestellt - die Ziele des Unternehmens wie auch die der Unternehmensangehörigen empirisch ermittelt werden. Eine optimale Koordination wird danach durch die Einführung von Anreizsystemen erreicht, die dafür sorgen, daß das Individuum durch Verfolgung seiner Ziele in bestmöglicher Weise zur Erreichung des Unternehmensziels beiträgt.[2]

Es ist oben bereits skizziert worden, warum die neuere Organisationstheorie die Leitvorstellung von der durch die Gestaltung formaler Regelungen optimierbare Organisation aufgegeben hat und statt dessen die Bedeutung informaler, mehrdeutiger Orientierungen betont: Eine zu starke Formalisierung schränkt die Flexibilität von Unternehmen ein. Vor dem Hintergrund des hier vertretenen Integrationsbegriffs scheint daraus zu folgen, daß eine maximale Integration für Organisationen dysfunktional ist. Denn die Formalisierung von Begriffen, die der Koordinierung dienen, führt zu einer Beseitigung von Mehrdeutigkeiten und damit - ceteris paribus - zu einer semantischen Anreicherung. Diese Interpretation greift allerdings zu kurz. Sie übersieht, daß Formalisierung nicht nur eine Auflösung von Mehrdeutigkeiten mit sich bringt, sondern auch die völlige Ausgrenzung mehrdeutiger Sachverhalte - und damit die Reduktion von Semantik. Sobald man aber die integrierenden Konstrukte informaler Organisation berücksichtigt, ist der Semantikgehalt solcher Konstrukte als Maß für das Integrationsniveau nur noch eingeschränkt brauchbar. Das liegt daran, daß die für Informationssysteme vorgeschlagene Definition von Semantik (vgl. II.1.2.2) an extensionaler Semantik orientiert ist. Im Unterschied dazu sind die Begriffe, Symbole und Metaphern informaler Organisation auch durch intensionale Semantik[3] gekennzeichnet.

---

1. Hier ist vor allem an die Strömung des sogenannten "Scientific Management" zu denken, die durch die Arbeiten von Taylor und Fayol begründet ist.
2. Diese Vorstellung geht zurück auf die von Barnard (1938) und Simon (1949) geprägte "Anreiz/Beitrags-Theorie" und die "Koalitionstheorie" von Cyert/March (1963).
3. In der Betonung intensionaler Semantik (auch als Gegenstand wissenschaftlicher Theorien) artikuliert sich die Überzeugung, daß es kommunizierbare Sachverhalte gibt, deren Sinngehalte nicht vollständig extensional, also durch Verweis auf eine enummerative Liste überprüfbarer Eigenschaften, darstellbar sind. "Die intensionale Beziehung von Gedanken und Erlebnissen auf ihren Gegenstand ist dadurch charakterisiert, daß das worauf diese sich richten, ihnen in gewisser Weise innewohnt - inexistiert." (Habermas 1984, Bd. 1, S. 311). Zum Begriff der Intensionalität vgl. auch von Wright (1974, S. 20) und Searle (1980).

Deren Gehalt läßt sich aber intersubjektiv noch nicht einmal komparativ messen.[1] Die Behauptung, daß eine zu hohe organisatorische Integration kontraproduktiv ist, läßt sich deshalb weder bestätigen noch widerlegen. Sie ist allerdings offensichtlich zutreffend, wenn man allein formale Koordinationsmechanismen betrachtet.

Wenn man diese Überlegungen auf Informationssysteme überträgt, ist zunächst die Verwendung des Begriffs Formalisierung zu relativieren. In Informationssystemen gibt es ohnehin nur formal definierte Integrationskonzepte (in den jeweils verwendeten semantischen Referenzsystemen). Zunehmende Formalisierung in Organisationen kann hier verglichen werden mit einer zunehmenden Spezialisierung der Konstrukte des semantischen Referenzsystems. Im Hinblick auf die Frage nach dem angemessenen Integrationsniveau stößt man dann auf einen bekannten Zusammenhang: Je spezieller solche Konstrukte (also je größer ihr Gehalt an Semantik) desto größer ist die Chance, daß Anforderungsänderungen nicht abgedeckt werden - die Flexibilität nimmt also ab.[2] Aber: So wichtig der Hinweis auf solche unangenehmen Konsequenzen eines erhöhten Integrationsniveaus auch ist, er liefert keine konkreten Aufschlüsse über ein angemessenes oder gar optimales Niveau. Vielmehr ist im Einzelfall zu prüfen, ob die zur Integration verwendeten Generalisierungen geeignet sind, zu erwartende Varianz abzudecken.

Darüber hinaus ist zu berücksichtigen, daß die Erhöhung des Integrationsniveaus (bei großen Systemen) eine komplexe Aufgabe ist. Das impliziert nicht nur einen entsprechenden Aufwand, sondern auch das Risiko, zu scheitern. Dieser Umstand führt in der Praxis der Datenverarbeitung bei manchem Verantwortlichen zu einer zögerlichen bis ablehnenden Haltung gegenüber dem Ziel Integration. In jedem Fall gibt er Anlaß, vor einer unkritischen Integrationseuphorie zu warnen. Die im Rahmen der Theorie sozialer Systeme durchgeführten Analysen über Integration geben allerdings einen Hinweis darauf, daß das Bemühen um Integration letztlich zu einer Verringerung von Komplexität führen kann. So implizieren die mit Integration verbundenen Anforderungen zunächst eine Steigerung von Komplexität. Erfolgreiche Integration führt dann allerdings zu einem "Gewinn an reduzierbarer Komplexität" (Luhmann 1967, S. 109) - durch erhöhte Selektivität. Um ein Beispiel zu geben: Die Einführung von Konstrukten mit einem hohen Maß an Anwendungssemantik (wie etwa "Kunde", "Konto", "Rechnung") steigert die Komplexität des Entwicklungsprozesses zunächst erheblich.

---

1. Dazu müßte man beispielsweise - um eine Metapher von Wittgenstein (1980, §63) - aufzugreifen, ein Verfahren angeben können, das die Semantik verschiedener Beschreibungen des Klangs einer Klarinette zu vergleichen gestattet.
2. Dieser Sachverhalt wird in II.2.2.4 - in Zusammenhang mit Wiederverwendbarkeit - ausführlich behandelt.

Sie resultiert dann aber in einem Informationssystem mit geringerer Komplexität, da seine Teile untereinander differenzierter kommunizieren können. Zugleich präsentiert sich das System dem Betrachter (bei der Wartung oder Weiterenwicklung) weniger komplex, da - in mehr oder weniger großem Umfang - an die Stelle verstreuter Einzelbeschreibungen eine wesentlich geringere Zahl generalisierter Konzepte tritt.

Auch diese abstrakten Überlegungen machen deutlich, daß Integration nicht allein als Gestaltungsziel anzusehen ist, sondern vielmehr eine Orientierung für einen evolutorischen Prozeß liefert.

2. Wiederverwendbarkeit

Es ist ein wesentliches Merkmal von Software, daß sie mit geringem Aufwand vervielfältigt werden kann. Darüber hinaus hat man schon früh erkannt, daß es gewisse Funktionen gibt, die in den meisten Anwendungsprogrammen benötigt werden. Diese Erkenntnis spiegelt sich in der Entwicklung der ersten Betriebssysteme ebenso wie in den Bibliotheken früher Programmiersprachen. Es hat allerdings eine Weile gedauert bis Wiederverwendbarkeit (oder Wiederverwendung - je nach Blickwinkel) explizit als wichtiges Ziel der Software-Entwicklung thematisiert wurde. Ende der sechziger Jahre - die in der Folgezeit mit dem Begriff Softwarekrise assoziierten Probleme drangen in das Bewußtsein einer wachsenden Zahl von Systementwicklern - wurden die reizvollen Eigentümlichkeiten der Produktionsfunktion von Software als Königsweg zur Überwindung der Hemmnisse realer Software-Entwicklung vorgeschlagen[1]: Weg von kostenintensiver Einzelfertigung mit hoher Fertigungstiefe, hin zur Verwendung von Komponenten, die in hohen Stückzahlen produziert zu geringen Kosten verfügbar sind. Trotz der Attraktivität einer solchen Orientierung ebbte die Diskussion bald wieder ab. Erst in den achtziger Jahren wurden die Verheißungen von Software-Wiederverwendung neu entdeckt - was sich in einer Fülle einschlägiger Publikationen[2] dokumentiert. In den letzten Jahren erhielt das Thema Wiederverwendbarkeit zudem durch die Verbreitung objektorientierter Ansätze eine zusätzliche Aufwertung.

Um die Potentiale, die durch Wiederverwendbarkeit eröffnet werden, besser einordnen zu können, werden wir zunächst betrachten, welche Hoffnungen mit dieser Idee verbunden werden. Der anschließende Überblick über die dedizierte Forschung soll deutlich machen, welche Objekte zur Wiederverwendung vorgesehen sind und welche konkreten Ansätze dazu verfolgt wurden. Um der Untersuchung

1. Als Auslöser dieser Diskussion wird häufig McIllroy (1969) genannt.
2. Ein Überblick über einschlägige Aufsätze aus dem gesamten Jahrzehnt findet sich in folgenden Sammelwerken: Biggerstaff/Perlis (1989, 2 Bände) sowie Tracz (1988 a).

nicht vorzugreifen, werden dabei objektorientierte Ansätze nicht explizit behandelt. Schließlich ist der Frage nachzugehen, warum der Umfang von Wiederverwendung bisher so enttäuschend ist. Oder anders gewendet: Was sind die wesentlichen Faktoren, die den Erfolg von Wiederverwendbarkeit bestimmen?

2.1 Visionen und Erwartungen

Die Ausweitung der Wiederverwendung von Software wird häufig mit großen, zum Teil euphorischen Hoffnungen verbunden. So empfiehlt Biggerstaff (1984) Wiederverwendbarkeit als "the essence of design" zu betrachten. Schmid/Uhl (1990, S. 1) sprechen von "one of the most promising issues in today´s arena of software engineering". Cox (1990) glaubt gar, hier die “silver bullets” zur endgültigen Überwindung der Software-Krise zu erkennen. Die einschlägigen Beiträge sind dabei vorrangig durch eine ingenieurwissenschaftliche Perspektive geprägt (bezeichnenderweise findet sich die Mehrzahl dieser Beiträge in IEEE-Publikationen). Dabei steht die technische Erstellung von Software im Vordergrund - mit deutlichen Assoziationen zur industriellen Fertigung, ausgedrückt durch Begriffe wie Software-IC (Cox 1990, Cox 1992) oder Software-Factory (Evans, Kobayashi, ESF). Dementsprechend werden die Defizite der Software-Entwicklung gegenüber der Entwicklung und Fertigung von Hardware (im weitesten Sinn) als ein Ansatzpunkt betrachtet, um zu einer ökonomischeren Erstellung (bzw. Produktion) von Software gelangen. Die folgende Grafik stellt - sicherlich in vereinfachter Form - die Eigenarten von Software denen von Hardware gegenüber:

Attribut	Hardware	Software
Das Produkt ist leicht identifizierbar.	ja	nein
Die Produktdokumentation wird vom Management wenigstens in den Grundzügen verstanden.	ja	nein
Ein "Abnahmetest" liefert ein sehr gutes Maß für die Gesamtqualität.	ja	selten
Der industrieweite Qualitätsstandard ist gut.	ja	nein
Wenn das Produkt schlecht entworfen oder hergestellt ist, erfährt dies jeder.	ja	nicht immer
Eine Fehlervorhersage ist möglich.	ja	kaum
Ein Ersatzteilvorrat ist Teil des Produkts.	ja	nein
Das Produkt kann modularisiert entworfen.	ja	ja

Abb. 1: Vergleich von Hardware- und Software-Eigenschaften, nach Budde/ Schnupp/Schwald (1980), S. 33

Die Zielsetzung ist allerdings nicht allein die Übertragung erfolgreicher Prinzipien industrieller Produktion[1] auf die Software-Entwicklung. Vielmehr setzt man darauf, Mißstände der Software gegenüber Hardware im allgemeinen auszugleichen und gleichzeitig die spezifischen Vorteile der Software-Produktion zu nutzen. Im Vergleich zu traditionellen ingenieurwissenschaftlichen Disziplinen ist das Software-Engineering in den zurückliegenden Jahren durch eine stürmische Entwicklung seiner Einsatzfelder gekennzeichnet. Damit einher geht eine unübersichtliche Vielfalt von mehr oder weniger bewährten Methoden und mit ihnen verbundenen Terminologien sowie - im Resultat - erhebliche Unterschiede in der Qualität der Software. Es liegt deshalb nahe, daß eine Software-Industrie, die auf wiederverwendbaren Komponenten gründet, Aspekte wie Qualitätssicherung und Gewährleistung besonders betont. Aus technischer Sicht wird denn auch die Förderung der Softwarequalität - neben einer gesteigerten Produktivität - als wesentliche Verheißung von Wiederverwendung betrachtet (vgl. beispielhaft Endres 1988, Biggerstaff/Richter 1987, Tracz 1988 b). Lenz/Schmidt/Wolf (1987, S. 34) fordern in diesem Sinne "zero defect quality". Auch in der bereits erwähnten ISO/IEC-Norm 9126 werden wiederverwendbare Komponenten explizit als Beitrag zur Verbesserung von Softwarequalität eingeführt. In der damit verbundenen Vision werden Software-Komponenten - ähnlich wie bei ICs üblich - mit großer Sorgfalt von Entwicklern mit herausragender fachlicher Kompetenz erstellt und aufwendig getestet. Sie genügen zudem verbreiteten Normen über die Gestaltung von Schnittstellen. Anwendungsentwicklung ist danach wesentlich auf das Zusammenfügen solcher Bausteine gerichtet. Dem Verwender präsentieren sie sich in Form großer, wohldokumentierter Verzeichnisse.

Auch wenn es durchaus sinnvoll sein kann, innerhalb eines Unternehmens die Wiederverwendung von Software zu fördern, so liegt die besondere Attraktivität dieses Ansatzes doch in der Verbreitung der Komponenten in größerem Maßstab.[2] Für die Distribution jedoch trägt die Analogie zur Hardware nicht mehr: Einerseits ist die physische Distribution sehr viel leichter zu realisieren, andererseits wirft die Kopierbarkeit von Software eine Reihe ernster Fragen auf. Es liegt auf der Hand, daß sich ein entsprechender Markt nur dann etablieren kann, wenn die Interessen der Anbieter hinreichend geschützt werden. In der Literatur findet

1. Dazu Endres (1988, S. 94): "Der besondere Reiz steckt in folgendem Aspekt: Genau wie beim Einsatz von Werkzeugen verschiebt sich der Schwerpunkt von einer traditionell arbeitsintensiven zu einer eher kapitalintensiven Tätigkeit, ein Prozeß, der auch anderswo den Übergang von der handwerlichen zur industriellen Tätigkeit kennzeichnet.
2. In diesem Sinne propagieren Gibbs et al. (1990, ACM) "software information systems" als "publicly available resources rather than confined within single organizations." Dittrich (1990, S. 235) hält es für "denkbar, daß ... ein neuer Markt entsteht, auf dem fertige Typbibliotheken für bestimmte Anwendungsgebiete 'von der Stange' angeboten werden.

sich eine Reihe von Vorschlägen dafür, wie diese Herausforderung angenommen werden könnte. Beispielhaft dafür ist der Ansatz, den Cox (1992) skizziert. Er ist unter anderem inspiriert durch das Abrechnungsverfahren, das Nelson (1987) für ein öffentliches Hypertext-Publikationssystem skizziert. Cox schlägt vor, daß Komponenten, die in komfortabler Weise über geeignete Netze verfügbar gemacht werden, nicht - wie Hardware - durch Kauf in das Eigentum des Verwenders übergehen (um danach gegebenenfalls unrechtmäßig vervielfältigt zu werden). Stattdessen soll die tatsächliche Nutzung einer Komponente vergütet werden. Dazu sollten dann verschiedene Tarife bereitgestellt werden. Um ein solches Verfahren technisch zu realisieren, verweist Cox auf die Etablierung eines öffentlichen Netzes - ähnlich dem Strom- oder Telefonnetz. Die Komponenten müßten dann so gestaltet sein, daß sie nur genutzt werden können, wenn der jeweilige Rechner an dieses Netz angeschlossen ist. Durch geeignete Protokolle könnte dann die Nutzung einer Komponente erfaßt werden. In Anlehnung an eine Skizze von Mori (1990) nennt Cox dieses Verfahren "superdistribution".[1] Ein ähnlicher Vorschlag findet sich in Muralidharan/Weide (1988).

Die an der Komponenten-Metapher orientierten Vorstellungen von Wiederverwendung versprechen einen erheblichen Gewinn an Produktivität und Qualität. Der Entwurf eines konkreten Anwendungssystems aus Komponenten wird allerdings allenfalls indirekt (da Entwerfen auch immer induktiv erfolgt) unterstützt. Seit einigen Jahren werden Visionen diskutiert, die darauf zielen, den Systemgestalter weitgehend von dieser Anforderung zu entlasten. In deren Mittelpunkt stehen komplexe Komponenten oder Pakete, die speziell für bestimmte Anwendungsbereiche erstellt wurden. Da nicht davon ausgegangen wird, daß die vorgefertigen Systeme allen spezifischen Anforderungen genügen, wird deren komfortable Anpassung bzw. Konfiguration als Lösung vorgeschlagen. Wiederhold/ Wegner/Ceri (1990) entwerfen in diesem Sinne eine Vision von sogenannten "Megaprograms", die die Anforderungen eines großen Realitätsbereichs - als Beispiele wird ein US-weites Programm zur Steuerung der Warendistribution ("General Logistics") genannt - abdecken und aus ungefähr sieben (in Anlehnung an menschliche Wahrnehmungskapazität) "Megamoduls" zusammengesetzt sind. Ein Megamodul ist selbst ein Programm, das große reale Domänen abdeckt - als Beispiele werden "Luftfracht", "Lagerhaltung" und ähnliche genannt. Ein Megamodul ist nach außen verkapselt. Die Anpassung der Megamodule an spezifische Anforderungen und ihr Zusammenfügen ("glueing") zu einem Megaprogramm soll in komfortabler Weise durch eine Script-Sprache erfolgen. In ähnlicher Weise skizzieren Nierstrasz et al. (1990), wie "collections of (application) objects" mit Hilfe von "visual scripting" an die Bedürfnisse einer Anwenderorga-

1. Es liegt auf der Hand, daß dieser Vorschlag eine Reihe gewichtiger Probleme mit sich bringt, die allerdings an dieser Stelle nicht thematisiert werden sollen.

nisation angepaßt werden könnten (vgl. dazu II.2.2.1.2).

Auch wenn der eine oder andere Aspekt der skizzierten Visionen von Wiederverwendbarkeit Zweifel an der Machbarkeit wecken mag: Es bleibt ein hohes Maß an Attraktivität und es verwundert, daß dem Thema Wiederverwendbarkeit in der Informatik und vor allem in der Wirtschaftsinformatik kein größeres Forum gewidmet wird als es zur Zeit der Fall ist. Ein Grund dafür dürfte sein, daß das Bemühen um Wiederverwendbarkeit implizit ein wesentliches Merkmal der Informatik-Forschung ist: Das Augenmerk ist darauf gerichtet, von spezifischen Besonderheiten zu abstrahieren, generelle Konzepte zu entwickeln. Dies gilt natürlich auch für andere Disziplinen, ist gleichsam ein konstituierendes Merkmal wissenschaftlicher Forschung.[1] Wegner (1989, S.89) glaubt gar eine grundlegende Bedingung menschlichen Daseins zu erkennen:

> "The drive to create reusable rather than transitory artifacts has aesthetic and intellectual as well as economic motivations and is part of man´s desire for immortality. It distinguishes man from other creatures and civilized from primitive societies."

Auch wenn Wegner seine pathetischen Ausführungen als Appell verstanden wissen will, Wiederverwendbarkeit eine größere Beachtung zu schenken, liefert er gleichzeitig ein Indiz dafür, warum solche Analogien nicht immer auf positive Resonanz stoßen: Wenn das Bemühen um wiederverwendbare Konstrukte impliziter Bestandteil intellektueller Tätigkeit ist, heißt es Eulen nach Athen zu tragen, wenn man explizit danach fordert. Tatsächlich gibt es eine Vielzahl von Hilfsmitteln, die den Entwickler unterstützen, indem sie ihm in der einen oder anderen Form wiederverwendbare Konstrukte bieten. Hier ist vor allem an die breite Palette von Software-Entwicklungswerkzeugen zu denken.

Solche kritischen Einwände sind gewiß wichtig, da sie den Stellenwert von Wiederverwendbarkeit relativieren und damit einer letztlich schädlichen euphorischen Überhöhung vorbeugen. Sie diskreditieren allerdings nicht die Bedeutung einer Forschung, die explizit am Ziel Wiederverwendbarkeit orientiert ist. Wie sich in den folgenden Abschnitten noch zeigen wird, sind mit leistungsfähiger Wiederverwendbarkeit Anforderungen verbunden, die nicht allein durch den Hinweis darauf erfüllt werden, daß es doch bereits wiederverwenbare Konstrukte gibt.

1. Dies gilt ebenso für das Wiederverwenden selbst: Nicht nur in der wissenschaftlichen Forschung ist es unerläßlich, auf Vorarbeiten aus der Vergangenheit aufzubauen. Curtis (1989) betont diesen Umstand als wesentliches Merkmal der Professionalisierung: "The hallmark of the professionals is their ability to reuse knowledge and experience to perform their tasks ever more efficiently."

2.2 Dedizierte Forschungsansätze

Auch wenn in der Diskussion um Wiederverwendung häufig Metaphern wie Software-ICs oder Software-Bausteine im Vordergrund stehen, sind neuere Arbeiten zu diesem Thema durch einen breiteren Ansatz gekennzeichnet: Der Software-Entwickler soll nicht nur auf vorgefertigten Code zurückgreifen können, sondern auch auf andere nützliche Vorarbeiten. In diesem Zusammenhang spricht Freeman (1983) von "information assets", die er in fünf Klassen einteilt:

- code fragments
- logical structures ("internal design")
- functional architectures ("sets of application functions")
- external knowledge ("utilization knowledge, technology-transfer knowledge")
- environmental level information ("application area knowledge, development knowledge")

Die kurze Charakterisierung der Klassen macht deutlich, daß sie nicht disjunkt sind. Bemerkenswert ist zudem, daß explizit die Bedeutung von domänenspezifischem Wissen hervorgehoben wird. Jones (1984) deckt eine ähnliche Bandbreite ab, wenngleich er nur vier Kategorien verwendet: "code", "abstract components", "domain knowledge", "development knowledge". Biggerstaff und Perlis (1989, I, Einleitung) schlagen eine noch weitere Begriffsfassung vor: "Software reuse is the reapplication of a variety of kinds of knowledge about one system similar to another system in order to reduce the effort of development and maintenance of that system."

Eine solche Sicht scheint durchaus angemessen: Einerseits reflektiert sie die gängige Bedeutung von Wiederverwendung, andererseits betont sie zu Recht, daß dem Entwickler gewiß nicht nur durch vorgefertigten Code die Arbeit erleichtert wird. Damit wird aber zugleich deutlich, daß die Betrachtung des Phänomens Wiederverwendbarkeit in unserer Untersuchung die Konzentration auf einige wichtige Aspekte nötig macht, da sonst nicht nur die ganze Bandbreite technischer Hilfsmittel (Werkzeuge im weitesten Sinn), sondern auch die Vielfalt von Methoden und Prinzipien des Software-Entwurfs zu berücksichtigen wären. Die dedizierten Ansätze sind denn auch entweder inhaltlich auf einzelne Formen der Wiederverwendung eingeschränkt oder aber durch das Bemühen um konzeptuelle Grundlagen gekennzeichnet. Endres (1988, S. 88 ff.) unterscheidet vier Formen der "geplanten Wiederverwendung":

- Programm-Portierung
- Programm-Adaptierung
- Schablonen-Technik
- Baustein-Technik

Howden (1987) sieht im wesentlichen zwei Ansätze: top-down und bottom-up. Biggerstaff und Richter (1987) empfehlen eine weitere Differenzierung. Sie unterscheiden zwei Hauptformen: Wiederverwendung durch *Komposition* vorgefertigter Komponenten und durch *Generierung* von Code aus Vorgaben des Entwicklers. In ähnlicher Weise werden wir im folgenden zwischen Ansätzen, die auf Komponenten zielen (die also eine bottom-up Entwicklung vorsehen) und solchen, die eher eine deduktive (top down) Vorgehensweise empehlen, unterscheiden. Auch wenn beide Ansätze zunächst getrennt behandelt werden, schließen sie sich gewiß nicht gegenseitig aus: Die Verwendung von Komponenten kann durch die Bereitstellung des Entwurfs einer Anwendung erheblich unterstützt werden - et vice versa. In einer abschließenden Betrachtung genereller Voraussetzungen erfolgreicher Wiederverwendbarkeit werden sie deshalb gemeinsam behandelt.

2.2.1 Bottom Up: Ansätze zur Gestaltung wiederverwendbarer Komponenten

Die Bereitstellung von Komponenten ist gleichsam die klassische Form der Wiederverwendung. Hier ist vor allem an Funktionsbibliotheken höherer Programmiersprachen zu denken. Aber so hilfreich solche Bibliotheken auch sind, der Vision einer komfortablen, die Produktivität dramatisch erhöhenden Unterstützung werden sie nicht gerecht. Zu viel bleibt noch für den Anwendungsentwickler zu tun. Forschungsansätze, die auf der Komponentenmetapher aufbauen, sind denn auch wesentlich ambitionierter. Die zentralen Herausforderungen solcher Ansätze liegen auf der Hand:

- Der Entwickler muß wirksam darin unterstützt werden, nicht nur prinzipiell geeignete Komponenten zu finden, sondern diejenigen, die für den jeweiligen Verwendungszweck am besten geeignet sind.
- Dazu muß ihm - auf einer angemessenen Abstraktionsebene - verdeutlicht werden, was die Komponten leisten. Anders formuliert: Er muß ihre Bedeutung im Rahmen seiner Entwurfstätigkeit verstanden haben.
- Die Komposition der Komponenten sollte für den Entwickler einerseits in komfortabler Weise möglich sein, andererseits sollte sie softwaretechnischen Anforderungen an Sicherheit und Flexibilität genügen.
- Da nicht davon ausgegangen werden kann, daß alle bereitgestellten Komponenten exakt den Anforderungen einer spezifischen Implementierung genügen, ist eine komfortable Modifikation der Komponenten zu ermöglichen.

Wiederverwendbare Bausteine lassen sich differenzieren in Quellcode-Skelette ("code sceletons"), Funktionen (einschließlich Prozeduren), abstrakte Datentypen und Klassen (als Objektdeklarationen).

Während die Wiederverwendung von Quellcode zur Routine der Software-Entwicklung gehört, kann die Erstellung geeigneter Quellcode-Skelette zum Zweck der Wiederverwendbarkeit mit erheblichen Anforderungen verbunden sein. Dies liegt zum einen an folgendem Konflikt: Einerseits ist es erstrebenswert, die Lesbarkeit des Codes zu fördern, andererseits sollte die Bandbreite der Wiederverwendbarkeit nicht unnötig eingeschränkt werden. Die Verwendung domänenspezifischer Bezeichner wird gemeinhin als wichtiges Merkmal anschaulichen Codes betrachtet. Sobald aber der bereitgestellte Code in mehreren Domänen verwendet werden könnte, ist diese Anforderung für alle Domänen nicht mehr zu erfüllen. Darüber hinaus sind die Quellcode-Skelette so zu entwerfen, daß die vorspezifizierten Teile in den intendierten Anwendungsbereichen möglichst invariant sind, während die für Spezialisierung vorgesehenen Teile die Varianz der Anforderungen abdecken sollte.

Ende der siebziger Jahre wurden in einem kommerziellen Projekt 5000 Anwendungsprogramme auf Gemeinsamkeiten bzw. Ähnlichkeiten untersucht. Auf diese Weise wurden häufig benötige Code-Fragmente ermittelt und mit Hilfe von Schablonen, die in COBOL codiert wurden, zum Zweck der Wiederverwendung bereitgestellt (Lanergan/Grasso 1983). Endres (1988, S. 89) zeigt das Beispiel einer Schablone für die Codierung von Datenbank-Transaktionen. Der Verwender muß anwendungsspezifische Details und Bezeichner (Variablen- und Feldnamen) selbst hinzufügen. Ein solcher Ansatz ist relativ flexibel, da die angebotenen Bauteile beliebig verändert werden können. Darin liegt allerdings auch gleichzeitig seine wesentliche Schwäche: Das mit Software-Komponenten eng verknüpfte Leitbild von Sicherheit und Qualität ist in diesem Fall weitgehend aufzugeben. Innerhalb der dedizierten Ansätze zum Entwurf wiederverwendbarer Software-Bauteile spielen Quellcode-Skelette denn auch eine untergeordnete Rolle. Bezeichnend die Beurteilung von Ledbetter und Cox (1985, S. 424): "Hardware components are deliverable in an unmodifiable form. That means that standard functions are protected."

Die Entwicklung von Komponenten-Bibliotheken ist bisher weitgehend auf anwendungsunabhängige Konstrukte oder auf formalwissenschaftlich/technische Bereiche beschränkt. Mathematische Anwendungen im Rahmen statistischer Erhebungen oder naturwissenschaftlicher Analyseverfahren bieten geradezu ideale Voraussetzungen für die Erstellung wiederverwendbarer Komponenten: Es gibt eine begrenzte Zahl von Verfahren, die einerseits mathematisch hinreichend untersucht sind, um qualitativ hochwertige Implementierungen zu ermöglichen, deren Programmierung andererseits mit erheblichen Herausforderungen verbunden ist. Domänenübergreifende Bibliotheken zielen darauf, den Entwickler durch die Bereitstellung komplexer häufig benötigter Komponenten zu entlasten und die Softwarequalität zu fördern. Die folgende Übersicht (nach Endres 1988, S. 92

f.) zeigt, welche Aspekte bei entsprechenden Ansätzen berücksichtigt wurden. Es handelt sich dabei zum einen um Funktionen, zum anderen um komplexe Datenstrukturen und auf ihnen definierte Verfahren, die in Form abstrakter Datentypen implementiert sind.

Komponenten	*Verfasser*		
	Wolf/Schmidt 1985	Booch 1987	McNicholl 1987
Sortieren	•	•	•
Suchen		•	
Umwandeln	•	•	•
Vergleichen		•	
Kalender		•	•
Liste	•	•	
Tabelle	•		
Menge	•	•	
Keller	•	•	•
Schlange	•	•	•
Doppelschlange		•	
Ringe		•	•
Zeichenkette		•	
Matrix			•
Baum	•	•	
Graph		•	

Abb. 2: Ausgewählte Komponenten von Software-Bibliotheken (nach Endres 1988)

Die von Booch (1987) vorgeschlagene Bibliothek ist mit mehr als 500 Komponenten besonders umfangreich. Die Zielgruppe solcher Komponenten bilden vor allem die Entwickler von Systemsoftware und komplexen Anwendungen. Um den Verwender darin zu unterstützen, die angebotenen Komponenten zu verstehen, empfehlen Lenz/Schmidt/Wolf (1987) sogenannte "Building Blocks", die in drei Repräsentationformen beschrieben werden: einer Spezfikation, einer Schnittstellendefinition und mit Hilfe einer grafischen Entwurfstechnik. Die Entwicklung umfangreicher Komponentenbibliotheken wurde häufig mit der Programmiersprache ADA (so auch bei den in Abbildung 2 dargestellten Bibliotheken) durchgeführt. ADA erlaubt die Verwendung abstrakter Datentypen, die besonders geeignet sind, als wiederverwendbare Komponenten zu dienen. Ein abstrakter Datentyp ist gekennzeichnet durch eine Datenstruktur und eine Menge darauf operierender Funktionen. Die in einer Instanz verwalteten Daten sind nur

über die externe Schnittstelle - also durch die verfügbaren Funktionsaufrufe - zugänglich. Diese Verkapselung erlaubt eine hohe Sicherheit (Vermeidung von Seiteneffekten, Verhinderung unzulässiger Manipulationen) und erleichtert das Einfügen neuer Komponenten: Für die Verbindung mit anderen Komponenten ist allein die klar definierte Schnittstelle von Bedeutung.[1]

Zusammenfassend läßt sich feststellen, daß die genannten Ansätze auf systemnahe Komponenten zielen. Der Auswahl und angemessene Verwendung setzt einen sachkundigen Programmierer voraus. Dabei ist zweierlei zu berücksichtigen: Es gibt eine große Wahrscheinlichkeit dafür, daß derartige Komponenten bei der Implementierung vieler Programme benötigt werden, gleichzeitig stellen sie nur einen bescheidenen Beitrag für die Entwicklung großer Anwendungssysteme dar, da ihnen der Anwendungsbezug völlig fehlt - ein Umstand, auf den noch einzugehen sein wird.

2.2.1.1 Die Unterstützung der Suche nach Komponenten

Die Mehrheit der einschlägigen Forschungsarbeiten ist darauf gerichtet, den Verwender darin zu unterstützen, geeignete Komponenten zu finden.[2] Dabei geht es letztlich darum, möglichst viele der denkbaren Assoziationen der Verwender bei der Suche zu berücksichtigen. Eine Anforderung also, die aus dem Bereich des Information Retrieval bekannt ist. Die Anwendung von Verfahren zur Volltext-Suche ist deshalb naheliegend. Frakes/Nejmeh (1988) skizzieren einen entsprechenden Ansatz. Er basiert darauf, daß Quellcode oder ausgezeichnete Teile (wie Kommentare) desselben invertiert werden und damit eine leistungsfähige Suche nach einzelnen Zeichenketten (die auch durch boolesche Operatoren verknüpft sein können) möglich wird. Ein solcher Ansatz ist relativ leicht zu implementieren. Seine Bedeutung ist allerdings dadurch eingeschränkt, daß einzelne Wörter des Quellcodes selten repräsentativ für die Bedeutung des gesamten Fragments sind. Außerdem ist dieses Verfahren darauf angewiesen, daß die wiederverwendbaren Komponenten im Quellcode vorliegen.

Pintado (1990) verwendet den Quellcode nicht unmittelbar. Vielmehr schlägt er ein Verfahren vor, das an bestimmten formalen Merkmalen von Klassen (die allerdings unter Umständen nur durch Auswertung des Quellcodes zu ermitteln sind) orientiert ist. Er geht dabei davon aus, daß die Bedeutung von Bezeichnern sowie ihre Beziehungen untereinander kontextabhängig sind. Deshalb schlägt er

1. Abstrakte Datentypen sind wichtige Vorläufer von Klassen. Der wesentliche Unterschied zwischen beiden Konzepten liegt darin, daß abstrakte Datentypen keine Vererbung vorsehen.

2. so Braun/Schmidt (1988), Fischer (1987), Fischer/Henninger/Redmiles (1991), Frakes (1986/87), Frakes/Nejmeh (1988), Gaube/Lockemann/Mayr (1986), Gibbs (1990 b), Onuegbe (1988), Pintado (1990), Prietro-Diaz (1991), Prietro-Diaz (1987)

einen Ansatz vor, der es erlaubt für jeden relevanten Kontext einen sogenannten "view" zu definieren. Ein View besteht aus Klassenbezeichnern und den zwischen ihnen bestehenden Beziehungen. Zur Beschreibung der semantischen Beziehungen zwischen den Bezeichnern können für jeden view Ähnlichkeitsmaße definiert werden, die an Eigenschaften der jeweils zugehörigen Klasse geknüpft sind. Als Beispiele dafür nennt Pintado Maße für die strukturelle Übereinstimmung zwischen Implementierungen und die relative Zahl gegenseitiger Referenzierungen.

Andere Ansätze zielen darauf, die angebotenen Komponenten vorab für eine leistungsfähige Suche zu präparieren. Dazu werden sie mit Annotationen versehen - was zu einem von der Beschlagwortung von Literatur bekannten Problem führt: Die Leistungsfähigkeit von Schlagworten steht und fällt mit ihrer Angemessenheit, deren Beurteilung unterliegt allerdings subjektiven Schwankungen - nicht zuletzt bei denen, die Schlagworte zuordnen. Ein im Information Retrieval üblicher Weg, diesem Problem zu begegnen, ist die Erstellung von Schlagwortkatalogen oder - anspruchsvoller - von Thesauri. Ähnliche Ansätze finden sich in der Literatur über das Retrieval von Software-Komponenten. Dabei wird in der Regel die Klassifikation der Komponenten - gleichsam eine strukturierte Beschlagwortung - vorgeschlagen. Onuegbe (1988) entwirft ein System von Kategorien, die hierarchisch geordnet sind. Jede Kategorie repräsentiert eine abstrakte Spezifikation, zu der mehrere Implementierungen gehören können. Wenn neue Komponenten eingeordnet werden sollen, für die keine abstrakte Beschreibung existiert, ist eine geeignete Kategorie zu definieren. Langfristig sollen auf diese Weise vollständige Kategoriensysteme für einzelne Anwendungsbereiche entstehen: "The emphasis is on building domain-specific libraries ..." (Onuegbe 1988, S. 164). Es ist offensichtlich, daß diese Form der Klassifikation durch Klassenhierarchien (im Sinne objektorientierter Sprachen) angeregt ist. Onuegbe nennt denn auch explizit Smalltalk als wichtige Orientierung.

functions	*objects*	*medium*	*system-type*	*functional-area*	*setting*
add	arguments	array	assembler	accounts-payable	advertising
compare	blanks	disk	compiler	auditing	association
create	descriptors	line	file-handler	bookkeeping	TV-station
delete	directories	mouse	hybrid-DB	capacity-planning	car-dealer
•	•	•	•	•	•

Abb. 3 "Faceted Classification" von Funktionen (nach Prietro-Diaz 1987, S. 11)

Der Nachteil eines solchen Ansatzes liegt darin, daß er letztlich in einer Vielzahl von Kategorien mündet, denen eine Komponente jeweils eindeutig zuzuordnen ist. Eine Suche, die an anderen Assoziationen orientiert ist, hat also kaum Aussicht auf Erfolg. Prieto-Diaz (1987, 1991) schlägt ein Verfahren vor, das es einerseits erlaubt, mehrere Aspekte, die bei der Suche im Vordergrund stehen mögen, zu berücksichtigen, und andererseits die Beschlagwortung durch ein vorgegebenes Schema anleitet. Er knüpft dazu an einen Ansatz an, der im Bibliothekswesen seit langem bekannt ist (Ranganathan 1967): Die "faceted classification" sieht vor, einen Gegenstand mit Hilfe der Ausprägungen einer vorgegebenen Liste von Aspekten (Facetten) zu beschlagworten. Die von Pietro-Diaz eingeführte Liste enthält sechs Aspekte, die in Abbildung 3 veranschaulicht sind.

Dabei kann jedem Aspekt nur genau ein Schlagwort zugeordnet werden. Um die Auswahl von Schlagworten für einzelne Facetten anzuleiten, wird ein Thesaurus verwendet. Die im Thesaurus definierten Beziehungen zwischen den zur Verfügung stehenden Begriffen bilden zudem die Grundlage dafür, beim Retrieval Ähnlichkeiten berücksichtigen zu können. Der Wert einer "faceted classification" hängt also wesentlich von der Qualität des je verwendeten Thesaurus ab. Prieto-Diaz läßt allerdings offen, wie er sich die Entwicklung eines solchen Thesaurus vorstellt. Grundsätzlich liegt es nahe, einen Thesaurus für eine bestimmte Anwendungsdomäne zu entwickeln, um dann anschließend einzelne Komponenten unter Rückgriff darauf zu beschlagworten. Die Beispiele lassen allerdings den Eindruck eines bottom up-Ansatzes entstehen: Komponenten werden in einem bestimmten Kontext ("setting", "functional area") entwickelt, anschließend werden dafür Schlagworte eingeführt, die gegebenenfalls auch noch in den Thesaurus eingefügt werden müssen.

Ein wesentliches Problem einer auf bestimmte Kategorien und einen einheitlichen Thesaurus beruhenden Beschlagwortung ist darin zu sehen, daß die Assoziationen der Suchenden interpersonell und kontextabhängig streuen. Dieser Umstand wird dadurch verstärkt, daß die Beschlagwortung in der Regel mit einer anderen Motivation durchgeführt wird als die Suche nach einer Komponente: Auf der einen Seite steht das Bemühen, eine Menge von Teilen möglichst konsistent und einheitlich zu bezeichnen, auf der anderen Seite sieht sich der Entwickler einem Auftrag gegenüber, der unter Rückgriff auf domänenspezifische Konzepte formuliert ist und sucht dafür geeignete Bauteile. Frakes/Henninger/Redmiles (1991, S. 326) sprechen hier von einem "mismatch between the situation and system model."

2.2.1.2 Anpassung und Zusammenfügen von Komponenten

Wenn vorgegebene Komponenten spezifischen Anforderungen nur unzureichend gerecht werden, hängt ihre Brauchbarkeit davon ab, ob und mit welchem Auf-

wand sie sich anpassen lassen. Grundsätzlich lassen sich drei Formen der Anpassung unterscheiden:

- Parametrisierung
- Modifikation der Komponenten
- Hinzufügen neuer Komponenten unter Nutzung vorgegebener Komponenten

Dabei könnte man durchaus die Ansicht vertreten, daß Parametrisierung keine Anpassung darstellt, schließlich gehört die Verwendung von Parametern zu den vorbestimmten Nutzungsformen der Komponenten. Jenseits solcher terminologischer Erwägungen scheint eine gemeinsame Betrachtung allerdings durchaus sinnvoll, da Parametrisierung mit den beiden anderen Formen zur Anpassung (oder Konfiguration) konkurriert (wobei sie sich gegenseitig nicht ausschließen). Die Verwendung von Parametern liegt immer dann nahe, wenn eine Funktion in unterschiedlichen Ausprägungen ausgeführt werden kann.[1] Bekannte Beispiele dafür sind die Funktionen von Betriebssystemen oder auch Funktionen zur Gestaltung grafischer Benutzerschnittstellen. Wer jemals die Funktionen eines Betriebssystems wie Unix benutzt hat, weiß um die Schattenseiten der Parametrisierung: Die häufig kryptische Verwendung von Parametern, deren Bedeutung unter Umständen sogar interdependent ist, macht es selbst dem geübten Benutzer nahezu unmöglich, auf ein Handbuch zu verzichten.

Andererseits kann nicht übersehen werden, daß die Verwendung von Parametern die Zahl der Funktionen (bzw. der Funktionsbezeichner) reduziert und damit das Finden einer Funktion erleichtern kann. Als wesentlicher Nachteil bleibt der mangelhafte Komfort. Diesem Nachteil kann dadurch entgegengewirkt werden, daß die Einzelheiten der Parametrisierung dem Entwickler oder Benutzer verborgen bleiben. Dazu kann mit Hilfe geeigneter Fragen, die sich unmittelbar auf den jeweiligen Anwendungsbereich beziehen, die jeweilige Parameterspezifikation indirekt ermittelt werden. Aus softwaretechnischer Sicht ist Parametrisierung wesentlich mit der Herausforderung verbunden, Systeme zu entwerfen, die es gestatten, einen möglichst großen Teil einer Spezifikation variabel zu formulieren (also beispielsweise Datentypen, Schnittstellen und dergleichen). Innerhalb der dadurch ermöglichten Bandbreite kann dann durch die Festlegung von Parametern die je gewünschte Spezialisierung festgelegt werden (vgl. Goguen 1989). Aus konzeptueller Sicht setzt Parametrierung die Analyse von Systemvarianz voraus: Welche Teile der Beschreibung eines Anwendungssystems können sich ändern, in welchem Rahmen spielen sich solche Änderungen ab und wie können sie formal beschrieben werden.

Um die Modifikation von Komponenten zu erleichtern, schlagen Katz et al.

1. Ein ausführlicher Bezugsrahmen zur Gestaltung und Verwendung der Parametrisierung in objektorientierten Systemen findet sich in Goguen (1989).

(1987) vor, eine Komponente als ein "partially interpreted schema" zu präsentieren. Ein solches Schema enthält variable Teile (Datentypen, Funktionsdeklarationen), die instanziert werden können. Um fehlerhafte Modifikationen möglichst zu verhindern, kann die Ablauflogik ("schema body") nicht verändert werden. Demgegenüber bietet eine uneingeschränkte Modifikation von Komponenten sicherlich die größte Flexibilität, birgt aber auch - wie oben bereits dargestellt - große Probleme: Sie kann mit einem erheblichen Aufwand verbunden sein und beinhaltet das Risiko, daß die Qualität der Komponenten erheblich gemindert wird.

Nach dem Leitbild industrieller Softwareerstellung mit Hilfe hochwertiger Bausteine erfolgt das Zusammenfügen vorgegebener Bausteine über wohldefinierte Schnittstellen. Dabei sollten Seiteneffekte vermieden und unzulässige Verbindungen möglichst verhindert werden. Schließlich sollte das "Zusammenstecken" der Komponenten in anschaulicher Weise möglich sein. Es gibt eine Reihe softwaretechnischer Prinzipien, die darauf zielen, solchen Herausforderungen zu begegnen. Dabei ist an das Prinzip größtmöglicher Lokalität zu denken, an Typisierung und an die anschauliche Darstellung von Schnittstellen. Aber selbst wenn das Zusammenfügen von Komponenten softwaretechnisch gut unterstützt wird (wie etwa in manchen objektorientierten Sprachen), bleibt für den Entwickler noch eine erhebliche Herausforderung: Nicht nur, daß er die für seinen Bedarf geeigneten Komponenten finden muß, er muß zudem das Verhalten des Systems definieren. Für den Fall, daß keine Anpassungen durchzuführen sind, ist dazu - vereinfacht dargestellt - ein Drehbuch zu erstellen, in dem festgelegt ist, welchem Objekt unter welchen Bedingungen welche Nachrichten zu schicken sind. Forschungsarbeiten, die darauf zielen, das Zusammenfügen von Komponenten zu einer lauffähigen Anwendung zu unterstützen, setzen denn auch in der Regel darauf, eine besonders komfortable Drehbuchbeschreibung ("scripting") zu ermöglichen.[1]

Während Wiederhold/Wegner/Ceri (1990) auf eine formalsprachliche Notation setzen (ohne dabei eine entsprechende Sprache zu beschreiben), schlagen Nierstrasz et al. (1990) eine komfortablere Lösung vor: Mit Hilfe einer sogenannten "Visual Scripting Language" soll der Entwickler das Verhalten der ausgewählten Komponenten festlegen. Ihm soll dazu ein dedizierter grafischer Editor geboten werden. Dabei gilt: "Visual Scripting requires (1) that objects to be scripted have a visual presentation, and (2) that objects have a "scripting interface" which permits their behaviour to be graphically edited." (Nierstrasz et al. 1990, S. 3). Als Beleg der Machbarkeit ("proof-of-concept prototype") wird ein Werkzeug vorgestellt, das es gestattet, ikonisch repräsentierte Komponenten miteinander zu einer

1. Eine flexibler, wenn auch wenig integritätsfördernder Ansatz dazu sind die "Filter" und "Pipes" zur Verknüpfung von Unix-Prozessen.

Anwendung - dargestellt als "visual script" zu verknüpfen. Eine Komponente ist entweder ein Unix-Kommando, eine Unix-Prozedur (shell script), eine Datei, eine Zeichenkette oder - um verschiedene Detaillierungsstufen zu ermöglichen - ein "visual script". Die Verknüpfung erfolgt über Eingangs- und Ausgangsports. Als Beispiel wird unter anderem ein Script präsentiert, das beschreibt, wie die Häufigkeit der Wörter einer Textdatei zu ermitteln und auszugeben ist.[1]

Für unsere Untersuchung bleibt festzuhalten: Die mit bottom up-Ansätzen verbunden Herausforderungen können nur bewältigt werden, wenn es für den je intendierten Einsatzbereich der Komponenten ein sorgfältig entworfenes Modell der relevanten Konzepte gibt, die zudem unter Rückgriff auf eine Terminologie bezeichnet sind, die den Verwendern vertraut ist. Anders formuliert: Reine bottom up-Ansätze können kaum erfolgreich sein.

2.2.2 Top down: Von der Anwendungsbeschreibung zur Implementierung

Komponenten entbinden den Entwickler nicht - oder nur zu einem geringen Teil - von der Aufgabe, das Gesamtsystem zu entwerfen. Diese Aufgabe ist nicht nur mit einem erheblichen Aufwand verbunden, sie determiniert auch wesentlich die Qualität der zu erstellenden Software. Die Bereitstellung genereller Entwürfe zielt darauf, eine entsprechende Unterstützung zu bieten. Unsere Betrachtung einschlägiger Arbeiten wird dabei unter anderem durch die folgenden Fragen geleitet:

- In welcher Form wird ein wiederverwendbarer Entwurf dem Entwickler präsentiert?
- Wie vollzieht sich die Anpassung von Entwürfen an spezielle Bedürfnisse?
- Wie wird die Transformation eines Entwurfs in ein lauffähiges Programm unterstützt?

In der Literatur werden im wesentlichen zwei Ansätze behandelt, die ein deduktives Vorgehen favorisieren: Die Bereitstellung von Entwürfen für bestimmte Anwendungsbereiche und die Entwicklung von Anwendungsgeneratoren, die eine möglichst anschauliche Anwendungsbeschreibung in ein lauffähiges Programm zu transformieren versprechen. Auch wenn es zwischen beiden Ansätzen erhebliche Überschneidungen gibt, werden wir sie (zunächst) getrennt betrachten.

1. Im Unterschied dazu setzen Kaiser/Garlan (1987) auf eine Sprache ("Meld"), die es erlaubt, das Verhalten einer Menge von Klassen ("feature") in deklarativer Form zu beschreiben. Dazu werden nicht die Nachrichten benannt, die jeweils ausgetauscht werden. Vielmehr werden mit Hilfe sogenannter "action equations" Bedingungen beschrieben, die nicht verletzt werden dürfen. Durch die Festlegung solcher Constraints wird also indirekt die zulässige Interaktion der ausgewählten Klassen beschrieben. Aber auch Kaiser/Garlan bleiben den Nachweis schuldig, daß durch "Meld" eine besonders komfortable Komposition von Klassen ermöglicht wird.

2.2.2.1 Wiederverwendbare Anwendungsentwürfe

Es gibt eine Fülle von Entwürfen, für die Wiederverwendung gleichsam konstitutiv ist. Neben unternehmensindividuellen Entwürfen (z.B. in Form von Datenmodellen) ist dabei vor allem an die Spezifikation von Funktionen, Datenstrukturen oder Systemarchitekturen zum Zwecke ihrer Standardisierung zu denken. Also beispielsweise Datenstrukturen für die Realisierung von Electronic Data Interchange (EDI) oder Compound Document Architectures. Im Unterschied dazu sind dedizierte Wiederverwendungsansätze weniger einschränkend: Sie sind nicht zuletzt darauf gerichtet, eine komfortable Modifikation vorgegebener Entwürfe zu erlauben. Die folgende Betrachtung zweier solcher Ansätze zielt nicht vorrangig auf eine detaillierte Kritik, vielmehr soll sie Hinweise darauf liefern, welche Anforderungen mit der Bereitstellung von Entwürfen verbunden sind und wie ihnen begegnet werden kann.

Kandt (1984) geht davon aus, daß die Wiederverwendbarkeit von Entwürfen wesentlich davon abhängt, in welcher Form ein Entwurf präsentiert wird und wie die Suche nach geeigneten Entwürfen unterstützt wird. Dazu beschreibt er einerseits die Einordnung eines Entwurfs in den jeweiligen Anwendungskontext mit Hilfe semantischer Netze, die die Suche nach geeignenten Komponenten unterstützen sollen. Zudem wird für jeden Entwurf eine Spezifikation abgelegt, die unmittelbar die Implementierung mit Hilfe einer geeigneten Programmiersprache anleiten soll. Es wird allerdings nicht deutlich, wie mit Hilfe eines solchen Ansatzes komplexe Anwendungsentwürfe anschaulich abgebildet werden können. Die Beispiele beschränken sich denn auch auf Entwürfe von Sortierverfahren, so daß der Eindruck entsteht, es wird - im Unterschied zum Anspruch - vor allem eine Unterstützung der Suche nach Komponentenspezifikationen geboten.

In dem am MCC in Austin durchgeführten ROSE-2-Projekt wurde ein umfangreiches softwaretechnisches Konzept zur Förderung wiederverwendbarer Entwürfe entwickelt (Lubars 1990). Dabei stand nicht die Betrachtung spezifischer Anwendungen im Vordergrund. Vielmehr sollten Entwürfe so generell sein, daß "a family of related design problems" (Lubars 1990, S. 2) abgedeckt wird. Lubars nennt die wiederverwendbaren Entwurfsdokumente "design schemas". Ein solches Schema ist eine generelle Beschreibung ("basic architecture") eines Systementwurfs. Die Suche nach einem geeigneten Entwurf kann durch eine Beschlagwortung unterstützt werden. Zur Darstellung der Entwürfe kann auf verschiedene Repräsentationsformen zurückgegriffen werden. Lubars nennt hier unter anderem Datenflußdiagramme, Struktogramme, State/Transition-Diagramme und Petrinetze. Im Unterschied zu Kandt wird dabei explizit auf die Abbildung komplexer Systeme geachtet. So führt Lubars als Beispiel ein Produktionsplanungssystem an. Um die damit verbundene Komplexität auf eine übersichtliche Darstellung zu reduzieren, werden dem Benutzer unterschiedliche Detaillierungsstufen geboten.

Der komfortablen und gleichzeitig möglichst konsistenten Anpassung an spezielle Anforderungen kommt in ROSE-2 besondere Bedeutung zu. Zur Unterstützung der Spezialisierung gehört zu jedem Schema eine Menge von Anforderungs- und Entwurfsalternativen, zwischen denen gewählt werden kann. Diese Alternativen sind möglichst in einer anwendungsnahen Form - also unter Verwendung domänenspezifischer Termini - abzubilden. Als Beispiel nennt Lubars unter anderem die Frage nach der gewünschten Verarbeitungskapazität eines Produktionsplanungssystems, zu deren Beantwortung der Benutzer aus einer Menge vorgegebener Alternativen wählen kann. Neben einer solchen Parametrisierung werden "specialisation rules" und "refinement rules" eingeführt. Diese Regeln beschreiben Zusammenhänge zwischen Entwurfsalternativen bzw. die Auswirkungen bestimmter Festlegungen. Sie werden mit Hilfe eines wissensbasierten Systems verwaltet. Sie dienen der Unterstützung einer konsistenten Spezialisierung. Wenn der Benutzer beispielsweise eine größere Produktionskapazität an einer bestimmten Stelle des Fertigungsprozesses festlegt, mag dadurch an einer anderen Stelle ein Engpaß entstehen. Dieser kann durch Rückgriff auf geeignete Regeln behoben werden oder aber wenigstens dem Benutzer mitgeteilt werden.

Um die rasche Implementierung eines spezialisierten Entwurfs zu unterstützen, sind die verfügbaren Entwurfsartifakte entsprechenden Modulen zugeordnet. Dieser Umstand soll dem Benutzer verborgen bleiben: "The reusable design can provide the lower-level design and implementation details, and pull in the appropriate inventory control algorithms, and the code to operate the factory machinery, from the associated libraries." (Lubars 1990, S. 2)

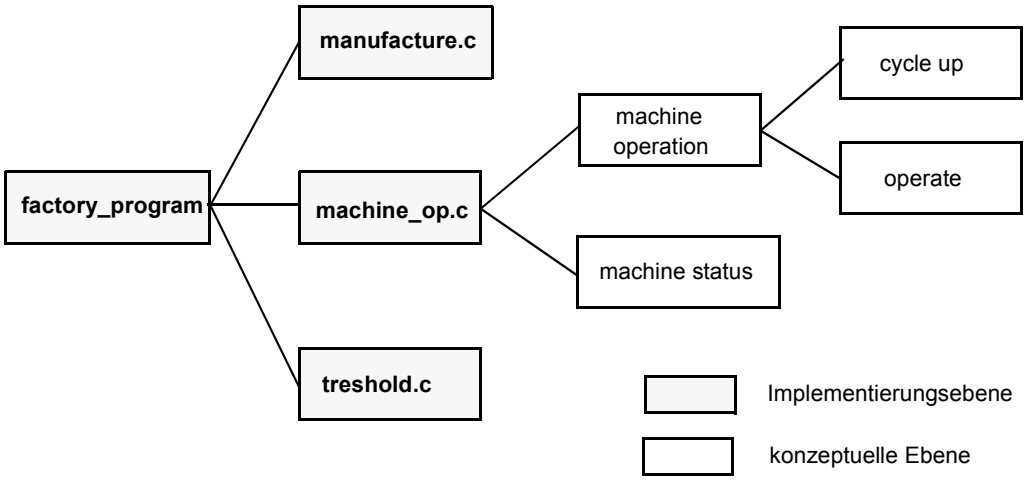


Abb. 4: Die Zuordnung von Elementen der konzeptuellen Ebene zu solchen der Implementierungsebene (nach Lubars 1990, S. 28)

Hier wird also eine Verbindung zu wiederverwendbaren Komponenten hergestellt. Der anschauliche, an den Gegebenheiten der jeweiligen Domäne orientierte Gesamtentwurf dient gleichsam als Front-End für die (transparente) Auswahl der geeigneten Komponenten. Ein solcher Ansatz impliziert, daß die auf der konzeptuellen Ebene durchgeführten Änderungen nur in einer Bandbreite möglich sind, die durch vorgegebene Implementierungen (genauer: durch im System vorhandenes Wissen zur Auswahl und Komposition vorhandener Bausteine) abgedeckt ist. Dem damit verbundenen Nachteil eingeschränkter Spezialisierungsmöglichkeiten stehen gewichtige Vorteile gegenüber: Die Modifikation ist komfortabel (Auswahl aus gegebenen Alternativen) und sicher (durch den Rückgriff auf vorhandene Integritätsregeln). Darüber hinaus wirken die bekannten Vorteile sorgfältig entwickelter Komponentenbibliotheken gleichsam in extenso: Die gesamte Anwendung ist allein aus vorgefertigten Komponenten zusammengefügt.

Der Erfolg eines solchen Ansatzes hängt damit wesentlich davon ab, wie gut die Theorien bzw. Modelle des jeweils abzudeckenden Realitätsbereichs sind. Es sind eben nicht nur invariante Merkmale zu erfassen. Vielmehr muß darüber hinaus auch die für spezielle Implementierungen bedeutende Varianz durch eine geeignete Vielfalt von Entwurfsartifakten und Komponenten abgedeckt werden. Dieser Anspruch wird in ROSE-2, das vor allem softwaretechnischen Anforderungen gewidmet ist, vernachlässigt. Lubars verweist hier vielmehr auf Forschungsarbeiten zur Analyse und Modellierung bestimmter Anwendungsdomänen, wie etwa Arango (1989).

DRACO (Neighbors 1989) ist ebenfalls eine umfangreiche Software-Entwicklungsumgebung, die den Einsatz von Entwürfen zur komfortablen Nutzung von Komponenten vorsieht. Neighbors spricht in diesem Zusammenhang von "domain models". Wir können hier von den vielfältigen (und m.E. nur unzureichend dokumentierten) Verfahren, die in DRACO eingesetzt werden, absehen. Wichtig ist allerdings die von Neighbors skizzierte Organisation von "domain models". Sie sind auf unterschiedlichen Abstraktionsebenen zu denken: "The domains with the highest level of abstraction are called application domains, while the domains in the abstraction levels between application domains and execution domains are called modeling domains." (Neigbors 1989, S. 307) Die Konstrukte, die der Beschreibung einer Domäne dienen, werden typischerweise unter Rückgriff auf Konstrukte einer in der Hierarchie darunter oder auf gleicher Ebene liegenden Domäne definiert. Ein Ansatz, der in der Software-Entwicklung seit langem eine Rolle spielt: So kann man zur Beschreibung einer Anwendung auf Konstrukte von Domänen wie "Grafische Benutzerschnittstelle", "Datenbankmanagementsystem" und dergleichen zurückgreifen. Im Unterschied dazu scheint Neighbors eine sorgfältig entworfene Hierachie von Domänen zu inten-

dieren, die darauf gerichtet ist, Redundanz zwischen den einzelnen Domänen zu vermeiden und gleichzeitig den Bedarf einer Domäne durch Konstrukte anderer Domänen möglichst vollständig zu decken.

2.2.2.2 Anwendungsgeneratoren

Im Unterschied zu Konzepten, die die Auswahl und das Zusammenfügen von Bausteinen beinhalten, sind Ansätze, die auf Code-Generierung zielen, schwierig abzugrenzen. Horowitz/Munson (1984, S. 34) dazu: "Generation technologies are not as easy to characterize as composition technologies, because the components that are reused are not concrete, self-contained entities." Diese Einschätzung von Horowitz und Munson ist m.E. nicht zutreffend. Ein Anwendungsgenerator kann durchaus auf konkrete Komponenten zurückgreifen.[1] Eine präzise Begriffsfestlegung wird vor allem dadurch erschwert, daß Generatoren erhebliche Ähnlichkeiten mit Compilern, Interpretern und dergleichen aufweisen: In allen Fällen wird von einer Repräsentationsform in eine andere Darstellung transformiert. Die Besonderheit von Generatoren liegt in der Repräsentation der jeweils verarbeiteten Eingabe: Sie soll sehr viel anschaulicher sein als es für die formalen Sprachen, die von Compilern oder Interpretern bearbeitet werden, üblicherweise der Fall ist. In diesem Zusammenhang wird mitunter von "Endbenutzerprogrammierung" oder "automatischem Programmieren" gesprochen: "An automatic programming system allows a computationally naive user to describe problems using the natural terms and concepts of a domain ..." (Barstow 1985, S. 1321). Dabei ist es letztlich weniger wichtig, daß die Benutzer von Generatoren unfähig sind, zu programmieren. Entscheidend ist, daß die durch einen Generator ermöglichte Repräsentationsform die Verwendung von Konstrukten erlaubt, die in anschaulicher Weise mit Elementen des jeweiligen Anwendungsbereichs bzw. der zu erstellenden Anwendung korrespondieren. Auf diese Weise wird eine komfortablere und produktivitätsförderdernde Beschreibung von Programmen gefördert. Damit ergibt sich die Schwierigkeit, von den oben skizzierten wiederverwendbaren Entwürfen abzugrenzen, die zum Teil auch eine wenigstens prototypische Implementierung ermöglichen. Die Beschreibung von ROSE-2 legt nahe, daß es sich dabei um die direkte Zuordnung von Elementen der anwendungsnahen Repräsentationsform zu implementierten Komponenten handelt. Im Unterschied dazu impliziert Generierung nicht eine solche eineindeutige Zuordnung. Eine überzeugende Abgrenzung ist allerdings kaum möglich.

Für unserer Betrachtung sind an dieser Stelle zwei Fragen von Bedeutung:

1. Hier ist beispielsweise an Systeme zur Generierung von Benutzerschnittstellen zu denken, die auf eine Menge vordefinierter Interaktionselemente zurückgreifen.

- Wie stellt sich der Zusammenhang zwischen Generatoren und Wiederverwendbarkeit dar?
- Welche Rolle spielen Entwürfe/Modelle von Anwendungsbereichen für die erfolgreiche Realisierung von Generatoren?

Eine anwendungsnahe Repräsentation dient als komfortables Front-End für implementierungsnähere Darstellungsformen - und die dort zu spezifizierenden Zugriffe auf wiederverwendbare Komponenten. Ein Beispiel dafür sind SQL-Generatoren. Der Benutzer kann durch die anschauliche Darstellung einer Bildschirmmaske eine Tabelle definieren und damit implizit bestimmte Zugriffsformen auf Elemente dieser Tabelle (also beispielsweise Zugriff über Spaltenwerte oder logische Verknüpfungen derselben). Aus der am Interaktionsmodell (also dem, was der Benutzer als Anwendung wahrnimmt) orientierten Repräsentation werden dann entsprechende SQL-Anweisungen generiert. Die Verbindung mit wiederverwendbaren Komponenten erfolgt erst danach: Durch SQL-Compiler und -Linker werden die jeweils benötigen Funktionen des Datenbankmanagementsystems angebunden. Dieses Beispiel macht auf einen grundsätzlichen Konflikt bei der Gestaltung der für einen Generator vorgesehenen Repräsentationsform aufmerksam: Einerseits soll sie anschaulich sein und eine komfortable Nutzung unterstützen, andererseits ist es wünschenswert, daß eine große Bandbreite der Anforderungen eines Benutzers berücksichtigt werden kann. Beides läßt sich nicht immer vereinbaren. So gestattet es etwa die am Query-by-example orientierte Beschreibung von Datenbankzugriffen nicht, die Mächtigkeit von SQL vollständig zu nutzen.

Ein Ansatz, diesen Konflikt abzuschwächen, besteht darin, dem Benutzer zu gestatten, die generierte Repräsentation - unter Nutzung der dort verfügbaren Ausdrucksmächtigkeit - zu nutzen. Die Nachteile eines solchen Vorgehens liegen auf der Hand: Nach der Modifikation des generierten Codes sind beide Repräsentationsformen nicht mehr äquivalent, die eigentlich für die Anwendungsbeschreibung vorgesehene Repräsentationsform kann nicht mehr weiterverwendet werden. Es ist deshalb wünschenswert, daß die Ausdrucksmächtigkeit der dem Generator zu übergebenden Repräsentationsform (man kann hier auch von der Modellierungsebene sprechen) genauso groß ist wie die der erzeugten Sprache. Für den Fall, daß die Forderung nach äquivalenten Ausdrucksmächtigkeiten nicht erfüllt werden kann, sind Mechanismen erstrebenswert, die die Konsistenz der Beziehungen zwischen den verschiedenen Ebenen kontrollieren.

Für unsere Betrachtung ist es wesentlich, folgenden Zusammenhang, auf den an anderer Stelle bereits hingewiesen wurde, festzuhalten: Um dem Benutzer eine Repräsentationsform zu bieten, die es gestattet, einen bestimmten Anwendungsbereich in komfortabler Weise zu beschreiben, ist ein geeignetes Modell dieser Domäne unerläßlich. So kann nur auf diese Weise eine Benennung der angebote-

nen Konstrukte erfolgen, die auf einer der Anwendungsebene entsprechenden Terminologie beruht. Vor allem aber ist ein Modell (oder eine Theorie) erforderlich, um die Breite der möglichen Anforderungen zu erfassen und deutlich zu machen, an welchen Stellen Anforderungen variieren können.

Wenn man den Bereich "grafische Benutzerschnittstellen" betrachtet, heißt das, es muß eine Auswahl konfigurierbarer Widgets geboten werden, die die Anforderungen in diesem Bereich möglich umfassend abdecken. Dazu muß eine geeignete Generalisierung über Benutzerschnittstellen der als wichtig erachteten Anwendung durchgeführt werden. Darüber hinaus ist zu ermitteln, in welcher Form der jeweilige Anwendungsbereich darzustellen ist, um für die intendierten Benutzer möglichst anschaulich zu sein. Dazu sind Konzeptualisierungen der Benutzer modellhaft zu rekonstruieren. Im Falle grafischer Benutzerschnittstellen ist dies aus doppeltem Grund relativ unproblematisch: So ist zum einen die Beschreibung einer Benutzerschnittstelle als Benutzerschnittstelle im Wortsinn anschaulich, zum andern handelt es sich hier um einen Bereich, der erst durch Informationstechnologie konstituiert wurde. Es müssen also keine vorhandenen Wahrnehmungsmuster nachgezeichnet werden, vielmehr geht es darum, die Funktionalität einer Technologie mit Hilfe verständlicher Konzepte zu präsentieren. Wenn man anwendungsnähere Domänen betrachtet, also beispielsweise "KFZ-Handel", wird es wesentlich schwieriger, eine geeignete Repräsentationsform zu finden.

Die konsequente Forderung nach Wiederverwendbarkeit legt es zudem nahe, über die Grenzen eines irgendwie abgegrenzten Anwendungsbereichs hinweg zu blicken: Wenn ein Bereichsmodell Gemeinsamkeiten mit einem anderen aufweist, kann dies ein Anlaß sein, durch geeignete Generalisierungen eine neue Domäne zu definieren - so wie Neighbors (1989) es mit dem Hinweis auf Domänenhierarchien skizziert.

2.2.3 Empirische Untersuchungen

Um auf einer anwendungsnahen Ebene brauchbare wiederverwendbare Komponenten oder Entwürfe zu gestalten, ist eine empirische Orientierung unerläßlich. Ob es sich dabei um die Unternehmen einer bestimmten Branche oder -weniger ambitioniert - um ein einziges Unternehmen handelt: In jedem Fall müssen die realen Anforderungen betrachtet werden. Nur so kann festgestellt werden, ob einzelne Konstrukte mehrfach verwendet werden könnten. Es gibt denn auch eine Reihe dedizierter empirischer Untersuchungen, die auf die Erfassung von Wiederverwendungspotentialen gerichtet sind. Sie lassen sich in ex post- und ex ante-Ansätze unterscheiden.

Ex post-Untersuchungen sind vor allem an der Frage orientiert, in welchem Umfang bestimmte Konstrukte in Software-Entwicklungsprozessen wiederver-

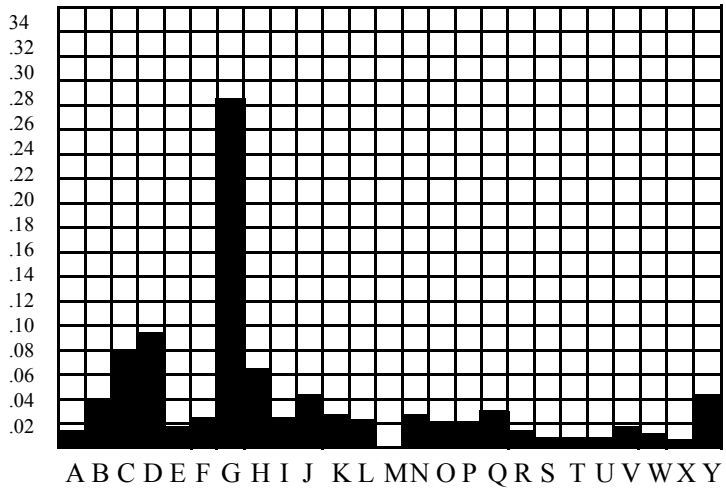
wendet wurden. Damit wären sie grundsätzlich geeignet, Hinweise für den Entwurf von Domänenmodellen zu liefern. Die vorliegenden Studien[1] zielen allerdings vorrangig auf eine Evaluation der Effizienz von Maßnahmen zur Förderung von Wiederverwendung. So betrachtet Selby (1989) die Wiederverwendung von mehr als 7000 Komponenten (Funktionen, Makros, Datenstrukturen) in einer Software-Entwicklungsabteilung der NASA. Er unterscheidet dabei Wiederverwendung ohne, mit geringen und mit umfangfreichen Modifikationen. Er verwendet allerdings eine Operationalisierung, die von der Semantik der Komponenten abstrahiert. Vielmehr untersucht er die Korrelation von Wiederverwendungshäufigkeit und -form (nach Umfang der Modifikationen) und formalen Eigenschaften der Komponenten wie Schnittstellenkomplexität und Größe.

Dabei sei nur ein Ergebnis zitiert, das für uns noch von Bedeutung sein wird: "Completely reused modules tended to be small ... (Selby 1989, S. 230). Unabhängig davon sollte nicht übersehen werden, daß sich derartige Untersuchungen einem Dilemma gegenübersehen: Einerseits ist es angebracht, die Verheißungen von Wiederverwendbarkeit empirisch zu überprüfen, andererseits sind die so erhobenen Resultate mit Vorsicht zu betrachten. Schließlich ist es unstrittig, daß das Ausmaß der durch Wiederverwendung erzielten Produktivitätsvorteile wesentlich von der Qualität und Brauchbarkeit der verfügbaren Komponenten abhängt. Mit anderen Worten: Wenn beispielsweise in einer Studie eine durchschnittliche Wiederverwendungsrate der betrachteten Komponenten von 2.5 festgestellt wird, ist damit nicht gesagt, daß - bei anderer Gestaltung der Komponenten - keine höhere Rate erzielt werden könnte.

Ex ante-Analysen sind nicht auf die tatsächlich realisierte Wiederverwendung bestimmter Komponenten gerichtet. Sie zielen vielmehr darauf, Ähnlichkeiten zwischen Anwendungen einer bestimmten Art festzustellen, um so Aussagen über Wiederverwendungspotentiale machen zu können. Einige dieser Untersuchungen umfassen eine große Bandbreite von Anwendungen. Sie haben deshalb vor allem Aufschlüsse über systemnahe Komponenten gebracht (Boehm 1981, Emery 1979, Lanergan/Poynton 1979). Im Unterschied dazu analysiert Goodell (1989) allein betriebswirtschaftliche Anwendungen. Er unterscheidet sieben Branchen, unter anderem Banken, Fertigungswirtschaft, Groß- und Einzelhandel. Insgesamt wurden 1338 Programme aus diesen Bereichen betrachtet. Es handelt sich dabei um Anwendungsprogramme aus dem Bestand eines Computerherstellers (Burroughs) Mitte der siebziger Jahre. Die Anwendungen einer Branche können dabei jeweils aus einem oder mehreren Programmen bestehen. So wurden im Bankenbereich 18 Anwendungen (wie "savings", "comsumer loans", "mortgage loans", "general ledger") erfaßt, die durch insgesamt 384 Programme

1. Dunn/Knight (1991), Davis/Fairley/Incorvaia (1990), Selby (1989), Lenz/Schmidt/Wolf (1988) nennen Ergebnisse einer internen Erhebung bei IBM

abgedeckt wurden. Die Untersuchung dieser Programme auf ähnliche bzw. gleiche Funktionsbestandteile ergab, daß 97 Prozent aller Programme mit Hilfe von vierundzwanzig "primary functions" klassifiziert werden konnten. In Abbildung 5 sind diese Funktionen zusammen mit ihrer relativen Häufigkeit für die untersuchten Programme aus dem Bankenbereich dargestellt.[1]



Funktionskategorien

A. Data Entry
B. Edit/validate
C. File maintenance
D. Posting/updating
E. Inquiry
F. Calculate/analyze/simulate
G. Print report
H. Print document
I. Work file create
J. Sort
K. Merge
L. Table file update
M. Clear/reset/roll-forward
N. Purge
O. File conversion
P. File initialize
Q. File build/load/copy
R. File reorganization
S. Item correction
T. Extract
U. System control
V. Device handler
W. Library create
X. Message handler
Y. Miscellaneous

Abb. 5: Die Verwendung von Funktionen in 384 Programmen aus dem Bankenbereich nach der Untersuchung von Goodell (1989, S. 205)

Um zu Aussagen über Ähnlichkeiten zwischen Anwendungen - und damit über wiederverwendbare Konstrukte - zu gelangen, ist der von Goodell gewählte Weg durchaus naheliegend. Wenn bereits Programme vorhanden sind, die die Anforderungen in bestimmten Bereichen widerspiegeln, führt deren Untersuchung zunächst schneller zum Ziel. Dabei ist allerdings zu berücksichtigen, daß die in

1. h := n / g, wobei h die relative Häufigkeit bezeichnet, n die Zahl der Programme, in denen die betreffende Funktion verwendet wurde und g die Gesamtzahl der Programme

den vorgefundenen Programmen implementierten Funktionen nicht unbedingt für die Erfüllung der Anforderungen in den betrachteten Domänen notwendig sind. Neben der möglichen Änderung von Anforderungen im Zeitverlauf ist dabei vor allem an technische Änderungen zu denken. Die Konstrukte, die bei der Erstellung eines Programms verwendet werden, sind eben zum Teil Ausdruck des verwendetetn softwaretechnischen Rahmens (Betriebssystem, Sprache, Bibliotheken und dergleichen). In Goodells Studie gilt dieser Umstand vor allem für Ein- und Ausgabefunktionen: Die meisten der untersuchten Programme waren für Stapelverarbeitung ausgelegt.

Im Hinblick auf den Entwurf von Konstrukten mit einem hohen Maß an Anwendungssemantik sind Goodells Ergebnisse dadurch eingeschränkt, daß er von branchenspezifischen Gegebenheiten abstrahiert und für alle Bereiche die gleichen - immer noch relativ systemnahen - Funktionskategorien betrachtet. Wendet man sich anwendungsnäheren Konstrukten zu (beispielsweise Funktionen zur Kontoführung), ist die Ermittlung von Gemeinsamkeiten über verschiedene Programme hinweg wesentlich schwieriger, weil der jeweils analysierte Quellcode (oder sonstige Dokumentation) eine mehr oder weniger domänenspezifische Sicht reflektiert. Dazu Budde/Züllighoven (1990, S. 117):

> "Die Probleme mit wiederverwendbarer Software liegen nicht darin, daß Moduln nicht für andere Zwecke verwendbar wären, sondern darin, daß es harte Arbeit ist, zu verstehen, ob wir entworfene Software in neuen Perspektiven verwenden können: wir müssen uns den alten Entwurf nahe bringen, mit ihm arbeiten, ihn uns zuhanden machen. Wiederverwendung von Software ist kein Problem eines guten *pattern matching*."

Eine weitere Einschränkung ergibt sich durch die Vernachlässigung struktureller Merkmale (beispielsweise in Form von Datenstrukturen). Ihr Entwurf kann sehr aufwendig sein - was sie zu attraktiven Gegenständen der Wiederverwendung macht, gleichzeitig können sie wesentlich für die Darstellung der Semantik von Funktionen sein, da sie eine anwendungsnähere Beschreibung von Schnittstellen erlauben.

Die Betrachtung der einschlägigen empirischen Untersuchungen existierender Anwendungssysteme zeigt, daß durch solche Arbeiten wichtige Hinweise auf Gemeinsamkeiten geliefert werden können. Gleichzeitig weckt sie Zweifel daran, allein auf diese Weise wiederverwendbare Konstrukte zu ermitteln: Existierende Anwendungssysteme entsprechen nicht immer aktuellen Anforderungen, ihr Entwurf (dabei ist vor allem an die je gewählte Modularisierung zu denken) spiegelt häufig nicht den Stand der Kunst des Software-Engineering wider.

2.2.4 Voraussetzungen für erfolgreiche Wiederverwendung

Angesichts der Plausibilität und Attraktivität der mit Wiederverwendung verbundenen Verheißungen verwundert es, daß das Ausmaß von Wiederverwendung auf anwendungsnahen Ebenen bisher sehr bescheiden ist - zumal es sich hier um eine Idee handelt, die seit langem bekannt ist. Bezeichnend dafür die Stellungnahme Hoppers anläßlich einer Umfrage in einer Fachzeitschrift[1]:

> "Ever since the early days of COBOL, something has disappointed me. I thought that when we designed COBOL, if you remember, we designed the ability to use the library - the subroutine. And I thought the Insurance Association would immediately get together and write all the subroutines necessary for insurance companies to meet the government needs. It's never been done. The banks never wrote that subroutine that would provide for all the reports. And those libraries don't exist. That libraries ability in COBOL has never been used the way it should have been. It should have been one subroutine."

Die sich daran anknüpfende Frage, welche Umstände Wiederverwendung behindern, wird in vielen einschlägigen Veröffentlichungen[2] diskutiert. Die folgende Liste enthält häufig genannte Gründe:

- Entwickler/Programmierer finden ihre wesentliche Befriedigung darin, selbst Code zu schreiben bzw. Entwürfe zu gestalten ("not invented here"-Syndrom).
- Sie sind häufig nicht mit Methoden und Technologien, die Wiederverwendbarkeit besonders unterstützen, vertraut, und sind wenig motiviert, sich entsprechend weiterzuqualifizieren.
- Die verantwortlichen Manager schrecken vor den für die Erstellung wiederverwendbarer Konstrukte zu veranschlagenden Kosten zurück.
- Es ist schwierig, eine zufriedenstellende Gratifikationsregelung für die Bereitstellung wiederverwendbarer Konstrukte zu schaffen.
- Potentiellen Anbietern fehlen die Anreize, solange ein Markt für wiederverwendbare Konstrukte nicht etabliert ist.
- Der Unternehmensführung fehlt das Bewußtsein für die ökonomische Bedeutung von Wiederverwendung. Sie bietet deshalb nicht die nötige Unterstützung.
- Es gibt keine leistungsfähigen Software-Entwicklungs-Werkzeuge zur komfortablen Unterstützung von Wiederverwendung (Suche, Komposition, Modifikation).

Um Ziele zur Überwindung der skizzierten Hindernisse zu formulieren, ist es sinnvoll, solche (Software-) Konstrukte zu betrachten, die - auf anderen Abstrak-

1. Vgl. BYTE, Sept. 1990, S. 318. Eine ähnliche Einschätzung findet sich bei Endres/Uhl (1992, S. 258): "Was bis heute als externes Angebot fast ganz fehlt, sind anwendungsbezogene Klassenbibliotheken."
2. So in den Beiträgen in Tracz (1988 b) oder Sommerville (1989, S. 354 f.)

tionsebenen - von vielen Entwicklern wieder- und wieder verwendet werden. Neben den Bibliotheken höherer Programmiersprachen ist dabei vor allem an Bibliotheken zu denken, die die Implementierung grafischer Benutzerschnittstellen unterstützen. Der entscheidende Grund für den Erfolg solcher Systeme ist darin zu sehen, daß sie den Entwicklern ausgesprochen lukrative Anreize bieten: Sie erlauben nicht nur - vor allem in Verbindung mit geeignenten Entwurfswerkzeugen - eine beeindruckende Steigerung der Produktivität, sondern bieten darüber hinaus eine Funktionalität, deren Implementierung die Kompetenz vieler Entwickler übersteigt. Die Anreize zur Wiederverwendung sind denn auch m.E. der entscheidende Faktor. Nicht nur, daß sie wesentlich zur Überwindung interner psychologischer Widerstände (gegen nicht selbst entworfene Software, gegen eine unvermeidliche Einarbeitung) beitragen, sie schaffen auch eine Nachfrage und damit Anreize für Anbieter, sei es unternehmensintern oder auf Märkten.

Um die Wiederverwendung von Konstrukten attraktiv zu machen, ist es nicht hinreichend, daß diese komplex und qualitativ hochwertig sind. Die Betrachtung dedizierter Ansätze hat nachhaltig gezeigt, daß es vielmehr wichtig, sie zusammen mit einem Modell des jeweiligen Anwendungsbereichs zu präsentiert. Nur so wird es möglich, die streuenden Anforderungen in diesem Bereich in zufriedenstellender Weise abzudecken und gleichzeitig einen terminologischen Bezugsrahmen zu bieten, der die Suche nach einzelnen Konstrukten unterstützt. Daneben ist zu berücksichtigen, daß die Komponentenmetapher im Kontext der Software-Entwicklung nur unzureichend greift: Durch die häufige Notwendigkeit, Beziehungen zwischen Komponten zu definieren, kann deren Semantik nicht autonom betrachtet werden. Darüber hinaus sind Modelle von Anwendungsdomänen notwendige Voraussetzung für die Bereitstellung geeigneter Werkzeuge, denn solche Werkzeuge sollten eine Schnittstelle zu den wiederverwendbaren Konstrukten bieten, die in anschaulicher Weise mit gängigen Konzeptualisierungen des jeweiligen Realitätsbereichs korrespondieren.

Um einerseits komplexe anwendungsnahe Konstrukte anbieten zu können (deren Verwendung, wenn sie denn brauchbar sind, einen besonders hohen Produktivitätsgewinn verspricht) und andererseits individuelle Anpassungsmöglichkeiten zu schaffen, ist neben einer komfortablen Modifikation der Komponenten eine hierarchische Verdichtung von Domänen anzustreben.

Nur dann, wenn anwendungsnahe Konstrukte eine spezifische Anforderung nicht erfüllen, sollte auf eine darunterliegende Ebene zugegriffen werden (vgl. Abb. 6). Rumbaugh et al. (1991, S. 200) unterscheiden in diesem Zusammenhang "open architectures", die einen Durchgriff auf die diversen Implementierungsebenen erlauben und "closed architectures", die allein die Nutzung von Diensten bestimmter Ebenen zulassen.

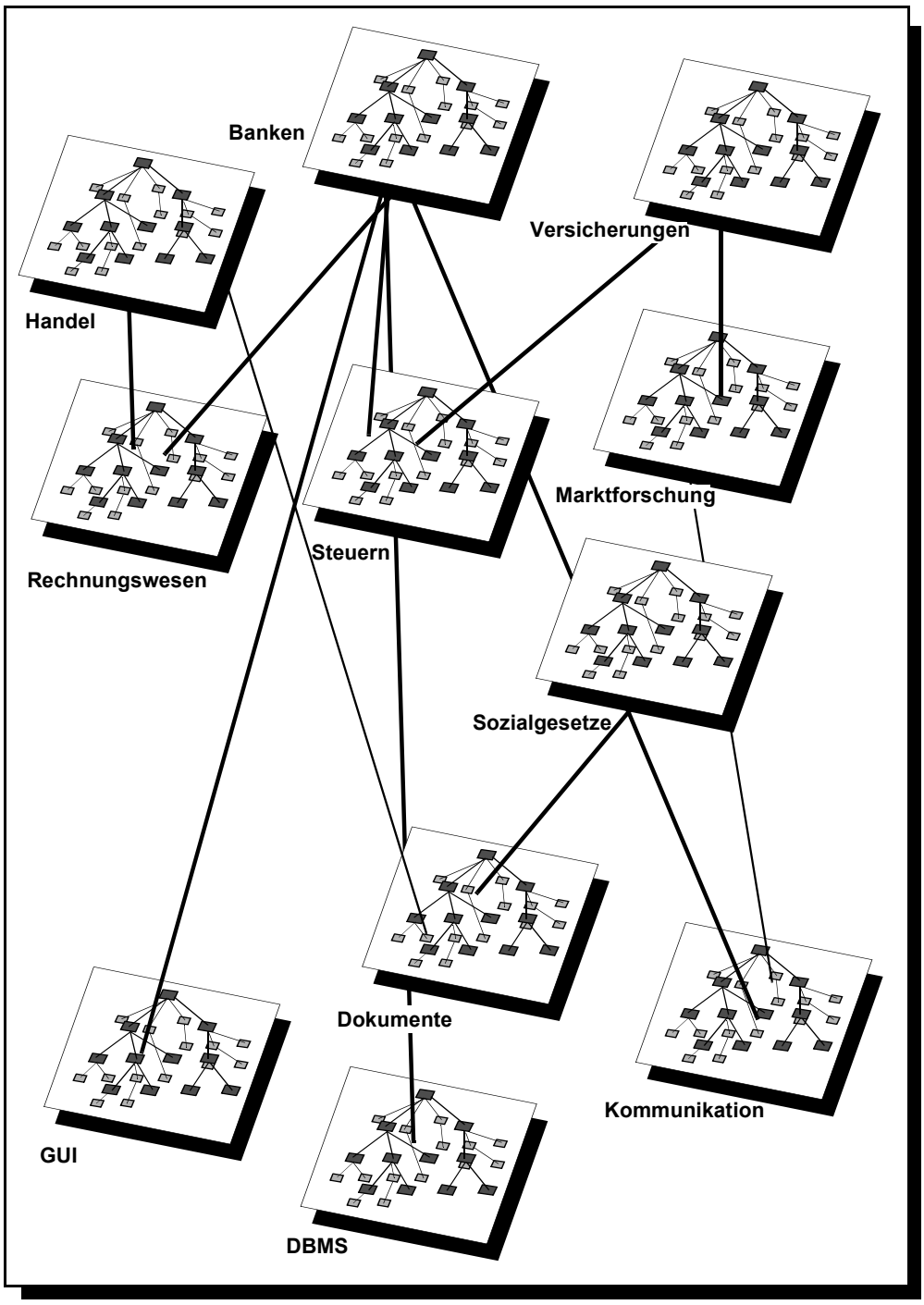


Abb. 6: Beispiele für die Abgrenzung domänenspezifischer Modelle

Die Komplexität eines solchen Vorhabens kann nur in einem längeren, evolutorischen Prozeß bewältigt werden.[1] In der Betriebswirtschaftslehre gibt es durchaus

1. Vgl. dazu Balzer (1984) und die Ausführungen in VI.3.

Parallelen zu einem solchen Vorgehen, das - zunächst - einer detaillierten Untersuchung spezifischer Unternehmen gegenüber einer generell-spekulativen Betrachtung den Vorrang gibt. Hier ist an das von Schmalenbach vorgeschlagene Musterkontor oder an Fallstudien neuerer Zeit zu denken - Ansätze, die nicht allein einen didaktischen Wert besitzen, sondern auch eine heuristische Funktion für die Wahrnehmung unternehmensübergreifender Gemeinsamkeiten aufweisen.

Die Betrachtung eines solchen idealtypischen Prozesses vor dem Hintergrund der sozialen Realität der Software-Entwicklung macht deutlich, daß es sich hier nicht allein um die intellekturelle Herausforderung handelt, brauchbare Generalisierungen zu finden, sondern auch darum, eine *Einigung* über konkurrierende Entwürfe zu erzielen. Die mit Wiederverwendbarkeit verbundene ökonomische Attraktivität steht und fällt mit der Zahl derjenigen, die die im Rahmen eines Domänenmodells angebotenen Konstrukte nutzen. Nur dann, wenn diese Zahl hinreichend groß ist, um den hohen Entwicklungsaufwand in einer für alle Beteiligten zufriedenstellenden Weise zu decken, kann sich die nötige Investitionsbereitschaft einstellen. Betrachtet man das skizzierte evolutorische Entwicklungsmodell, ist die Verpflichtung späterer Nutzer nicht nur nötig, um Investitionsrisiken zu reduzieren, sondern auch um die rechtzeitige Beteiligung (ausgewählter) Nutzer) im Entwurfsprozeß zu sichern. Für das einzelne (Nutzer-) Unternehmen ist ein solches Engagement ambivalent: Der Verheißung leistungsfähigerer und gleichzeitig kostengünstigerer Informationssysteme steht das Risiko gegenüber, daß das gemeinsame Vorhaben scheitert. Ein Problem, das an die eigentümliche Dialektik der Standardisierungsrationalität erinnert: Für den einzelnen kann es sehr vernünftig sein, wiederverwendbare Konstrukte zu nutzen. Gleichzeitig mag es fragwürdig sein, sich vorab in geeigneter Form an deren Entwicklung zu beteiligen - obwohl ein solches Engagement wesentliche Voraussetzung dafür ist, zu Entwürfen zu gelangen, die den jeweiligen Anforderungen gerecht werden. Eine Strategie, die auf die Verbreitung anwendungsnaher Konstrukte zielt, muß dieser Schwierigkeit Rechnung tragen - auch wenn eine völlige Auflösung des dargestellten Dilemmas kaum zu erwarten ist.

2.2.5 Wiederverwendbarkeit und Semantik - ein Antagonismus?

Die mit Wiederverwendbarkeit verknüpften Hoffnungen gründen wesentlich auf zwei Vorstellungen:

- Durch die Bereitstellung wiederverwendbarer Konstrukte läßt sich die Produktivität von Entwicklern erheblich erhöhen.
- Vielfache Wiederverwendung qualitativ hochwertiger, mit relativ großem Aufwand erstellter Konstrukte ermöglicht attraktive Einstandspreise.

Zwischen den mit diesen Aspekten verbundenen Zielen besteht allerdings ein

Konflikt. Um den einzelnen Entwickler zu entlasten, sind Konstrukte wünschenswert, die dessen spezifische Anforderungen weitgehend abdecken. Je spezfischer Konstrukte sind, desto geringer sind jedoch ihre Wiederverwendungsmöglichkeiten. Graham (1991, S. 239) sieht sich durch diesen Umstand zu der Feststellung verleitet: "The fact is that all semantics compromise reuse". Eine solche Einschätzung kann allerdings kaum uneingeschränkt akzeptiert werden, da sie nur auf einen Aspekt von Wiederverwendbarkeit, nämlich die Wiederverwendungshäufigkeit, rekurriert. Wörtlich genommen führt sie zu dem Schluß, daß man Anwendungssemantik in wiederverwendbaren Konstrukten vermeiden sollte. Ein solcher Ansatz, der seinen Niederschlag unter anderem in den generellen Funktionsbibliotheken höherer Programmiersprachen findet, ist aber allein nicht befriedigend. Der damit verbundene Konflikt wird von Hutchins/Holland/Norman (1986, S. 101 u. 103) nachdrücklich betont:

> "Beware the Turing tar-pit in which everything is possible but nothing of interest is easy. ... Beware the over-specialized systems where operations are easy, but little of interest is possible."

Macht es angesichts dieses Konflikts überhaupt Sinn, auf Wiederverwendbarkeit als Orientierung zu setzen? Sicherlich nur dann, wenn es eine Chance gibt, den Konflikt zu entschärfen.

Ein ähnliches Problem ist in den Wissenschaften seit langem bekannt. Ein Reflex darauf sind epistemologische Bemühungen, Bewertungsmaßstäbe für Theorien zu entwickeln. Jenseits der damit verbundenen Kontroverse um unterschiedliche Auffassungen von Theorien[1] ist für unsere Betrachtung vor allem die Frage nach dem Informationsgehalt (also der Semantik) einer Aussage von Bedeutung. Da vor dem Hintergrund der Software-Entwicklung zudem intensionale Semantik (vgl. II.1.3.2) hier keine Rolle spielt, reicht die bereits erwähnte Auffassung rationalistischer Wissenschaftstheorie, wonach der Informationsgehalt einer Aussage über einen Sachverhalt mit der Zahl der dadurch jeweils ausgeschlossenen denkmöglichen Konstellationen steigt. Die Aussage, daß alle Unternehmen in Deutschland durch die Rechtsform der GmbH gekennzeichnet sind, schließt wesentlich mehr andere Konstellationen aus als eine entsprechende Aussage für ein spezielles Unternehmen. Der positive Effekt eines hohen Informationsgehalts ist allerdings mit einem erhöhten Risiko verbunden, durch empirische Überprüfung diskreditiert zu werden. Im Unterschied zu den Naturwissenschaften sind in den Sozialwissenschaften empirisch nicht widerlegte Theorien mit hohem Informationsgehalt selten anzutreffen.

Um die widerspenstige Realität dennoch - indirekt - einer logisch-systematischen

1. Hier ist unter anderem an rationalistische und hermeneutische Ansätze zu denken - und die mit ihnen teilweise korrelierenden Vorstellungen über die gesellschaftliche Funktion von Wissenschaft.

Analyse zugänglich zu machen, wurden vor allem in den Wirtschaftswissenschaften Konstruktionen (ceteris paribus-Klauseln, vereinfachende Operationalisierungen etc.) zur Reduktion von Komplexität eingeführt. Wie immer man den erkenntnistheoretischen Wert solcher Konstruktionen einschätzt, die auf diese Weise entstehenden Theorien sind zumeist nicht (unmittelbar) auf ein konkretes Unternehmen zu übertragen. Ein Umstand, der in einer anwendungsorientierten Disziplin wie der Betriebswirtschaftslehre besonders schwer wiegt. Bisher ist es der Betriebswirtschaftslehre allenfalls in bescheidenem Maße gelungen, Theorien der Unternehmung zu entwickeln, die informativ und zudem - für die Software-Entwicklung wichtig - formalisierbar sind. Das legt die für unsere Untersuchung wenig erbauliche Folgerung nahe, daß es kaum gelingen kann, Konstrukte zur Gestaltung betrieblicher Informationssysteme zu entdecken, die den Anforderungen einer größeren Zahl von Unternehmen gerecht werden.

Auch wenn eine solch skeptische Sicht wichtig ist, um allzu positivistischen Vorstellungen über die Erfaßbarkeit (oder das Vorhandensein) genereller Gemeinsamkeiten entgegenzuwirken, so ist sie dennoch nicht hinreichend, um die Hoffnung auf wiederverwendbare Konstrukte auf einem hohen semantischen Niveau zunichte zu machen. Bemühungen um Theorien in der Betriebswirtschaftslehre sind daran orientiert, handlungs- oder entscheidungsrelevante Zusammenhänge aufzudecken, um so - durch eine mehr oder weniger aufwendige technologische Transformation - Empfehlungen ableiten zu können.

Die dabei auftretenden Schwierigkeiten[1] können jedoch nicht darüber hinwegtäuschen, daß es Sachverhalte gibt, die in vielen Unternehmungen in gleichen oder sehr ähnlichen Ausprägungen vorliegen. Hier ist beispielsweise an das Rechnungswesen, die Personalwirtschaft oder das Steuerwesen zu denken. Auch wenn die empirische Erforschung von Invarianzen in diesen Bereichen für die Betriebswirtschaftslehre nicht von vorrangigem Interesse ist (nicht zuletzt, weil sie mehr oder weniger evident, zum Teil sogar gesetzlich vorgeschrieben sind), für den Entwurf betrieblicher Informationssysteme sind sie umso wichtiger.

Darüber hinaus ist zu berücksichtigen, daß das Bemühen um mehrfach verwendbare Konstrukte in der Software-Entwicklung nicht unbedingt das Vorhandensein tatsächlicher Gemeinsamkeiten in Unternehmen voraussetzt. So ist es einerseits wenig sinnvoll, ineffiziente Strukturen und Abläufe abzubilden - es gibt also neben der deskriptiven auch eine präskriptive Komponente. Andererseits gilt ohnehin - auch in einem einzelnen Unternehmen, daß der Entwurf von Informationssystemen nicht allein eine (authentische) Rekonstruktion der Realität ist, sondern selbst - in Teilen - eine neue Wirklichkeit schafft.[2] Jenseits solcher metho-

1. Gutenberg (1953, S. 340 f.) hält wegen der Komplexität und Individualität realer Unternehmen schon das Ansinnen, Empfehlungen geben zu wollen, für "klein und eng".

dologischer Reflektionen kann hier auch auf empirische Evidenz verwiesen werden: Der Einsatz der von einer wachsenden Zahl von Unternehmen verwendeten betriebswirtschaftlichen Anwendungspakete impliziert in der Regel eine mehr oder weniger umfangreiche organisatorische Anpassung sowie das Erlernen einer neuen Terminologie. Daraus folgt für unsere Betrachtung: Eine um Verbesserung des status quo bemühte Konstruktion von Informationssystemen ist zwar mit einer zusätzlichen Herausforderung verbunden, gleichzeitig schafft sie aber auch eine Chance dafür, bestehende Varianzen zwischen Unternehmensanforderungen durch vielfach akzeptierte Gestaltungsvorschläge zu überwinden.

Außerdem ist daran zu denken, daß es für die Gestaltung wiederverwendbarer Konstrukte nicht notwendig ist, generelle Muster in einer Grundgesamtheit von Unternehmen zu entdecken oder vorzuschlagen. Vielmehr kann (unerklärte) Varianz von Anforderungen durch eine geeignete Vielfalt von Konstrukten abgedeckt werden. Wenn also beispielsweise die Fakturierung in Großhandelsunternehmen nicht einheitlich modelliert werden kann, kann zunächst die Grundgesamtheit systematisch eingeschränkt werden. Wenn auch das nicht zu brauchbaren Vereinheitlichungen führt, könnte immer noch eine Menge unterschiedlicher Konstrukte angeboten werden, aus denen auszuwählen dann Aufgabe der Entwickler spezifischer Systeme wäre.

3. Zusammenfassende Beurteilung: Zum Verhältnis von Integration und Wiederverwendbarkeit

Die bisherige Untersuchung hat gezeigt, daß sowohl Integration als auch Wiederverwendbarkeit die Etablierung von Modellen voraussetzen: Auf der einen Seite semantische Referenzmodelle, die eine leistungsfähige Kommunikation zwischen den Teilen eines Informationssystems möglich machen, auf der anderen Seite Domänenmodelle, die eine systematische Präsentation wiederverwendbarer Konstrukte erlauben. Auch wenn es sich damit je unterschiedliche Funktionen der Modelle im Vordergrund stehen, bei eingehender Betrachtung wird deutlich, daß Integration und Wiederverwendbarkeit zwei Seiten einer Medaille sind. Oder - anders formuliert: Sie befördern sich gegenseitig. So sind die Elemente eines semantischen Referenzmodells wie etwa Datenstrukturen, Funktionen oder Klassen ja durch das Bemühen gekennzeichnet, systemweit nur einmal definiert und implementiert zu sein (unabhängig von der Zahl der physischen Kopien). Die sich darin ausdrückende Redundanzminimierung ist gleichzeitig konstitutiv für das Bemühen um Wiederverwendung. Dieser Umstand wird deutlich, wenn man sich Systeme ansieht, die nach dem gegenwärtigen Stand der Technik als besonders weit integriert angesehen werden können. Ein Beispiel dafür sind Betriebs-

2. Eine ausführliche Diskussion dieser konstruktivistischen Interpretation von Software-Entwicklung findet sich in Floyd (1989) und Winograd/Flores (1988).

systeme, die eine grafische Benutzerschnittstelle beinhalten. Sie fördern einerseits die Wiederverwendung. Dies gilt zumeist nicht allein für die zur Implementierung von Fenstern und Widgets bereitgestellten Bibliotheken, sondern auch für Vorgaben zur ergonomischen Gestaltung von Benutzerschnittstellen (in Form sogenannter Style-Guides). Andererseits wird die Integration von Anwendungen erleichtert. Wenn beispielsweise in einer Anwendung eine Grafik erzeugt und dargestellt wird, kann diese Grafik in einer anderen Anwendung unter Verwendung des gleichen Widgets in gleicher Weise präsentiert werden. Neben diesen technischen Aspekten wird durch die Verwendung gleichartiger Dialogelemente auch eine Integration der Anwendungen aus der Sicht des Benutzers unterstützt.

Nach der in dieser Arbeit vertretenen Sicht nimmt Integration mit der Semantik der Elemente eines Referenzmodells zu. Auch die Qualität von Wiederverwendbarkeit ist mit der Semantik der entsprechenden Konstrukte korreliert: Je mehr Semantik sie bereits enthalten (also je anwendungsnäher sie sind), desto weniger muß der sie verwendende Entwickler noch selbst hinzufügen, desto höher also tendenziell seine Produktivität. In beiden Fällen sind aber auch dysfunktionale Effekte von Semantik denkbar. Wenn zur Integration der Teile eines Informationssystems auf semantisch stark angereicherte Elemente referiert wird, ist dies einerseits unter Umständen mit einem großen Aufwand für die Definition dieser Semantik verbunden, andererseits wird mit zunehmender Spezialisierung die Reichweite der Integration eingeschränkt (vgl. II.2.2.4). Ähnliches gilt für die Wiederverwendbarkeit von Konstrukten: Je mehr Semantik sie enthalten, desto geringer die Zahl möglicher Einsatzfelder. Dabei ist allerdings zu berücksichtigen, daß die Aussicht auf ein höheres Integrationsniveau die Entwicklung wiederverwendbarer Konstrukte erst sinnvoll erscheinen läßt, umgekehrt die Verfügbarkeit solcher Konstrukte den Aufwand für die Etablierung komplexer Referenzmodelle auf ein akzeptables Maß reduziert. Ein Beispiel dafür sind Compound Document Architectures. Allein die Definition einer solchen Architektur, von der Implementierung zugehöriger Software ganz zu schweigen, wäre für einzelne Unternehmen in der Regel mit einem unzumutbaren Aufwand verbunden. Sobald aber entsprechende Komponenten verfügbar sind, kann der Aufwand unter Umständen auf ein akzeptables Maß reduziert werden.

Dem Konflikt zwischen dem Bedürfnis nach einem hohen semantischen Niveau auf der einen Seite, den einschränkenden Wirkungen von Semantik auf der anderen Seite, kann mit hierarchisch geschichteten Modellen begegnet werden. Für deren Verwendung gilt in beiden Fällen eine ähnliche Regel. Wenn verschiedene Teile integriert werden sollen, sollte dazu auf Elemente auf dem höchstmöglichen Niveau referiert werden. Nur wenn einzelne Teile dazu nicht in der Lage sind oder aber auf einem hohen Niveau für den jeweiligen Kontext kein geeignetes Referenzelement verfügbar ist, ist ein tiefere Stufe zu wählen. Bei der Ent-

wicklung einer Anwendung sollten ebenfalls Konstrukte, die bereits viel Anwendungssemantik enthalten, verwendet werden. Nur wenn spezielle Anforderungen durch die verfügbaren Konstrukte auf dieser Ebene nicht abgedeckt werden können, ist auf darunterliegende Ebenen zuzugreifen.

Auch im Zusammenhang mit der Überwindung der Friktionen zwischen den einzelnen Phasen der Software-Entwicklung zeigen sich deutliche Parallelen. Zur Integration dieser Phasen ist ein Referenzssystem erforderlich, das Elemente enthält, auf die in allen Phasen (soweit sie nötig sind) verwiesen werden kann. Wenn beispielsweise in der Analyse, im Entwurf und auch zur Implementierung auf ein Konstrukt "Kunde" referiert werden kann, ist die schrittweise - mit Aufwand und Risiko verbundene - Transformation dieses Konstrukts von der Repräsentationsform einer Phase in die der jeweils nachfolgenden obsolet. Die Idee von Wiederverwendbarkeit auf einem anwendungsnahen Niveau zielt genau darauf, solche Konstrukte bereitzustellen. Beim Prototyping wird dieser Zusammenhang genutzt, um die traditionellen Lücken zwischen Analyse und lauffähigem System möglichst schnell zu überbrücken.[1] Während Prototypen häufig nur als Vorlage für anschließende (Re-) Implementierungen dienen, könnten sorgfältig entworfene und implementierte anwendungsnahe Komponenten im besten Fall eine Überwindung der Grenzen zwischen Prototyping und Implementierung ermöglichen.

Die Betrachtung des Integrationsbegriffs hat gezeigt, daß es für den Einsatz betrieblicher Informationssysteme empfehlenswert ist, nicht nur technische Dimensionen von Integration zu betrachten. Dabei ist an Unternehmensorganisations im allgemeinen, die organisatorische Integration von Informationssystemen im besonderen zu denken. Um die Komplexität der damit verbundenen Gestaltungsaufgaben zu reduzieren, ist es - aus oben näher erläuterten Gründen - hilfreich, Modelle einzuführen, in denen die wesentlichen Zusammenhänge in anschaulicher und zweckdienlicher Weise dargestellt werden. Dazu ist es naheliegend, *ein* Modell (also die zu seiner Darstellung verwendeten Konstrukte und Beziehungen) zu entwickeln, in dem den Anforderungen organisatorischer Gestaltung ebenso Rechnung getragen wird wie softwaretechnischen Randbedingungen. Da allerdings die Gestaltung von Informationssystemen ein spezielles, an die restriktiven Möglichkeiten formaler Systeme gebundenes, Abstraktionsniveau nötig macht und in der Regel nicht alle für die Unternehmensorganisation bedeutsamen Aspekte zu berücksichtigen hat, empfiehlt das Bemühen um Integration unter Umständen die Einführung von Teilmodellen. Also beispielsweise

1. Honiden et al. (1988) präsentieren einen Ansatz zum Prototyping, in dem der Einsatz wiederverwendbarer Komponenten im Mittelpunkt steht. Aber auch in solchen Prototyping-Ansätzen, in denen nicht explizit auf wiederverwendbare Konstrukte hingewiesen wird, spielen diese in der Regel eine zentrale Rolle.

ein Teilmodell, in dem die Unternehmensorganisation dargestellt wird, das vielleicht wieder unterteilt ist in ein Modell der Ablauf- und eines der Aufbauorganisation. Daneben könnte ein Informationssystem-Modell stehen, dessen Darstellung ebenfalls eine statische und eine dynamische Sicht differenziert. Durch eine anschauliche Präsentation solcher Modelle wäre bereits ein Beitrag zur organisatorischen Integration von Informationssystemen geleistet: Sie erleichtert den Gestaltern - von Software auf der einen, von Unternehmensorganisation auf der anderen Seite - das Verstehen der jeweils anderen Ebene und unterstützt damit eine angemessene gegenseitige Anpassung. Die integrationsfördernde Wirkung von Teilmodellen läßt sich zudem erhöhen, wenn Beziehungen zwischen ihnen etabliert werden. So kann ein Betrachter, der mit den Konstrukten eines Modells vertraut ist, auf die Darstellung des entsprechenden Konstrukts in einem anderen Modell verzweigen. Also beispielsweise von der Beschreibung eines Beschaffungsvorgangs im Modell der Unternehmensorganisation auf die zugehörige Vorgangsbeschreibung im Modell des Informationssystems.

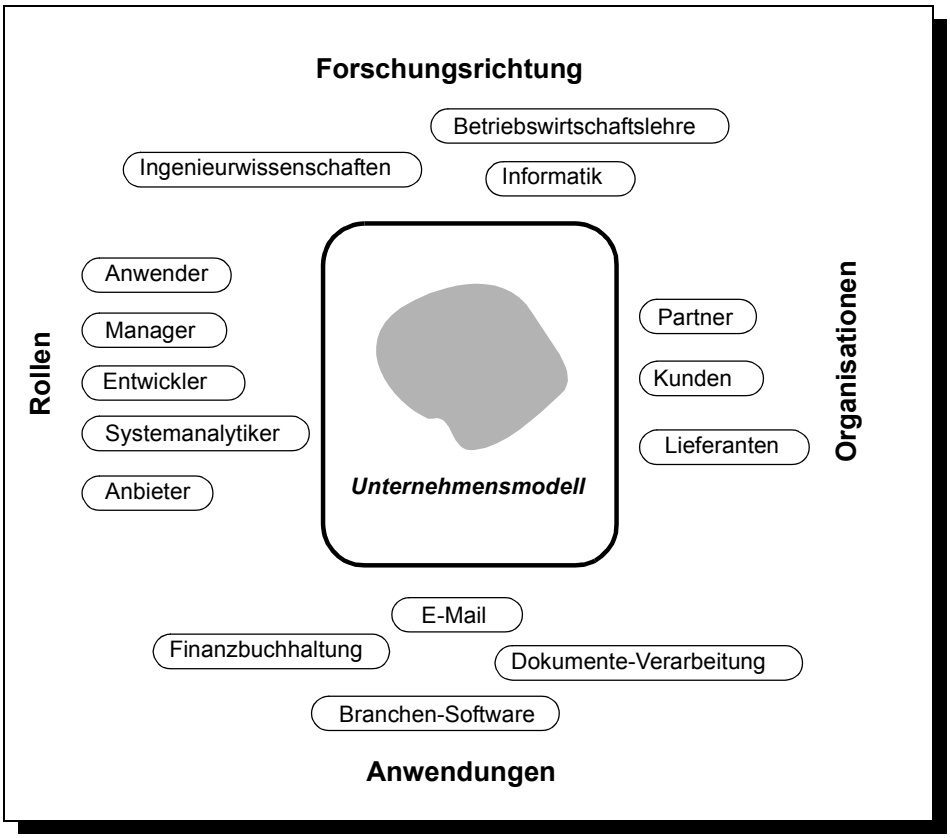


Abb. 7: Durch die Einführung von (Teil-) Modellen der Unternehmung geförderte Dimensionen von Integration

Auch Wiederverwendung sollte nach gängiger Überzeugung nicht allein auf softwaretechnische Konstrukte beschränkt sein. Gerade im Zusammenhang mit betrieblichen Informationssystemen ist es wünschenswert, Wissen über das Unternehmen und sein Umfeld für zukünftige Verwendungen in geeigneter Form zu konservieren.[1] Das betrifft einerseits Darstellungen spezifischer Unternehmensgegebenheiten - etwa in Form eines detaillierten Organisationsmodells. Andererseits ist hier auch an generalisierte Zusammenhänge oder an analytisches Wissen zu denken, also beispielsweise an Verfahren und Konzepte, die in der Betriebswirtschaftslehre entwickelt werden. Die Bereitstellung solchen Wissens in Form von Modellen, die werkzeuggestützt auf Rechnern verwaltet werden, verspricht eine Reihe von Vorteilen. So erleichtert ein einheitlich dargestelltes Modell im Wortsinn die Einordnung neuen Wissens. Für den möglichen Nutzer erhöht sich damit die Chance, solches Wissen in angemessener Weise beurteilen zu können. Die Verwendung von Werkzeugen verspricht zudem einen erleichterten und schnelleren Zugriff. Die damit kurz skizzierten Potentiale für eine Verbesserung des Verhältnisses der Betriebswirtschaftslehre zu ihrer Praxis sollten allerdings nicht allein im Hinblick auf unidirektionalen Theorietransfer gesehen werden. Rechnergestützte Modelle von Unternehmen bieten zudem die Chance, ein Medium für einen wirksamen Dialog zwischen Wissenschaft und Praxis zu schaffen. Ähnliches gilt für die Förderung der interdisziplinären Zusammenarbeit der Forscher, die aus unterschiedlichen Blickwinkeln mit der Gestaltung, Einführung und Bewertung betrieblicher Informationssysteme beschäftigt sind.

1. Diese Aussage ist natürlich durch den Hinweis darauf einzuschränken, daß - wie grundsätzlich bei der Schaffung wiederverwendbarer Artifakte - ein ökonomisch sinnvolles Verhältnis von Aufwand und Ertrag erreicht werden kann.

III. Überblick über Ansätze zur informationsorientierten Modellierung von Unternehmen

"Data models are central to information systems."

Michael L. Brodie

"We observe that this inherent complexity derives from four elements: the complexity of the problem domain, the difficulty of managing the developmental process, the flexibility possible through software, and the problems of characterizing the behavior of discrete systems."

Grady Booch

Die bisherige Untersuchung war darauf gerichtet, Integration und Wiederverwendbarkeit als zentrale Orientierungen im Detail zu betrachten. Dabei hat sich unter anderem gezeigt, daß beide Orientierungen den Entwurf von Modellen als Abstraktionen der jeweils relevanten Domänen erfordern. Solche Modelle dienen einerseits als Grundlage der Software-Entwicklung, andererseits bieten sie günstige Voraussetzungen auch für die Unterstützung der organisatorischen Integration von Informationssystemen und können als ein Medium für organisatorische Entwicklungsprozesse dienen.

Die Bedeutung von Modellen für die Entwicklung integrierter Anwendungssysteme ist seit langem bekannt. Dieser Umstand spiegelt sich in einer Vielzahl von Modellierungsmethoden und -techniken sowie mit ihnen einhergehenden Werkzeugen. Sie haben eine betont softwaretechnische Orientierung und werden gemeinhin unter dem Sammelbegriff "konzeptuelle[1] Modellierung" geführt. Um zu näheren Aufschlüssen über die Beurteilung solcher Ansätze vor dem Hintergrund der oben skizzierten Anforderungen zu gelangen, wird zunächst - vor dem Hintergrund eines noch zu entwickelnden Kriterienkatalogs - die Evolution der konzeptuellen Modellierung in ihren wesentliche Zügen rekonstruiert.

Die Methoden und Techniken der konzeptuellen Modellierung sind bewußt so angelegt, daß sie von den spezifischen Anforderungen einer bestimmten Anwendung abstrahieren - sie sollen eben möglichst universell einsetzbar sein. Für die Entwickler des Modells einer bestimmten Anwendungsdomäne bleibt dann immer noch ein großer Aufwand. Vor allem die Orientierung an Wiederverwendbarkeit legt es nahe, den Entwicklungsaufwand durch die Bereitstellung anwen-

1. wobei anstelle von "konzeptuell" - ohne erkennbare Bedeutungsverschiebung - auch mitunter "konzeptionell" verwendet wird.

dungsnaher Modelle zu reduzieren. Der Betrachtung entsprechender Ansätze dient der zweite Teil dieses Kapitels. Zunächst jedoch sollen Evaluationkriterien entwickelt werden, die es gestatten, die verschiedenen Ansätze hinsichtlich ihrer Potentiale zur Unterstützung von Integration und Wiederverwendbarkeit zu beurteilen.

1. Ein abgeleiteter Bezugsrahmen für die Beurteilung von Modellierungsansätzen

Die Evaluation von Methodologien ist im allgemeinen eine problematische Aufgabe. Für technikgestützte Entwurfsverfahren gilt dies ganz besonders: Das Urteil derjenigen, die den Umgang mit ihnen mühsam erlernt haben, die sie verwenden und die sie deshalb besonders gut kennen, ist dadurch getrübt, daß sie die mit solchen Verfahren einhergehenden Konzeptualisierungen mehr oder weniger internalisiert haben. Ihre Wahrnehmung ist also verzerrt. Die auf diese Weise entstehenden Meinungsverschiedenheiten zwischen den Anhängern verschiedener Ansätze führen bisweilen dazu, daß eine Einigung auf gemeiname Bewertungskriterien nahezu unmöglich erscheint - ein bekanntes Beispiel dafür ist die mitunter ebenso pathetisch wie apologetisch geführte Diskussion um die Leistungsfähigkeit von Programmiersprachen. In diesem Umstand artikuliert sich allerdings nicht allein ein Mangel an Objektivität, vielmehr ist er ein notwendiger Reflex darauf, daß die Einschätzung von Modellierungsansätzen nicht losgelöst von den persönlichen Präferenzen und sonstigen Prädispositionen der Verwender betrachtet werden kann. Schließlich geht es auch um die Beziehung von Modellierungsmethode und Modellierer, um die Etablierung produktiver Arbeitsstile. Die im folgenden präsentierten Kriterien klammern solche subjektiven Einstellungen weitgehend aus - ohne daß ihre Bedeutung damit geschmälert ist. Die Kriterien leiten sich mehr oder weniger eng aus den Anforderungen an Integration und Wiederverwendbarkeit ab. Sie zielen in erster Linie auf eine konzeptionelle Beurteilung. Andere Kriterien, die für Entscheidungen über einen praktischen Einsatz ebenfalls von Bedeutung sind (wie beispielsweise Qualifikation des verfügbaren Personals, Unterstützung durch Anbieter etc.) bleiben unberücksichtigt.

Um einen ersten Beurteilungsrahmen zu erhalten, werden im folgenden Kriterien entwickelt, die sich aus den oben erläuterten Anforderungen an Integration und Wiederverwendbarkeit ableiten lassen.

Grundsätzliche Kriterien

- Anschaulichkeit: Die Anschaulichkeit eines Modellierungsansatzes hängt ab von den verfügbaren Präsentationsformen und deren Verständlichkeit. Beide Kriterien können angewendet werden auf die zur Modellierung angebotenen Konstrukte und die Kompositionsregeln bzw. die Durchführung der

Komposition. Dazu sollten die Konstrukte wie auch die aus ihnen zusammengefügten Modelle möglichst unmittelbar mit den Konzeptualisierungen des jeweiligen Realitätsausschnitts korrespondieren. Die Präsentation sollte dazu beitragen, daß diese Korrespondenz leicht zu erkennen ist. Da es im Kontext des Entwurfs, der Implementierung und Nutzung betrieblicher Informationssysteme eine Reihe zum Teil sehr unterschiedlicher Perspektiven gibt, ist es unter Umständen notwendig, mehrere Modelle eines Anwendungsbereichs, in denen jeweils unterschiedliche Konzepte im Vordergrund stehen, zu erstellen. Je nach Verwendungskontext kann es wichtig sein, neben einer abstrakteren Sicht einzelne Teile eines Modells im Detail zu betrachten. Modelle sollten deshalb verschiedene *Detaillierungsstufen* bieten.

- Views: Während Integration eine ganzheitliche Betrachtung des Systems empfiehlt, ist es dennoch wichtig, einzelne Nutzungskontexte bzw. Views als solche modellieren zu können. Das fördert zum einen die Anschaulichkeit des Modells für einen bestimmten Benutzer, zum anderen ist die Beschreibung von Views Grundlage für die Definition von Datenschutzmechanismen. Darüber hinaus gibt sie Hinweise für die Gestaltung von Benutzerschnittstellen.
- Semantisches Niveau: Um eine enge Korrespondenz mit Teilen einer Anwendungsdomäne herzustellen, ist es wichtig, daß die verfügbaren Konstrukte es erlauben, ein *hohes Maß an Anwendungssemantik* abzubilden. Gleiches gilt für die Beziehungen, die zwischen den Konstrukten definiert werden können. In diesem Zusammenhang ist zu fordern, daß möglichst viele der in einer Domäne erkannten Gleichförmigkeiten mit Hilfe geeigneter *Generalisierungen* dargestellt werden können.
- Darstellungsmächtigkeit/Flexibilität: Das Bemühen um Integration der Teile eines Informationssystems empfiehlt ein gemeinsames semantisches Referenzsystems. Ein Modellierungsansatz sollte deshalb für möglichst alle Teile geeignet sein - also für die Bearbeitung von Dokumente ebenso wie für die Fakturierung. Außerdem sollte es möglich sein, alle wesentlichen Aspekte des Entwurfs zu berücksichtigen. Hier ist beispielsweise an die Modellierung der Benutzerschnittstelle zu denken. Es liegt auf der Hand, daß Darstellungsmächtigkeit und hohes semantisches Niveau der Konstrukte keine komplementären Kriterien sind. Je anwendungsnäher ein Konstrukt, desto eingeschränkter eben sein Einsatzbereich. Um diesen Konflikt zu entschärfen, ist es wünschenswert, daß Modellierungskonstrukte auf verschiedenen, aufeinander aufbauenden Semantik-Stufen angeboten werden. Der Modellierer verwendet dann vorzugsweise solche Konstrukte, die auf der jeweils höchsten semantischen Stufe verfügbar sind. Nur dann, wenn sich dort keine geeigneten Konstrukte finden, wendet er sich einer darunter

liegenden Ebene zu.

- Formalisierung: Die zur Modellierung verwendeten Konstrukte sollten wohldefiniert sein. Gleiches gilt für die Regeln zur Erstellung eines Modells. Es ist zudem wünschenswert, die Zahl der Konstrukte klein und die Konstruktionsregeln einfach zu halten.
- Unabhängigkeit von technischem Wandel: Auch wenn Modelle der Software-Entwicklung und damit letztlich der Implementierung dienen, sollten sie es gestatten, von technischen Aspekten zu abstrahieren - sofern diese einem häufigem Wandel ausgesetzt sind.

Integration der Phasen des Life-Cycle

- Gemeinsame Referenzen: Das Bemühen um die Integration der verschiedenen Phasen des Software-Lifecycle empfiehlt die Verwendung *gleicher oder ähnlicher Konstrukte von der Analyse bis zur Implementierung*. Es sollte dabei möglich sein, diese Konstrukte - um den unterschiedlichen Anforderungen der einzelnen Phasen gerecht zu werden, in variierenden Detaillierungsformen zu präsentieren.
- Stufenweise Formalisierung/Detaillierung: Auch wenn nicht auf bereits vorhandene Konstrukte zurückgegriffen werden kann, sollte dennoch die durchgehende Verwendung eines Konstrukts in allen Phasen möglich sein. Da während der Analyse in der Regel von den Details anschließender Phasen abstrahiert werden soll und eine Formalisierung schon während der Analyse in der Regel zu aufwendig ist, ist es in diesem Zusammenhang wünschenswert, das in der Analysephase erstellte Modell schrittweise weiter detaillieren und formalisieren zu können.
- Vermeidung inkonsistenter Phasenübergänge: Da der Entwurf von Modellen komplexer Anwendungsdomänen ein evolutorischer Prozeß ist, sollten die komfortable und konsistente Durchführung auch größerer Änderungen wirksam unterstützt werden. Wenn Transformationen der Konstrukte zwischen den einzelnen Phasen durchzuführen sind, sollte dabei vermieden werden, Semantik zu reduzieren: Transformationen sind sonst irreversibel und - was schwerer wiegt - die spätere Rekonstruktion von Semantik ist mit Aufwand und Risiko verbunden. Daneben sollte es Möglichkeiten geben, die Auswirkungen von Änderungen, die an einem Konstrukt durchgeführt wurden, auf die entsprechenden Konstrukte anderer Transformationsstufen nachzuhalten, also referentielle Integrität zu wahren. Entwurfsfehler sollten nicht nur auf syntaktischer, sondern auch auf semantischer Ebene, durch automatisierbare Verfahren vermeidbar bzw. erkennbar sein. Hier ist beispielsweise an die Entdeckung von Redundanzen, Anomalien oder Generalisierungspotentialen zu denken.

Wiederverwendbarkeit

- Modularisierung: Die mit den Modellierungskonstrukten korrespondierenden Implementierungskomponenten (beispielsweise Datenstrukturen, Prozeduren oder Klassen) sollten wirksam vor von anderen Systemteilen ausgehenden Seiteneffekten geschützt sein.
- Domänenspezifische Modelle/Modellgerüste: Solche Gesamtmodelle sind eine erhebliche Unterstützung für die Analyse und den Entwurf und erleichtern das Auffinden anwendungsnaher Konstruke.
- Produktions- und Laufzeitumgebungen: In Informationssystemen gibt es eine Reihe allgemeiner Anforderungen, wie beispielsweise Interprozeß bzw. -programmkommunikation (auf einem Rechner und zwischen Rechnern), Verwaltungsaufgaben, wie etwa die Wahrung gewisser Integritätsformen oder die Unterstützung der Suche nach Informationen. Hier ist also an die Unterstützung durch Betriebssysteme und vor allem Datenbankmanagementsysteme zu denken.[1]

Förderung leistungsfähiger Architekturen[2]

- Verteilung: Die im Modell verwendeten Konstrukte bzw. die korrespondierenden Implementierungskomponenten sollten eine verteilte, transparente Verarbeitung unterstützen. Grundsätzlich werden die Chancen für eine transparente Verteilung mit zunehmender Semantik der bei der Kooperation zwischen den Rechnern verwendeten Konstrukte verbessert. Daneben aber hängt die Möglichkeit einer verteilten Architektur wesentlich von der Offenheit einzelner Systemteile ab, also der Verbreitung der im Modell verwendeten Konstrukte sowie der zugehörigen Produktions- und Laufzeitumgebungen. Auch hier zeichnet sich der schon mehrfach erwähnte Konflikt zwischen Semantik und Verwendungsbandbreite ab.
- Integrität: Je höher das semantische Niveau der Konstrukte eines Modells und je größer der Teil eines Informationssystems, der durch ein Modell abgedeckt ist, desto besser sind die Chancen, eine Architektur zu verwirklichen, die ein hohes Maß an Integrität verspricht.[3] Modularisierung ist mit Integrität ebenfalls positiv korreliert.

---

1. Diese Unterscheidung ist sicherlich nicht denknotwendig. So wird in der Diskussion um zukünftige Betriebssysteme wird häufig vorgeschlagen, Dateisysteme durch (objektorientierte) Datenbankmanagementsysteme zu ersetzen.
2. Hier ist einerseits an die durch den Modellierungsansatz forcierten softwaretechnischen Gestaltungsregeln zu denken, andererseits an die Qualität der bereits genannten, mit dem Modellierungsansatz mehr oder weniger eng verbundene Produktionsumgebung zu denken
3. Damit ist nicht gesagt, daß der Aufwand zur Realisierung solcher Architekturen unter Umständen geringer ist.

Organisatorische Integration

- Organisatorischer Kontext: Auch organisatorische Aspekte, die nicht unmittelbar auf das Informationssystem abgebildet werden, können für dessen Gestaltung von Bedeutung sein. Hier ist unter anderem an nicht formalisierte oder nicht formalisierbare Sachverhalte zu denken. Die anschauliche Abbildung solcher Sachverhalte fördert das Gesamtverständnis und liefert ein Medium zur Überwindung von Perspektivendifferenzen.
- Analyse/Simulation organisatorischer Alternativen: Ein effizienter Einsatz von Informationstechnik macht es unter Umständen nötig, geeignete Reorganisationsmaßnahmen durchzuführen. Ein Modell, das Bewertungskriterien und Gestaltungsregeln enthält, bietet die Chance für die Implementierung entsprechender Analyse- und Simulationsverfahren.
- Berücksichtigung von Kosten/Nutzen-Kriterien: Neben der Berücksichtigung der Funktionalität empfiehlt sich für die Beurteilung von Gestaltungsalternativen eine eingehende betriebswirtschaftliche Bewertung. Dazu sollte es möglich sein, das Modell (bzw. eine Modellebene) mit Angaben über - mehr oder weniger formalisierte - Größen wie Kosten, Beitrag zur Umsetzung von Unternehmensstrategien und dergleichen anzureichern.

Für die Auswahl eines Modellierungsansatzes in praxi ist es sicherlich wichtig, die jeweils verfügbaren Werkzeuge (beispielsweise zur Überwachung der Konsistenz eines Modells, zur Generierung von Gestaltungsalternativen oder zur Versionsverwaltung) zu berücksichtigen. Für unsere Betrachtung wird dieses Kriterium allerdings ausgespart, da es hier in erster Linie um die Qualität eines Modellierungsansatzes geht. Sie setzt den Rahmen für die Mächtigkeit zugehöriger Werkzeuge. Konkrete Implementierungen können diesen Rahmen in ganz unterschiedlicher Weise füllen.[1]

Die unter verschiedenen Gesichtspunkten gruppierten Kriterien überschneiden sich mitunter und sperren sich zum Teil gegen eine griffige Operationalisierung. Darüber hinaus gibt es Konkurrenzbeziehungen. Mit anderen Worten: Die Anwendung der Beurteilungskriterien setzt ihre angemessenen Interpretation voraus. Diese Einschränkung gilt es zu berücksichtigen, wenn im folgenden die aufgeführten Kriterien im Hinblick auf tatsächliche Formen der konzeptuellen Modellierung weiter detailliert werden.

1. Diese Einschränkung hat auch einen pragmatischen Grund: Eine Berücksichtigung von Werkzeugen würde die Betrachtung einer großen Vielfalt von CASE-Umgebungen nötig machen.

2. Evolutionsstufen der konzeptuellen Modellierung

Während Programmierung traditionell mehr oder weniger stark durch die von Digitalrechnern gesetzten Randbedingungen geprägt ist, zielt die konzeptuelle Modellierung auf eine möglichst anschauliche Repräsentation eines Anwendungsbereichs. Dadurch soll es dem Entwickler und - in Grenzen - den beteiligten Benutzern ermöglicht werden, sich auf die Erfassung der Anwendungsdomäne zu konzentrieren, die Details der Technik zu vernachlässigen. Angesichts der vielfältigen Möglichkeiten Realität zu beschreiben, verwundert es wenig, daß im Laufe der Zeit eine große Zahl von Modellierungsansätzen entstanden ist.[1] Sie wurden vor allem in drei Forschungsbereichen entwickelt: Datenbanken, Künstliche Intelligenz und Programmiersprachen.[2] Es ist nicht zuletzt ein Ziel der vorliegenden Untersuchung, aufzuzeigen, wie zukünftige Arbeiten aus diesen Bereichen synergetisch verbunden werden können. Hewitt/DeJong (1984, S. 147) sehen darin sogar ein Ziel für die Weiterentwickung der konzeptuellen Modellierung:

> "A goal of conceptual modelling is to aid in the implementation of next generation computing systems by applying and unifying results from the fields of programming languages, data bases, and artificial intelligence."

Die Vielfalt der Arbeiten in diesen Forschungsbereichen macht es unmöglich, im Rahmen der vorliegenden Arbeit auch nur alle besonders originellen Ansätze zu betrachten. Die Auswahl der zu untersuchenden Modellierungsansätze richtet sich nach ihrer Bedeutung - in der Praxis wie auch in der wissenschaftlichen Forschung - für die Unternehmensmodellierung. Wir werden zunächst einige in diesem Sinne bedeutsame Verfahren zur konzeptuellen Datenmodellierung sowie deren Erweiterung um Verfahren zur Modellierung operationaler Aspekte betrachten. Diese Modellierungsansätze bilden den Hintergrund für Informationssystem-Architekturen, die gegenwärtig von großen Anbietern propagiert werden. Anschließend gilt unser Augenmerk dem objektorientierten Softwareentwurf - er scheint besonders gut geeignet, die Anforderungen der Unternehmensmodellierungen zu erfüllen. Das Ziel dieses Kapitels ist keine umfassende und

---

1. In Olle/Hagelstein unter anderem (1988, S. 269 ff.) findet sich - unter der Überschrift "Some existing methodologies" eine Liste von 34 kommerziell vertriebenen Methoden. Balzert (1990) gibt einen Überblick von Ansätzen aus den Bereichen Datenmodellierung und Programmentwurf, wobei ein besonderes Gewicht auf die Beschreibung der zugehörigen Entwicklungsumgebungen gelegt wird. Eine ähnliche Sicht wird - allerdings mit stärkerer Betonung konzeptioneller Grundlagen - von Batini/Ceri/Navathe (1992) wahrgenommen. Brodie (1984) ergänzt die Charakterisierung kommerziell eingesetzter Datenmodelle durch die Betrachtung von Ansätzen zur Wissensmodellierung.
2. Eine fundierte Darstellung dieser drei Wurzeln der konzeptuellen Modellierung sowie ihrer Zusammenhänge findet sich in den Beiträgen in Brodie/Mylopoulos/Schmidt (1984).

detaillierte Revision der vielfältigen Formen konzeptueller Modellierung. Vielmehr ist es darauf gerichtet, wesentliche Unterschiede deutlich zu machen und brauchbare Evaluationskriterien zu erarbeiten.

2.1 Verfeinerung der Beurteilungskriterien

Ein konzeptuelles Modell ist eine Abstraktion eines Realitätsausschnitts: Nur die Aspekte der jeweiligen Domäne sollen berücksichtigt werden, die auf das zu entwerfende Informationssystem abzubilden sind. Daneben soll von einzelnen Ausprägungen der Domäne abstrahiert werden. Stattdessen sollten möglichst generelle Konzepte verwendet werden. Im oben entwickelten Bezugsrahmen kommt der Semantik eines konzeptuellen Modells herausragende Bedeutung bei. Der Zweck eines jeweils zu modellierenden Informationssystems ist die Verwaltung von Informationen. Die Semantik des Modells nimmt also tendenziell zu mit der Einschränkung zulässiger Interpretationen der verwaltbaren Informationen oder - anders formuliert: mit dem Umfang, der durch Integritätsregeln (Constraints) ausgeschlossenen, also unzulässigen Systemzuständen. Wenn wir ein konzeptuelles Modell zunächst unter strukturell-statischen Aspekten betrachten, können wir es vereinfacht wie folgt charakterisieren:

> Ein konzeptuelles Modell besteht aus Abstraktionen realweltlicher Gegenstände oder Sachverhalte und zwischen ihnen existierenden Beziehungen.[1] Für die Zustände dieser Elemente eines Modells sowie deren Veränderung können mehr oder weniger detaillierte Integritätsbedingungen angegeben werden.

Die Abstraktionen realweltlicher Gegenstände oder Sachverhalte bezeichnen wir hier in Einklang mit gängiger Terminologie als Entitäten oder Objekte. Durch Generalisierung über gemeinsame Merkmale einer Menge von Entitäten entsteht ein Entitätstyp. Die Integrität eines Systems kann dann beschrieben werden durch eine isolierte Definition von Entitätstypen bzw. Beziehungen sowie durch eine zusammenhängende Betrachtung dieser Modellierungskonzepte. Dabei kann jeweils mehr oder weniger klar zwischen statischen und operationalen Integritätsbedingungen differenziert werden:

---

1. Eine ähnliche Generalisierung findet sich in Mayr/Dittrich/Lockemann (1987, S. 498): "Generell wird nämlich (implizit) der Systembegriff der allgemeinen Systemtheorie als Gerüst zugrunde gelegt: jede Miniwelt wird beschrieben als eine Menge von Gegenständen (unserer Anschauung und unseres Denkens), den Systemelementen, zwischen denen wohldefinierte (System-) Beziehungen bestehen." Dabei mag es sein, daß auch gleichartige Beziehungen zu Beziehungstypen zusammengefaßt werden können.

Isolierte Betrachtung von Entitäten

- Angaben über zulässige Zustände der Entitäten (Instanzen) eines Entitätstyps. Dabei ist neben der isolierten Definition von Wertebereichen auch daran zu denken, daß der Wertebereich eines Attributs vom Zustand eines anderen Attributs abhängt. Um ein einfaches Beispiel zu geben: Der Wertebereich für einen Verkaufspreis kann für sich genommen - je nach Sortiment - eine bestimmte Bandbreite abdecken. Zur Vermeidung unzulässiger Verkaufspreise kann es - je nach Preispolitik - sinnvoll sein, den Wertebereich im Einzelfall durch den jeweiligen Einstandspreis nach unten zu begrenzen.
- Angaben über zulässige Zustandsänderungen dieser Instanzen. Solche Regeln sind dann von Bedeutung, wenn nicht beliebig von einem zulässigen Zustand einer Instanz in einen anderen übergegangen werden kann.

Betrachtung von Entitäten im Zusammenhang mit anderen Entitäten

- Angaben über zulässige Zustände einer Entität in Abhängigkeit von Zuständen anderer Entitäten. Beispiel: Das Gehalt einer Führungskraft sollte mindestens so hoch sein wie das seines bestbezahlten Mitarbeiters.
- Angaben über zulässige Zustandsänderungen einer Entität in Abhängigkeit von Zuständen anderer Entitäten. Beispiel: Ein in einem Versicherungsunternehmen bearbeiteter Antrag auf Schadenersatz kann zu den, je für sich genommen zulässigen, Zuständen "abgelehnt" und "gebilligt" führen. Nur eine der beiden Zustandsänderungen aber sollte nach Maßgabe der Prüfungsregeln (die auf die Zustände anderer Entitäten referieren) zulässig sein.
- Angaben über zulässige Zustandsänderungen einer Entität in Abhängigkeit von Zustandsänderungen anderer Entitäten. Es mag sein, daß der Zustand einer Entität nur im Zusammenhang mit der Veränderung des Zustands anderer Entitäten verändert werden darf. Hier ist an die Modellierung von Transaktionen oder Vorgängen zu denken.
- Angaben darüber, wodurch Zustandsänderungen ausgelöst werden.

In allen Fällen kann "zulässig" durch das komplementäre "unzulässig" ersetzt werden, je nachdem, wie die jeweilige Integritätsbedingung formuliert ist.[1] Die skizzierten Integritätsbedingungen werden im folgenden durch die Verwendung

1. Mit zunehmender Semantik eines konzeptuellen Modells werden die Grenzen zur Programmierung fließender. Es gibt vor allem zwei Unterschiede: Ein konzeptuelles Modell ist tendenziell darauf gerichtet, Interpretationsspielräume festzulegen, während durch die in der Programmierung festgelegten Operationen eine bestimmte Interpretation ausgewählt wird. Schmidt (1987, S. 3) spricht in diesem Zusammenhang vom Hinzufügen eines "Interpretationsprozesses". Vor allem aber sind in einem Programm die Randbedingungen heutiger Digitalrechner-Architekturen tendenziell stärker zu berücksichtigen.

von drei wesentlichen Blickrichtungen differenziert: *Strukturelle* Aspekte beschreiben zulässige Strukturen bzw. Zustände, *funktionale* Aspekte konzentrieren sich auf Funktionen des Systems und *dynamische* Aspekte dienen der Beschreibung des Systemverhaltens im Zeitverlauf. Auf der funktionalen und der dynamischen Ebene können - mehr oder weniger detailliert - Kontrollstrukturen abgebildet werden. Sie beschreiben die Bedingungen, unter denen bestimmte Funktionen ausgeführt werden bzw. bestimmte Zustandsänderungen erfolgen. Um zu einem differenzierteren Beurteilungsraster zu gelangen, ist also näher zu charakterisieren, in welcher Weise die Semantik von Entitäten - isoliert und im Zusammenhang betrachtet - dargestellt werden kann. Im folgenden Bezugsrahmen sind die Kriterien näher erläutert, die später zur Beurteilung einzelner Ansätze verwendet werden.

Strukturelle Aspekte			Erläuterung
Attribute	maschinen-orientiert	definierbar als Intervall	innerhalb vorgegebener Mengen (z.B. Festpunktzahlen von 1 bis 100)
		durch funktionale Spezifikation	wiederum bezogen auf vorgegebene Mengen (z.B. gerade Festpunktzahlen)
	anwendungs-orientiert	durch Enumeration	z.B. Farben, Namen
		Entitätstyp	Der Typ eines Attributs kann durch einen frei definierten Entitätstyp (-klasse) festgelegt werden.
	Kardinalität	minimal = 0	Einem Attribut muß nicht unbedingt ein Wert zugewiesen werden
		maximal > 1	Einem Attribut kann mehr als ein Wert aus der jeweils vereinbarten Wertemenge zugewiesen werden.
	Zugriffsrecht		Der Zugriff auf ein Attribut kann differenziert eingeschränkt werden.
	Beziehungen		Hier ist an Integritätsbedingungen zu denken, die die zulässigen Zustände eines Attributs vom Zustand anderer Attribute abhängig machen. Beispielsweise könnte so festgelegt werden, daß der Verkaufspreis eines Produkts mindestens so groß sein muß wie der Einkaufspreis.
Identität einer Entität vom Zustand determiniert			Eine Entität wird allein durch ihren Zustand identifiziert. Daraus folgt als Integritätsbedingung, daß niemals zwei Entitäten eines Typs in gleichem Zustand koexistieren dürfen. Kontrast: Eine Entität hat eine vom Zustand unabhängige Identität.

Strukturelle Aspekte				Erläuterung
Historie	generell			Im Modell kann spezifiziert werden, ob die Protokollierung von Veränderungen des Zustands einer Entität erwünscht ist.
	partiell			Die Protokollierung kann auf die Veränderung einzelner Attribute beschränkt werden.
Beziehungen zwischen Entitäten	Kardinalität	einfach		Es kann für jeden der assoziierten Entitätstypen jeweils ein Wert angegeben werden.
		min-max	eingeschränkt	Es kann jeweils eine minimale und eine maximale Kardinalität definiert werden - jeweils aus einer eingeschränkten Menge (wie etwa {0, 1} oder {1, n }).
			frei	Minimale und maximale Kardinalität können differenzierter festgelegt werden.
	Richtung	Rolle		Die unterschiedlichen Bedeutungen einzelner Entitätstypen innerhalb einer Beziehung können durch Rollen ausgezeichnet werden.
		inverse Beziehung		Eine Beziehung ist gerichtet. Die mit den unterschiedlichen Richtungen einhergehenden Bedeutungen können durch Angabe einer inversen Bezeichnung ausgedrückt werden (z.B. "benutzt" - "wird benutzt").
	Referentielle Integrität			Die existentielle Abhängigkeit der Entitäten eines Typs von denen eines anderen kann ausgedrückt werden (alternativ zu Kard.).
	Aggregation			Eine Assoziation zwischen mehreren Entitäten, die auf einer höheren Abstraktionsstufe wiederum wie ein Entität behandelt werden kann.
	sonstige			Weitere Beziehungen mit einer speziellen vorgegebenen Semantik - wie beispielsweise "ist Version von".
Generalisierung	einfach			Es kann über ein gemeinsames Muster generalisiert werden. Damit wird Einfachvererbung möglich.
	mehrfach			Eine Generalisierung über mehrere Muster ist möglich - damit auch Mehrfachvererbung.

Strukturelle Aspekte

		Erläuterung
Views		
Entität	Spezifikation von Views	Durch die Auswahl von Attributen bzw. Operationen können Sichten auf eine Entität definiert werden.
	differenzierte Zugriffsrechte	Dito, allerdings mit einer differenzierten Festlegung von Zugriffsrechten.
	Benutzerschnittstelle	Der Entität kann eine prototypische Benutzerschnittstelle zugeordnet werden.
auf Kollektionen	Spezifikation von Views	Es kann ein View auf mehrere Entitäten, der für einen Arbeitskontext bedeutsam ist, zusammengestellt werden.
	Benutzerschnittstelle	Für einen solchen Arbeitskontext kann eine prototypische Benutzerschnittstelle modelliert werden.

Dynamische/funktionale Aspekte

Operationen/Funktionen		
Schnittstellenspezifikation		Die Eingabe- und Ausgabedaten einer Operation können unter Rückgriff auf das strukturelle/statische Modell spezifiziert werden.
Preconditions		Es kann eine Bedingung angegeben werden, unter der allein die Operation ausgeführt werden darf.
Postconditions		Es kann eine Bedingung angegeben werden, die beim Terminieren der Operation erfüllt sein muß.
Trigger		Es können Ereignisse und daraufhin auszuführende Aktionen definiert werden.
Ausnahmebehandlung		Die Methode sieht die Beschreibung von Ausnahmen und den zu ihrer Behandlung zu ergreifenden Maßnahmen vor.
Systemverhalten		
Zeitliche Aspekte/ Kontrollstrukturen	für Entität	Es können Bedingungen für die zulässigen Zustandsänderungen der Entitäten eines Typs angegeben werden.
	für Kollektionen	Es können Bedingungen für die zulässigen Zustandsänderungen einer Kollektion interagierender Entitäten verschiedener Typen angeben werden.

Sonstige Aspekte		Erläuterung
Modularisierung	Kohärenz	Strukturelle und funktionale/dynamische Merkmale einer Entität werden zusammen modelliert.
	Assoziierung	Zwischen zusammengehörigen strukturellen und dynamischen Merkmalen können Referenzen etabliert werden. Das setzt voraus, daß beide Aspekte einander mit Hilfe eindeutiger Identifikatoren zugeordnet werden können.
Produktionsumgebung		
Datenbankmanagementsystem	DDL	Das Modell kann unmittelbar oder durch (teil-) formalisierbare Transformation mit Hilfe der Data Definition Language eines DBMS dargestellt werden.
	DML	Es ist für das Modell eine geeignete Data Manipulation Language vorhanden und implementiert.
	Standards	DDL und DML sind weitgehend standardisiert.
	Kontrolle über Operationen	Funktionen/Operationen können vom DBMS verwaltet werden.
	Werkzeugunterstützung	Es ist eine Reihe ausgereifter Werkzeuge, die auf dem Modell aufsetzen, verfügbar.

2.2 Konzeptuelle Datenmodellierung

Die besondere Bedeutung der konzeptuellen Datenmodellierung wird mitunter durch den Hinweis darauf begründet, daß in Informationssystemen die Verwaltung von Daten im Vordergrund steht und darüber hinaus die Struktur von Daten im Zeitverlauf einer geringeren Änderungswahrscheinlichkeit unterliegt als die von Funktionen. Für die herausragende Stellung der Datenmodellierung gibt es sicherlich noch einen anderen Grund: Im Hinblick auf die praktische Relevanz kann ein Modellierungsansatz nicht ohne die ihn ergänzende Technologie betrachtet werden. Die Entwicklung der Datenmodellierung und der Datenbanktechnologie vollzog sich in einem Wechselspiel, in dem aber die Technologie die wichtigste Rolle besetzte. Gerade die in der Praxis vorherrschenden Formen der

Datenmodellierung haben ihre Bedeutung also wesentlich dem Erfolg der korrespondierenden Datenbanktechnologie zu verdanken.[1] Für eine allgemeine Charakterisierung von Datenmodellierung bzw. Datenmodellen sind an dieser Stelle die beiden folgenden Definitionen hinreichend:

> “Mit “Datenmodellierung” oder genauer: “Modellierung durch Daten” bezeichnen wir ... die Erstellung von Beschreibungen direkt oder über einfache Zwischenstufen durch Kombination von Daten aus vorgegebenen Datenmengen.” (Schmidt 1987, S. 5)

> "Ein Datenmodell ist ein Gestaltungsrahmen (Metamodell), der die verfügbaren Arten von Objekt- und Beziehungstypen, ihre Semantik, sowie syntaktische Regeln für ihre Kombinierbarkeit angibt." (Sinz 1987, S. 77)

Die folgende Betrachtung des Entity-Relationship-Modells und seiner Erweiterungen ist ein Tribut an die herausragende Bedeutung dieses Ansatzes in der Praxis. Ähnliches gilt für die daran anknüpfende Beschreibung des relationalen Modells: Auch wenn es nicht unbedingt notwendig ist, so werden doch unter den Rahmenbedingungen der heute vorherrschenden Technik Entity-Relationship-Modelle zumeist in relationale Modelle transformiert.

2.2.1 Das Entity-Relationship Modell

Das Entity-Relationship Modell (ERM) wurde von Chen (1976) mit der Zielsetzung eingeführt, ein Modellierungsverfahren zu schaffen, das von den in Datenbankmanagement-Systemen vorrangig implementierten Datenmodellen[2] zu abstrahieren erlaubt. Es ist gegenwärtig - in Zusammenhang mit der Verbreitung relationaler Datenbankmanagementsysteme - das wohl bedeutendste Datenmodell. Das ERM existiert in einer Reihe unterschiedlicher Versionen. Die folgende Darstellung ist an Beschreibungen des Modells in neueren Quellen orientiert.

2.2.1.1 Darstellung

Datenmodellierung dient einerseits der systematischen datenorientierten Analyse eines Realitätsausschnitts, andererseits vor allem dem Entwurf eines Datenmodells, das eine leistungsfähige Verwaltung von Daten in einem Datenbankmanagement-System verspricht. Im ersten Fall geht es also um die Erfassung der jeweils relevanten Konzepte, im zweiten Fall darum, sie im Hinblick auf die Datenverwaltung zu gestalten. Chen spricht im ersten Fall von "level 1" und nennt das eigentliche konzeptuelle Datenmodell "level 2" bzw. "information structure".

Auf der ersten Ebene unterscheidet Chen “entities” und “relationships”. Eine

1. Damit soll nicht gesagt sein, daß der Entwurf eines Datenbankmanagement-Systems nicht die Entwicklung eines brauchbaren Datenmodells impliziert.
2. Chen (1976, S. 9) nennt explizit: "relational model", "network model" und "entity set model".

Entität bezeichnet zunächst einen Gegenstand bzw. Sachverhalt[1] in dem jeweils betrachteten Realitätsausschnitt. Mit Hilfe einer Relationship kann ausgedrückt werden, daß zwischen Entitäten eine Beziehung besteht. Zum Zweck der Modellierung wird auf die Entitäten eine *Datenabstraktion* angewandt: Von allen Eigenschaften einer Entität, die sich nicht mit Hilfe von Daten beschreiben lassen, wird abgesehen.[2] Zudem wird von der konkreten Ausprägung einer Entität abstrahiert. Stattdessen wird versucht, mit Hilfe der Datenabstraktion Eigenschaften zu definieren, die für eine Menge von Entitäten charakteristisch sind. Auf diese Weise entsteht die Ebene 2. Sie beinhaltet Entitätsmengen (“entity sets”) bzw. Entitätstypen. Die Semantik einer Entitätsmenge wird beschrieben durch eine bestimmte Zahl von Attributen. Attribute wiederum sind gekennzeichnet durch jeweils eine Wertemenge. Wertemengen können durch Rückgriff auf maschinennahe Repräsentationen wie auch durch Aufzählung von anwendungsorientierten Konstrukten charakterisiert werden. Mehrere Attribute können zu einer Attributgruppe zusammengefaßt werden - also beispielsweise "Name", "Straße" etc. zur Gruppe "Adresse". Die Kardinalität der Attribute wird von Chen nicht explizit eingeschränkt: Ein Attribut kann durch keinen, einen oder mehrere Werte gekennzeichnet werden.

Die Entitäten zweier oder mehrerer Entitätstypen können mit Hilfe von Beziehungen ("relationships") verbunden werden. Dabei können die beteiligten Entitätstypen mit Hilfe einer Rolle ausgezeichnet werden. Die durch eine Beziehung verknüpften Entitäten müssen nicht unterschiedlichen Entitätstypen angehören. So können beispielsweise durch die Beziehung "verheiratet mit" zwei Entitäten des Typs "Person" assoziiert werden - differenziert durch die Rollen "Ehemann" und "Ehefrau". Man spricht dabei auch von rekursiver Beziehung. Die Rollen können jeweils mit einer Kardinalität gekennzeichnet werden. Eine Kardinalität kann, je nach Version des ERM, entweder durch einen einzigen Wert beschrieben werden, oder durch zwei Werte in min/max-Notation. Für den ersten Fall stehen zwei Wertausprägungen zur Verfügung: eine 1 steht für "genau eine Entität", n bzw. m für "null bis viele". Damit lassen sich - für binäre Beziehungen - drei Kardinalitätspaare formulieren: 1:1, 1:m und n:m. Die min/max-Notation erlaubt eine semantisch angereicherte Beschreibung: Für jede Rolle kann sowohl die minimal nötige wie auch die maximal zulässige Anzahl angegeben werden. Auf diese Weise erlauben Kardinalitäten, die existentielle Abhängigkeit einer Entitäts eines Typs von der eines anderen Typs auszudrücken. Eine Beziehung wird repräsentiert durch ein Tupel der jeweils verknüpften Entitäten. Die Menge der durch eine bestimmte Beziehung zwischen den Entitäten der assoziierten Enti-

1. Chen (1976, S. 10): “An *entity* is a ’thing’ which can be distinctly identified.”
2. Andere Bezeichnungen für durch Datenabstraktion gebildete Entitäten sind Datenobjekt oder Objekt.

tätstypen darstellbaren Beziehungen beschreibt ein "relationship set". Relationstypen können eigene Attribute zugewiesen werden, also z.B. für die Beziehung zwischen den Entitätstypen "Projekt" und "Projektmitarbeiter" das Attribut "Arbeitsstunden im Projekt".

Für die Definition von Aggregationen stehen im ERM keine speziellen Konstruktoren zur Verfügung. Ein Aggregat kann also auch nicht mit einem eigenständigen Namen bezeichnet werden. Der Rückgriff auf Beziehungen zusammen mit Kardinalitäten erlaubt allerdings die Rekonstruktion wichtiger mit einer Aggregation verbundener Integritätsbedingungen.

Im Hinblick auf die Verwaltung von Entitäten verwendet Chen das Konzept der Relation. Eine Relation beschreibt eine umkehrbar eindeutige Abbildung der in einem Informationssystem erfaßten Entitäten auf die zulässigen Entitäten eines Entitätstyps. Die Relation beschreibt also ebenfalls eine Menge. Auf diese Weise wird implizit die Integritätsbedingung festgelegt, daß eine bestimmte Ausprägung einer Entität nicht mehr als einmal vorkommen darf. Eine Gruppe von Attributen einer Entität kann als Primärschlüssel ausgezeichnet werden. Falls mit Hilfe einer brauchbaren Kombination von Attributen die nötige Eindeutigkeit nicht hergestellt werden kann, können explizit identifizierende Schlüssel als künstliche Attribute eingeführt werden.

Eine Menge von Entitäten eines Typs bildet eine "entity relation", eine Menge von Beziehungen gleichen Typs eine "relationship relation". Im Hinblick auf Maßnahmen zur Wahrung (referentieller) Integrität unterscheidet Chen "regular" und "weak" Relations[1]. Eine reguläre Relation umfaßt Entitäten, die eine unabhängige Existenz haben, also allein durch die ihnen eigenen Attributwerte identifizierbar sind. Demgegenüber erfordert die Identifikation der Tupel einer schwachen Relation den Verweis auf ein Tupel einer anderen Relation, das durch einen zusätzlich eingeführten Schlüssel identifiziert wird. Die Extraktion von Attributen, die nicht unmittelbar zu einer Relation gehören, ist vergleichbar mit der im relationalen Modell vorgesehenen Transformation in die zweite Normalform.

Die wesentlichen Konzepte des ERM und ihre grafische Visualisierung in einem ERM-Diagramm sind in Abbildung 8 im Überblick und in Abbildung 9 im Beispiel dargestellt. Die in einem ERM verwendeten Bezeichner sollten in Anlehnung an die jeweils abgebildeten realweltlichen Sachverhalte gewählt werden. Während sich für Entitätstypen dazu Substantive anbieten, werden Beziehungstypen teilweise durch eine Kombination der verknüpften Entitätstypen bezeichnet, teilweise werden dazu mitunter Prädikate verwendet.

1. Dabei wird dann noch weiter zwischen Beziehungsrelationen und Entitätsrelationen unterschieden.

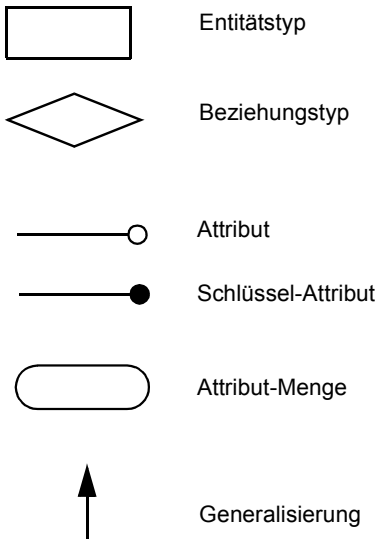


Abb. 8: Die wesentlichen Konzepte des ERM und ihre Symbolisierung (in Anlehnung an Batini/Ceri/Navathe (1992, S. 44) und Chen (1976)).

Eine an Prädikaten orientierte Bezeichnung kann die Lesbarkeit eines Diagramms fördern. Es ist dabei allerdings einschränkend zu berücksichtigen, daß die Beziehungen nicht gerichtet sind - die Richtung ist aber für die Interpretation eines Prädikats unter Umständen unerläßlich (vgl. dazu die Beziehungen in Abbildung 9).

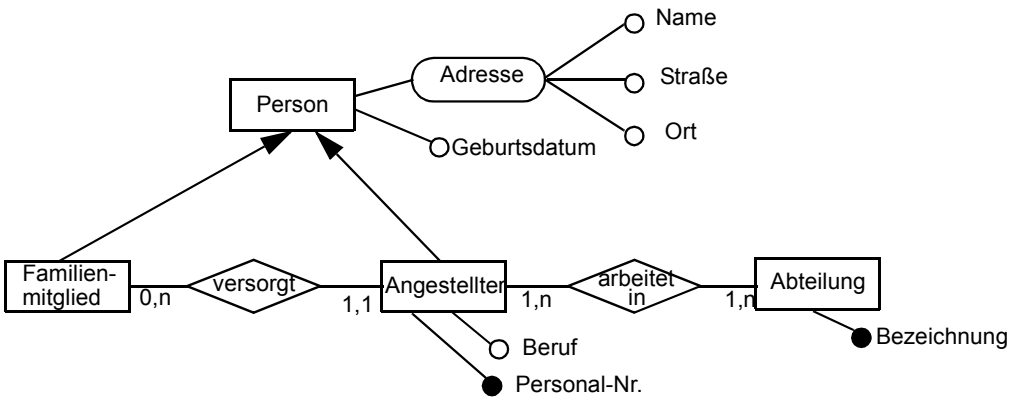


Abb. 9: Beispiel für ein ERM-Diagramm. Um die Übersichtlichkeit nicht zu gefährden, wird zumeist auf die Darstellung der Attribute verzichtet.

Um die in einem ERM-Diagramm dargestellte Semantik maschinell interpretierbar zu machen, hat Chen eine Data Description Language (DDL) eingeführt. Ein mit Hilfe dieser Sprache beschriebenes konzeptuelles Modell könnte unmittelbar als die Definition eines Datenbankschemas dienen. Zudem skizziert Chen eine an der Mengenalgebra orientierte Data Manipulation Language (DML), die in einem entsprechenden Datenbankmanagementsystem zu Formulierung von Queries eingesetzt werden könnte. Sowohl die DDL als auch die DML werden jedoch kaum unmittelbar genutzt: "Für das ER-Modell sind (fast) keine DBMS verfügbar." (Sinz 1987, S. 100) Im praktischen Einsatz werden ERM deshalb zumeist in das Schema eines relationalen Datenbankmanagementsystems transformiert.

2.2.1.2 Die Transformation in das relationale Modell

Für unsere Untersuchung ist es wesentlich, festzustellen, welcher Verlust an Semantik mit der Überführung eines ERM in ein relationales Modell verbunden ist. Angesichts der vielfachen Beschreibungen des relationalen Modells in der Literatur und seiner übersichtlichen Struktur, können wir uns auf eine kurze Charakterisierung[1] der wichtigsten Merkmale beschränken:

- Ein relationales Modell bzw. Schema besteht aus einer Menge eindeutig benannter Relationstypen, die sich in erster Normalform[2] befinden.
- Mit der Definition eines Relationstyps ist implizit die Deklaration genau einer Relationsvariablen verbunden, die durch eine Relation instanziiert werden kann. Eine Relation ist eine Menge von Tupeln, die aus einer festen Anzahl von Werten eines jeweils vorgegeben Attributtyps bestehen. Die Reihenfolge der Werte in einem Tupel ist bedeutungslos.
- Für jeden Relationstyp wird ein Primärschlüssel definiert. Dabei kann es sich - um zwei Extreme zu nennen - um einen eigens eingeführten Schlüssel oder auch um eine Kombination aller Attributwerte handeln.
- Die Attribute des Primärschlüssels müssen mit einem Wert des jeweils zugeordneten Typs instanziert sein - Nil-Werte sind also nicht erlaubt.
- Referentielle Integrität: Sobald eine Entität nur dann korrekt interpretiert werden kann, wenn eine oder mehrere andere Entitäten existieren, ist zu fordern, daß während der Lebenszeit der abhängigen Entität keine der Entitäten, auf die verwiesen wird, gelöscht werden darf. Die damit verbundene Integritätsbedingung wird gemeinhin - in Anlehnung an eine Forderung Codds - als referenti-

1. In Anlehnung an die Darstellungen in Sinz (1987, S. 101 ff.) und Schmidt (1987, S. 30 ff.)
2. Die erste Normalform ist dadurch gekennzeichnet, daß jeder Attributwert elementar sein muß.

elle Integrität bezeichnet[1].

Die Transformation eines ERM in ein relationales Schema vollzieht sich in einer Reihe von Schritten, auf die hier nicht eingegangen werden muß.[2] Sie ist mit einem Verlust an Anschaulichkeit verbunden, da es für das relationale Modell üblicherweise keine grafischen Darstellungen gibt. Dieser Umstand fällt allerdings erst bei Wartungsarbeiten ins Gewicht. Die Semantik des relationalen Modells ist gegenüber dem ERM in einigen Punkten eingeschränkt. So kann keine Generalisierung[3] dargestellt werden. Außerdem können Beziehungen nicht mit Kardinalitäten ausgezeichnet werden - die schon im ERM nur in Ansätzen mögliche Beschreibung von Aggregaten wird so noch weiter aufgeweicht.

Die Transformation in das relationale Modell ist also mit einem erheblichen Verlust an Semantik verbunden. Tsichritzis (1989, S. 501) betont, daß dies durchaus beabsichtigt war: “The whole idea was to collapse down the semantics as much as possible to almost no semantics at all.” Der mit dem niedrigen semantischen Niveau einhergehende Vorteil liegt auf der Hand: Dadurch daß alle Informationen in einer relationalen Datenbank in Relationen abliegen und alle Operationen der Relationenalgebra wiederum Relationen produzieren, wird die Entwicklung konsistenter Verwaltungsverfahren[4] sowie einfacher DDL und DML erheblich erleichtert. Zudem steht mit der schrittweisen Normalisierung ein formales Verfahren zur Vermeidung von Anomalien bereit.[5] In praxi äußern sich diese Vorteile unter anderem in einer mit SQL weitgehend standardisierten kombinierten DDL und DML. Zudem wird die Definition von Teilsichten auf das Modell (sogenannte Views) dadurch erleichtert, daß jeder View grundsätzlich in Form von Relationen definiert wird.

---

1. Es handelt sich hier genau genommen um eine spezielle Form der referentiellen Integrität. Schließlich gibt es in Informationssystemen gibt auch andere Referenzen von existentieller Bedeutung als solche auf Entitäten.
2. Eine ausführliche Darstellung der Transformation findet sich in Batini/Ceri/Navathe (1992, S. 313 ff.). Dabei ist zu beachten, daß diese Transformation nicht vollständig automatisierbar ist. Eine modifizierte Version des ERM, die eine algorithmische Beschreibung der Transformation erlaubt, wird in Wong/Katz (1980) beschrieben.
3. Es gibt allenfalls die wenig befriedigende Möglichkeit, Generalisierungen mit Hilfe der Definition von Bedingungen referentieller Integrität zu rekonstruieren. So könnte jedes Tupel einer Relation mit einem Schlüssel versehen werden, der auf das entsprechende Tupel in einer übergeordneten Relation verweist.
4. Wobei sich Konsistenz natürlich auf die Verwaltung der eingeschränkten Semantik der verfügbaren Integritätsbedingungen beschränkt.
5. Die Automatisierung der Normalisierung setzt allerdings voraus, daß eine einheitliche Namensgebung erfolgt, da sonst die zwischen einzelnen Relationen bestehenden Zusammenhänge nicht erkannt werden können.

2.2.1.3 Beurteilung

Bevor dazu das oben entfaltete Beurteilungsraster angewendet wird, soll zunächst die Einbettung des ERM in fortschrittliche werkzeuggestützte Entwicklungsprozesse skizziert werden. Anschließend werden wesentliche Merkmale der mit einem solchen Ansatz verknüpften Architektur von Informationssystemen dargestellt.

Gleichwohl die Trennung in Analyse- und Entwurfsphase in einem durch zyklische Rückkopplungen gekennzeichneten Entwicklungsprozeß nicht eindeutig durchzuführen ist, ist das ERM tendenziell eher der Entwurfsphase zuzurechnen. Am Markt ist eine Vielzahl von Werkzeugen verfügbar, die die Erstellung und weitere Bearbeitung von ERM unterstützen. Typischerweise kann der Entwickler dazu mit Hilfe eines dedizierten Grafik-Editors ein ERM-Diagramm erstellen, um dann auf einer weiteren Detaillierungsebene die Struktur der Entitätstypen zu beschreiben. Dabei werden häufig Versionen des ERM verwendet, die eine automatische Transformation in ein relationales Schema erlauben.[1] Je nach Werkzeug wird dabei jeweils eine mehr oder weniger große Zahl von SQL-Dialekten unterstützt. Unter den in Fußnote 2 skizzierten Voraussetzungen können die generierten Relationstypen normalisiert werden. Durch die damit verbundene Reduktion unkontrollierter Redundanz werden die Chancen für Systemintegrität erheblich verbessert. Zur Unterstützung der Anwendungsentwicklung können auf das relationale Schema externe Sichten (Views) definiert werden. Die Definition von Views wiederum kann dazu genutzt werden, Schnittstellen bzw. Schnittstellengerüste in den für die Anwendungsprogrammierung erstellten Programmiersprachen zu generieren. Dazu zählen vor allem die den externen Sichten auf das Schema entsprechenden Variablendeklarationen, aber auch Funktionsaufrufe für wichtige Datenbankzugriffe.[2] Die externen Views können - sofern sie Benutzersichten widerspiegeln - zudem als Basis für den Einsatz von Werkzeugen zur Gestaltung von Benutzerschnittstellen dienen. Die Entwicklung von Anwendungen sollte dann auf der konzeptuellen Ebene am ERM orientiert sei: Einerseits, um gegebenenfalls das ERM zu modifizieren, andererseits, um bereits vorhandene Entitäten und Beziehungen zu nutzen. Während der Implementierung kann dann auf die generierten Code-Fragmente zugegriffen werden.

Für eine differenzierte Bewertung des ERM genügt also ein Ausschnitt des in III.2.1 erläuterten Bezugsrahmens: Funktionale und dynamische Aspekte bleiben im ERM weitgehend unberücksichtigt.[3] Für den Vergleich mit der nachfolgenden

1. vgl. dazu die Werkzeugübersicht in Batini/Ceri/Navathe (1992, S. 439 ff.)
2. Dazu wird häufig sogenanntes "embedded" SQL verwendet. Die Integration mit der jeweiligen Programmiersprache erfolgt dann mittels Variablen, die mit Hilfe spezieller Datentypen deklariert werden, und sowohl in SQL als auch in umgebenden Programmiersprache genutzt werden.

Entwicklungsstufe wird dabei von einem relationalen Schema ausgegangen, weil diese Transformation heute die bei weitem wichtigste ist.[1]

Zusammenfassend läßt sich feststellen: Der wesentliche Vorteil des ERM ist in seiner Kohärenz mit einem weitgehend standardisierten Datenmodell und der darauf aufbauenden ausgereiften Datenbanktechnologie zu sehen. Dem stehen vor allem zwei Nachteile gegenüber: Durch die Beschränkung auf bestimmte Datenstrukturen bleiben weite Teile des Informationssystems in einem ERM und der darauf aufbauenden Datenbank unberücksichtigt. Hier ist beispielsweise an Dokumente, Grafiken, Bilder und dergleichen zu denken. Damit wird eine Integration solcher Daten mit den im ERM repräsentierten Daten allenfalls auf einem sehr niedrigen semantischen Niveau möglich.[2] Ein weiterer Nachteil ist die Vernachlässigung dynamischer bzw. funktionaler Aspekte. Sie verhindert eine Integration der modellierten Daten auf einem anwendungsnäheren Niveau und verschlechtert gleichzeitig die Chancen für Wiederverwendbarkeit und Integrität: Die Interpretation der Daten wird tendenziell isoliert bei der Entwicklung einzelner Anwendung definiert - mit einem entsprechenden Risiko für Redundanzen und Inkonsistenzen. Eine Reihe von Erweiterungen des ERM[3] wie auch des relationalen Datenmodells zielen auf die auf die Überwindung der einen oder anderen Schwäche, bleiben aber ebenfalls auf die Modellierung statischer Aspekte beschränkt. Jenseits der konzeptuellen Ebene wird allerdings in fortschrittlichen relationalen DBMS der Bedeutung dynamischer Aspekte Rechnung getragen. So können in manchen Systemen sogenannte "stored procedures" zur Überwachung von Integritätsbedingungen eingesetzt werden - in einer für die beteiligten Anwendungen transparenten Weise. Solche Prozeduren können zum Teil aus einem ERM generiert werden - beispielsweise Prozeduren, die die Kardinalität von Beziehungen überwachen.

---

3. Zulässige Operationen sind lediglich implizit mit den darstellbaren Wertemengen (bzw. Datentypen) für Attribute definiert.

1. Vgl. dazu die zusammenfassende Beurteilung in III.2.4. Dabei ist der erste Teil (strukturelle Aspekte) allein auf das ERM bezogen.

2. So gibt es in vielen relationalen DBMS die Möglichkeit, sogenannte "Binary Large Objects" abzulegen. Dabei handelt es sich um Felder einer Relation, die variabel große binäre Dateien (genauer: eine Referenz darauf) aufnehmen können. Ähnlich wie in gegenwärtigen Dateiverwaltungssystemen ist die Semantik eines solche Objekts im Schema nicht beschrieben. Es gibt lediglich die Möglichkeit, mit Hilfe von Annotationen für Interpretationshilfen zu sorgen (beispielsweise, indem Angaben über die zugehörige Anwendung gemacht werden). Einschränkend ist weiter zu berücksichtigen, daß gängige Büro-Anwendungen wie etwa Desktop Publishing Systeme in der Regel keine Schnittstelle zu einem RDBMS aufweisen.

3. So reichern DosSantos/Neuhold/Furtado (1980) das ERM um ein Konzept zur Darstellung von Aggregationen an. In dem von Klopprogge (1981) modifizierten ERM ist die Abbildung zeitlicher Aspekte möglich. Das von Sinz (1987) vorgeschlagene "Strukturierte ERM" erlaubt unter anderem die Modellierung von Generalisierungsbeziehungen.

2.3 Erweiterung um funktionale und dynamische Aspekte

Auch wenn es gute Gründe für die Betonung einer struktur- bzw. datenorientierten Sicht gibt (vgl. III.2.2), kann doch nicht übersehen werden, daß die Beschränkung auf eine solche Sicht für den Entwurf eines Informationssystems nicht hinreichend ist. Dies gilt in mehrfacher Hinsicht. So ist während der Analyse ein Fokussierung auf Strukturen nicht immer angemessen. Viele Fachleute einer Domäne - und dies dürfte für Unternehmen ganz besonders gelten - neigen zu einer funktions- bzw. ablauforientierten Konzeptualisierung. Funktionale und dynamische Aspekte sind für das Verständnis des Systems und die Implementierung ohnehin unerläßlich. Schließlich ist zu berücksichtigen, daß die Ergänzung einer unternehmensweiten Modellierung von Daten um funktionale bzw. dynamische Sachverhalte bessere Systeme verspricht: Die unternehmensweite Modellierung dieser Sachverhalte erleichtert die Reduktion funktionaler Redundanz und die sichere Verwaltung von Funktionskomponenten und funktionaler bzw. dynamischer Integritätsbedingungen außerhalb der darauf aufbauenden Anwendungen. Die Modellierung sowohl funktionaler als auch dynamischer Aspekte hat eine lange Tradition - allerdings weniger im Zusammenhang mit dem Entwurf unternehmensweiter konzeptueller Modelle als vielmehr für den Programmentwurf. Um einige bekannte Techniken dieser Art zu nennen: Datenflußpläne, Jackson Diagramme, Structured Analysis and Design Technique (SADT), Entscheidungstabellen, Programmablaufpläne, Struktogramme, State-Transition-Diagramme, Petri-Netze. Erst in neuerer Zeit werden Techniken zur Modellierung funktionaler und dynamischer Aspekte zusammen mit der Datenmodellierung zum Entwurf von Unternehmensmodellen eingesetzt. Diese Ausdifferenzierung der Modellierung geht einher mit einer wachsenen Zahl entsprechender Werkzeuge.[1] Zusätzliches Gewicht erhalten diese Ansätze dadurch, daß gegenwärtig von großen Anbietern Architekturen propagiert werden, die auf diesen Werkzeugen aufsetzen. Hier ist vor allem an das wesentlich von IBM geprägte Schlagwort "Repository" zu denken.

Die angedeutete Vielfalt dedizierter Modellierungsansätze macht eine Auswahl unumgänglich. Wir können uns dabei auf einige in der einschlägigen Literatur im Vordergrund stehenden Ansätze beschränken. Sie gestatten es, die wesentlichen Merkmale der Modellierung funktionaler und dynamischer Aspekte zu verdeutlichen. Anschließend ist ihre Integration mit Datenmodellen auf der konzeptuellen Ebene zu untersuchen.

1. Ausführliche Beschreibungen ausgewählter Werkzeuge finden sich in den Beiträgen in Balzert (1991) sowie in Batini/Ceri/Navathe (1992).

2.3.1 Datenflußpläne

Die Erfassung der in einem Unternehmen bzw. in dem zugehörigen Informationssystem auszuführenden Funktionen wird häufig als wichtiger Einstieg in die Problemanalyse betrachtet.[1] So ist die Frage nach den zu erfüllenden Funktionen für viele Beteiligte mitunter von größerem heuristischen Wert als die Frage nach Gegenständen bzw. Sachverhalten. Die Analyse einer Funktion liefert Erkenntnisse über den jeweiligen Dateneingang und Datenausgang. So ergeben sich Hinweise für die Strukturierung von Daten. Umgekehrt liefert die Betrachtung von Gegenständen bzw. Sachverhalten Hinweise für die Modifikation von Funktionsschnittstellen. Funktionsanalyse bzw. -modellierung und Datenmodellierung werden denn auch in der Regel als komplementäre Aktivitäten betrachtet.[2] Datenflußpläne bzw. Datenflußdiagramme stellen die wichtigste Technik zur Analyse und konzeptuellen Modellierung von Funktionen dar.

2.3.1.1 Darstellung

Ähnlich wie bei der konzeptuellen Datenmodellierung ist das Augenmerk beim Entwurf von Datenflußplänen auf anwendungsnahe Konzepte gerichtet. Es sollte zunächst weniger an die zu implementierenden Funktionen eines Informationssystems gedacht werden als vielmehr an die Funktionen eines Unternehmens:

> "The term functional analysis indicates the modeling of working activities within an enterprise; a function is simply a portion of the enterprise. Functional analysis concentrates on understanding how information is used by each function and how it is exchanged among functions." (Batini/Ceri/Navathe 1992, S. 195)

Vor dem Hintergrund solcher Anforderungen erscheint die in DIN 66001 definierte Technik zur Erstellung von Datenflußplänen wenig geeignet, da sie deutlich an den Gegebenheiten von DV-Systemen orientiert ist. So gibt es beispielsweise unterschiedliche Symbole für Speicher mit sequentiellem und solchen mit direktem Zugriff. Demgegenüber wird in neueren Darstellungen eine abstraktere Symbolik favorisiert. Auch wenn in der Analyse zunächst realweltliche Aufgaben betrachtet werden, wird im Zuge der Modellierung auf solche Funktionen eingeschränkt, die im Informationssystem zu implementieren sind. Man spricht in diesem Zusammenhang von *Funktionsabstraktion*. Die Modellierung basiert dabei wesentlich auf zwei Konstrukten: Funktionen[3] und Datenflüssen. Eine

---

1. Dieser Umstand wird nicht zuletzt dadurch betont, daß häufig allein von Funktionsanalyse und nicht von Funktionsmodellierung die Rede ist.
2. In diesem Sinne Batini/Ceri/Navathe (1992, S. 195): "... (functional analysis) enables us to verify the completeness of the database, that is, to verify that all data required by functions are included into the database."
3. Im strengen Wortsinn handelt es sich dabei nicht unbedingt um Funktionen, da nicht genau ein ausgehender Datenfluß resultieren muß.

Funktion wird beschrieben durch einen Namen, einen eingehenden und einen ausgehenden Datenfluß. Ein Datenfluß wird lediglich durch einen Namen für eine bestimmte Informationsart gekennzeichnet. Da Informationen in der Regel nicht permanent im Fluß sind, sondern auch abgelegt werden, ist es zudem möglich, entsprechende Ablagen ("data store") zu modellieren. Dabei wird von der technischen Realisation einer Ablage abstrahiert. Darüber hinaus gibt es Akteure, die Informationen erzeugen bzw. aufnehmen. So mag ein Kunde am Telefon seinen Namen nennen oder - umgekehrt - nach dem Saldo seines Kontos fragen, ein Meßgerät mag Daten liefern, eine Maschine Steuerdaten benötigen. Der Informationsaustausch mit solchen Akteuren beschreibt also die Schnittstellen eines Informationssystem zu seiner Umwelt. Sie werden deshalb mitunter auch als Interface bzw. Schnittstelle bezeichnet. Abbildung 10 zeigt die wichtigsten Symbole von Datenflußplänen zusammen mit einem kleinen Beispiel.

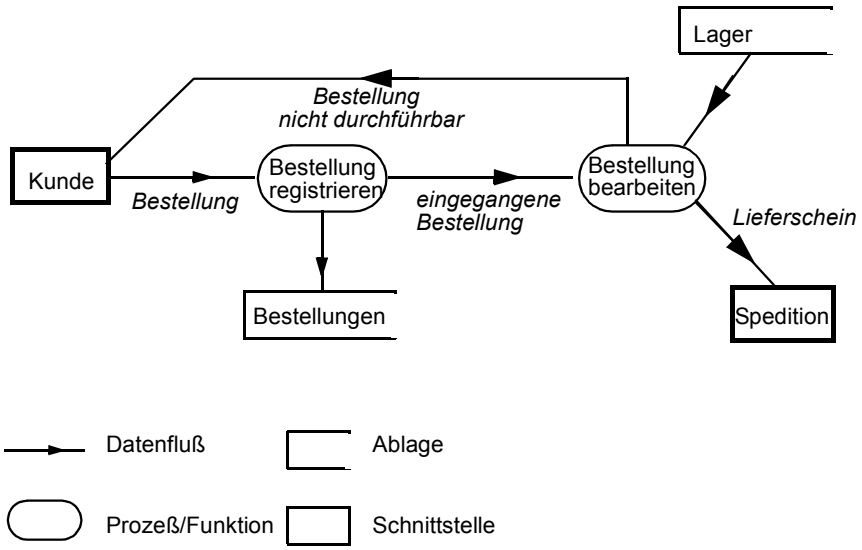


Abb. 10: Beispiel eines Datenflußplans

Für das Zusammenwirken von Datenflußplänen und konzeptuellen Datenmodellen sind zunächst Datenflüsse zu betrachten. Die in ihnen beschriebenen Daten sollten sich auch im Datenmodell wiederfinden. Entweder als Entitätstyp oder als Menge von Attributen verschiedener Entitätstypen. Während die Analyse vor allem der gegenseitigen Korrektur von Datenflußplan und Datenmodell dient, ist für den Entwurf anzustreben, die entdeckten Korrespondenzen durch die Verwendung gemeinsamer Namen deutlich zu machen - sofern das möglich ist.[1] Eine weitere Gemeinsamkeit kann die Betrachtung von Ablagen liefern. Sie können mit Entitäts- oder Relationstypen des ERM korrespondieren. Auch die

Schnittstellen sind für die Datenmodellierung von Bedeutung, da sie häufig Entitäten im Datenmodell repräsentieren. Schließlich liefern Funktionen, die mehr als eine Ablage bzw. Schnittstelle nutzen, Hinweise auf mögliche Beziehungen zwischen den jeweils repräsentierten Entitäten.

Für den Entwurf und die Repräsentation von Datenflußplänen ist es wesentlich, daß eine funktionale Dekomposition auf sie angewandt werden kann. Jede Funktion kann auf einer Stufe größerer Detaillierung selbst als Datenflußplan dargestellt werden. Grundsätzlich werden in Datenflußplänen keine Kontrollstrukturen abgebildet. Es werden lediglich die Datenflüsse beschrieben, die eine Funktion potentiell erzeugt - ohne daß die Bedingungen spezifiziert werden, unter denen der eine oder andere Ausgang produziert wird. Nähere Aufschlüsse über den internen Ablauf einer Funktion können allenfalls durch ihre Dekomposition vermittelt werden.

Es gibt eine Reihe von Prinzipien für den Entwurf von Datenflußplänen. Um die Anschaulichkeit nicht zu gefährden, wird häufig empfohlen, die Zahl der in einem Diagramm abgebildeten Funktionen nicht zu groß werden zu lassen. Hier werden unterschiedliche Werte genannt. Als oberste Grenze kann von 7+-2 ausgegangen werden. Wenn ein Entwurf komplexer gerät, sollte die Zahl der enthaltenen Funktionen durch geeignete Komposition verringert werden. Auf der anderen Seite ist zu klären, wie weit dekomponiert werden sollte. Grundsätzlich sollte eine Funktion weiter dekomponiert werden, wenn noch nicht hinreichend deutlich ist, was in ihr geschieht. Ein Kriterium also, das im Einzelfall subjektiver Interpretation bedarf. Es gibt keinen Konsens darüber, ob die unterste Dekompositionsstufe einer Funktion durch eine weitere Verfeinerung in Form von Kontrollstrukturen ergänzt werden soll. Die Verfechter einer konzeptuellen Modellierung, die möglichst unabhängig von Implementierungsaspekten durchgeführt werden sollte, wenden sich gegen eine solche Verfeinerung. Typisch dafür Batini/ Ceri/Navathe (1992, S. 128): "... the functional schema resulting after analysis *should not contain procedural aspects* ...". Im Unterschied dazu ist in SADT eine Beschreibung der Funktionen auf der untersten Dekompositionsstufe mit Hilfe von Pseudo-Code in Form sogenannter "mini-specs" vorgesehen.[1]

Die Qualität von Datenflußplänen hängt wesentlich davon ab, in welchem Maße es gelingt, die einzelnen Funktionen so zu modellieren, daß sie unabhängig voneinander sind. Die an einer Funktion durchgeführten Modifikationen sollten also möglichst keine unkontrollierten Seiteneffekte auf andere Funktionen haben. Der Zusatz "unkontrolliert" ist deshalb wichtig, da durch die Dekomposition gezielt eine Abhängigkeit zwischen Funktionen definiert wird.

1. Wenn ein Datenfluß aus Teilen mehrerer Entitäten besteht, ist im Datenmodell keine singuläre Referenz dafür vorhanden.

1. Vgl. dazu Balzert (1990), S. 21 f.

2.3.1.2 Zur Verbindung von Funktionenmodell und Datenmodell

Im Hinblick auf das Ziel, Funktionsredundanz zu minimieren und eine zentrale Bibliothek wiederverwendbarer Funktionen zu etablieren, ist es erstrebenswert, Datenflußpläne zu einem unternehmensweiten Funktionenmodell zu verdichten. Einen Ansatzpunkt dafür bieten die Dekompositionshierarchien, die auf *Funktionsbäume* abstrahiert werden können. Die Betrachtung mehrerer Funktionsbäume mag Ähnlichkeiten aufdecken, die eine Re-Modellierung der betroffenen Datenflußpläne und damit der zugehörigen Funktionsbäume empfehlen. Auf diese Weise entsteht als Pendant zum unternehmensweiten Datenmodell ein unternehmensweiter Funktionsbaum.

Auch wenn sich Daten- und Funktionsorientierung während der Analyse und des Entwurfs gegenseitig befruchten, stehen Datenmodell und Funktionenmodell getrennt nebeneinander. Es liegt auf der Hand, daß eine Verbindung beider Modelle über die Dateneingänge und -ausgänge einer Funktion erfolgen muß. Im einfachsten Fall entspricht die Struktur dieser Datenströme einem Entitätstyp des Datenmodells. Andernfalls ist jeweils eine geeignete externe Sicht auf das Datenmodell zu definieren. Wenn die Elemente des Funktionenmodell um Verweise auf zugehörige Elemente des Datenmodells angereichert werden - et vice versa, kann die dadurch begründete Form referentieller Integrität während der Pflege des Gesamtmodells werkzeuggestützt überwacht werden. Ein solches *cross-referencing* führt allerdings nicht zu einer Modularisierung. Es wird lediglich beschrieben, welche Funktionen welche Daten nutzen. Die Verwaltung von Daten wird also nicht in die Hoheit einzelner Funktionen gelegt.

Datenflußpläne bzw. daraus abgeleitete Funktionenmodelle reichern die Semantik von Datenmodellen durch folgende Integritätsbedingungen an:

- die Festlegung, welche Funktionen auf Daten operieren dürfen,
- Spezifikation von Eingangs- und Ausgangsdatenflüssen durch Verweis auf deren Definition im Datenmodell,
- referentielle Integrität zwischen Funktionen durch die Definition von Aggregationsbeziehungen (im Zuge der Funktionsdekomposition).

2.3.2 Die Modellierung dynamischer Aspekte

In einem Funktionenmodell wird nicht beschrieben, wodurch die Ausführung einer Funktion veranlaßt wird. Es bleibt zudem unberücksichtigt, welche dynamischen Beziehungen zwischen den Funktionen bestehen: In welchen Sequenzen werden Funktionen ausgeführt?[1] Können sie parallel ausgeführt werden? Es fehlt

1. Datenflußpläne beschreiben eben lediglich mögliche Datenströme zwischen Funktionen.

also letztlich eine Beschreibung von Kontrollstrukturen. Von besonderer Bedeutung für die Integrität eines Informationssystems sind dabei Zusammenfassungen zeitlich geordneter Funktionen, in denen die Ausführung einer Funktion nur wirksam wird, wenn alle anderen Funktionen ebenfalls ausgeführt werden (Transaktionskonzept). Es gibt eine Reihe von Ansätzen zur Modellierung solcher dynamischer Aspekte. Um einige wichtige zu nennen: Zustand- bzw. Zustand/Ereignis-Diagramme, auch Zustandsautomaten genannt, und verschiedene Formen von Petri-Netzen.[1] Ihnen allen ist gemeinsam, daß sie - in unterschiedlichen Abstraktionsformen - die Dynamik eines Systems mit Hilfe von Beziehungen zwischen Ereignissen und Zuständen bzw. Zustandsänderungen beschreiben. Die Modellierung dynamischer Zusammenhänge spielt im Rahmen der konzeptuellen Modellierung betrieblicher Informationssysteme sowohl in der Praxis als auch in der einschlägigen Literatur eine untergeordnete Rolle. Ein Grund dafür mag sein, daß Kontrollstrukturen häufig erst im Zuge des Programmentwurfs spezifiziert werden. Damit zusammenhängend ist daran zu denken, daß die Modellierung nicht-sequentieller Abläufe vor allem in der Systemprogrammierung und der Prozeßsteuerung von Bedeutung ist. Dennoch werden wir im folgenden einen Ansatz zur Modellierung dynamischer Zusammenhänge betrachten. Zum einen, weil es sich dabei um eine - im Sinn der oben genannten Modellierungsziele - konsequente Ergänzung von Daten- und Funktionenmodellierung handelt, zum anderen, weil es durchaus eine Reihe von Entwurfswerkzeugen gibt, die solche Ansätze nutzen.[2]

2.3.2.1 Prototypische Darstellung dedizierter Modellierungsansätze am Beispiel von Petri-Netzen

Petri-Netze wurden zu Beginn der sechziger Jahre (Petri 1962) mit dem Anliegen eingeführt, eine einheitliche Abbildung von Prozessen der Informationsverarbeitung und Informationsübermittlung in formaler Notation zu ermöglichen (Reisig 1982, S. 1). Sie basieren auf der mathematischen Netz-Theorie. Dadurch wird es möglich, bestimmte Formen der Korrektheit eines Netzes zu beweisen bzw. inkonsistent modellierte Sachverhalte (wie etwa Verklemmungen) durch die Anwendung formaler Verfahren aufzudecken. Ein Netz wird mit Hilfe vorgegebener Symbole für Ereignis- und Zustandsknoten sowie deren Verbindung durch Pfeile dargestellt. Im Laufe der Zeit wurden verschiedene Ausprägungen von Petri-Netzen entwickelt. Sie unterscheiden sich vor allem durch die jeweils für Ereignisse und Zustände darstellbare Semantik.

---

1. Zudem sind in diesem Zusammenhang Arbeiten aus dem Bereich der temporalen Logik zu nennen (wie etwa Rescher/Urquhart 1971; Brzoska 1990). Sie spielen allerdings im Rahmen der konzeptuellen Modellierung keine nennenswerte Rolle.
2. Vgl. beispielsweise Scheschonk (1990).

Auf dem geringsten semantischen Niveau sind sogenannte *Bedingungs/Ereignis-Netze* angesiedelt. Bedingungen - man könnte auch von Zuständen sprechen - werden dabei allein mit den expressiven Möglichkeiten der Aussagenlogik dargestellt: "Bestellung liegt vor" oder "Es ist Sonntag". Gleiches gilt für Ereignisse. Daraus folgt, daß Zustandsänderungen inhaltlich nicht zueinander in Beziehung gesetzt werden können. Es wird lediglich festgehalten, daß ein neuer Zustand (eine neue Bedingung) gilt, aber nicht, in welcher Weise er sich vom alten unterscheidet.

Während Zustände in Bedingungs/Ereignis-Netzen lediglich in zwei Ausprägungen, nämlich den Wahrheitswerten der Aussagenlogik, auftreten können, ist in *Stellen/Transitions-Netzen* eine differenzierter Kennzeichnung möglich. Stellen sind Zustandsknoten, die n verschiedene Zustände darstellen können. Dazu können einer Stelle 0 bis n sogenannte *Marken* zugeordnet werden. Jede Stelle wird durch eine Kapazität, nämlich die maximal zulässige Zahl von Marken gekennzeichnet. Ein zustandsveränderndes Ereignis wird durch das Schalten ("Feuern") einer Transition dargestellt. Ein solches Ereignis tritt ein, wenn alle Stellen, die unmittelbar vor der Transition liegen, die jeweils benötigte Mindestzahl von Marken aufweisen und bei den folgenden jeweils noch eine hinreichend große Kapazität verfügbar ist. Der Zustand des Netzes wird dann dadurch verändert, daß den Vorgängern jeweils eine vorgegebene Zahl (man spricht hier auch von Gewichtung) von Marken entnommen wird und an die folgenden in einem ebenfalls vorgegebenen Verhältnis weitergereicht wird. Stellen/Transitions-Netze sind vor allem für die Abbildung von Prozessen geeignet, deren Steuerung wesentlich mit Hilfe von Mengen bestimmter Gegenstände (dargestellt durch Marken) beschrieben werden kann. Hier ist beispielsweise an Produktionssteuerungsprozesse oder auch an die Ressourcenverwaltung in Betriebssystemen zu denken.

Im Hinblick auf die Modellierung betrieblicher Informationssysteme ist es wünschenswert, die verarbeiteten Gegenstände differenzierter, also semantisch angereichert, darstellen zu können. Zu diesem Zweck bieten sich vor allem *Prädikat/Ereignis*-Netze an.[1] Eine Stelle wird durch ein Prädikat ausgezeichnet. Sie kann Gegenstände unterschiedlicher Art aufnehmen. Das der Stelle zugeordnete Prädikat wird auf diese Gegenstände angewandt. Ein Ereignis tritt ein, wenn eine Bedingung zutrifft, die mit Hilfe der verwendeten Objekte und deren Eigenschaften - in Form von Attributen - gebildet wird. Durch ein Ereignis wird ein Objekt von einer vorgelagerten Stelle in die nachgelagerten weitergereicht.

1. Eine ausführliche formale Definition findet sich in Reisig (1990, S. 117 ff.). Einen verlgeichbaren Ansatz stellen sogenannte "Coloured Petri Nets" (Jensen 1986) dar. Sie sehen eine Visualisierung von Prädikaten mit Hilfe von Farben vor. Damit wird das Ziel verfolgt, dynamische Zusammenhänge übersichtlich und anschaulich darzustellen. Dabei ist vor allem an animierte Simulationen zu denken.

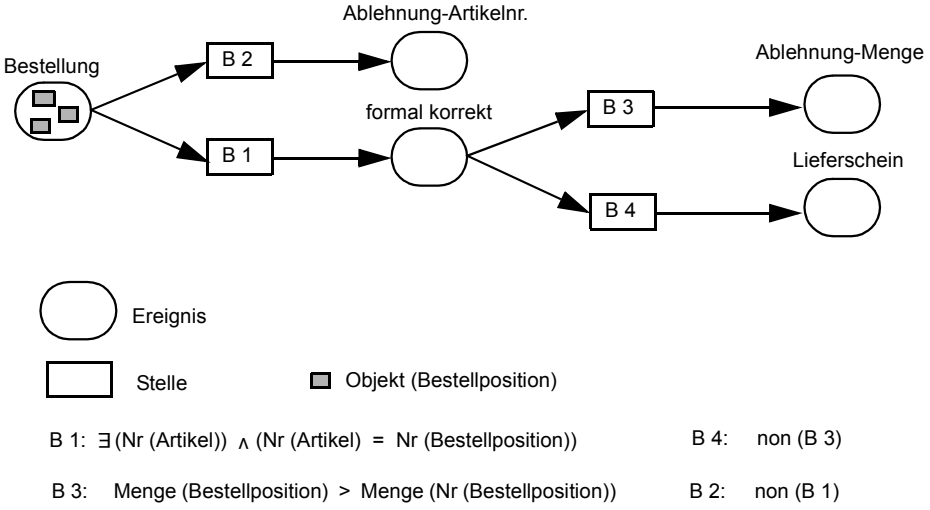


Abb. 11: Prädikat/Ereignis-Netz für die Bearbeitung einer Bestellung.

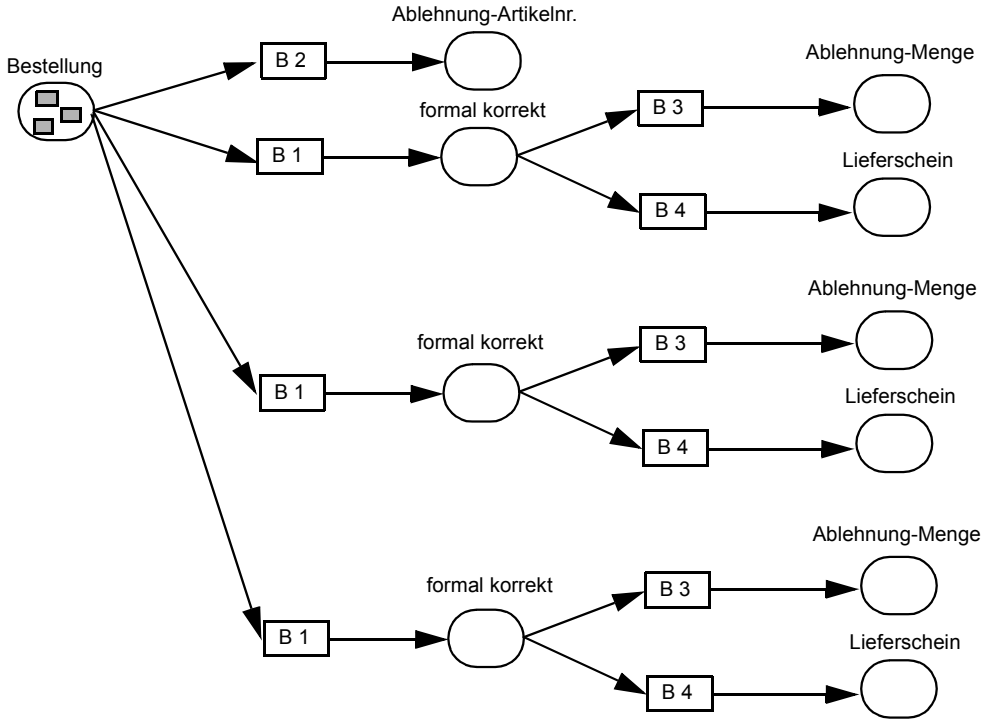


Abb. 12: Erhöhung der Kapazität eines Netzes durch nebenläufige Verarbeitung.

Um die Lesbarkeit des Netzes zu verbessern, können neben den Stellen auch

Ereignisse und Netzkanten beschriftet werden. Abbildung 11 zeigt das Beispiel eines Prädikat/Ereignis-Netzes, das eine stark vereinfachte Abbildung eines Bestellvorgangs darstellt. Dabei wird durch eine Transition, die durch die an ein Ereignis geknüpfte Bedingung ausgelöst wird, eine Bestellposition (Artikelnummer, Menge) von Stelle zu Stelle weitergereicht. Die Stellen sind durch Prädikate gekennzeichnet.[1] Die Variante in Abbildung 12 skizziert, wie Nebenläufigkeit dargestellt werden kann: Durch die Einführung weiterer Transitionen wird die Kapazität des Netzes erhöht. Dadurch daß die Dynamik eines abgebildeten Systems durch den Transport von Marken bzw. Objekten durch das Netz dargestellt wird und die Bedingungen für das Schalten der Transitionen formal vorgegeben sind, eignen sich Petri-Netze gut für die Visualisierung von Simulationen mit Hilfe geeigneter Animationswerkzeuge. Zum Zweck der Simulation ist eine prototypische Instanzierung vorzunehmen, indem dem Anfangszustand eine Menge von Objekten mit vorgegebenen Eigenschaften zugeordnet wird. In Abbildung 12 ist angedeutet, wie Simulationen animiert werden können: Ikonisierte Objekte fließen durch das Netz.

Um auch komplexe Prozesse übersichtlich darstellen zu können, erlauben Petri-Netze eine schrittweise Dekomposition. Dazu kann eine Transition durch ein Petri-Netz ersetzt werden.

2.3.2.2 Kritik

Petri-Netze erlauben eine anschauliche Darstellung zeitlicher Beziehungen. Das gilt besonders für die Abbildung nebenläufiger Prozesse. Die formale Definition der Semantik der verwendeten zeitlichen Beziehungen erlaubt die automatisierte Ermittlung unzulässiger Entwürfe. Die Bewertung von Gestaltungsalternativen kann durch die Simulation der mit ihnen einhergehenden Abläufe erleichtert werden. Simulationen können zudem erweitert werden durch Verfahren zur Optimierung[2] von Netzen. Damit sind Petri-Netze geeignet, Hinweise für die betriebswirtschaftliche Bewertung alternativer Abläufe zu liefern.

Den Vorteilen von Petri-Netzen stehen einige Nachteile gegenüber. So wird die Darstellung komplexer Fallunterscheidungen, wie sie etwa in Programmiersprachen durch binäre Verzweigungen oder mehrstufige Auswahl möglich sind, sehr schnell unübersichtlich. Auch wenn die Anschaulichkeit durch eine geeignete Komposition/Dekomposition gefördert werden kann, sind Petri-Netze für die Abbildung komplexer Kontrollstrukturen weniger geeignet als dedizierte Techniken zum Entwurf von Programmlogiken (wie etwa Struktogramme oder Pseudo-

1. Dieser Umstand wird ein wenig dadurch verdeckt, daß - aus Gründen der Anschaulichkeit - zum Teil Substantive verwendet werden.
2. Dazu sind bestimmte Randbedingungen vorzugeben, unter denen beispielsweise die Durchlaufzeit minimiert werden soll.

Code).

Von herausragender Bedeutung für unsere Betrachtung ist die Frage, inwieweit sich Petri-Netze - wobei wir uns auf Prädikat/Ereignis-Netze beschränken - mit Daten- und Funktionenmodellen integriert werden können. Dabei fällt zunächst auf, daß in der Praxis des Software-Engineering der Entwurf dynamischer Modelle vor allem auf die isolierte Beschreibung und Simulation komplexer Prozesse gerichtet ist. Eine Integration mit Daten- und Funktionenmodell ist zumeist nicht vorgesehen.[1]

Betrachten wir zunächst, wie eine Verknüpfung mit einem Datenmodell aussehen könnte. Im einfachsten Fall korrespondieren die Objekte in den Stellen mit Entitäten des ERM. Häufig jedoch wird eine solch einfache Zuordnung nicht möglich sein. Wenn in den einzelnen Verarbeitungsstellen eines Vorgangs einzelne Eigenschaften mehrerer Entitäten benötigt werden, reicht das einfache - für Simulationen so schön geeignete - Prozeßmodell von Petri-Netzen nicht mehr aus. Wenn in unserem Beispiel die Objekte der Art "Bestellposition" realistischerweise durch ein zugehöriges Objekt der Art "Kunde" ergänzt werden sollen, reicht es nicht hin, ein solches Objekt zusätzlich in die mit "Bestellung" gekennzeichnete Stelle zu legen: Das Ablaufmodell von Prädikat/Ereignis-Netzen sieht vor, daß die einzelnen Objekte einer Stelle nacheinander in die folgenden Transitionen gereicht werden. Es kann also nicht zwischen Objekten differenziert werden. Man müßte deshalb, falls beispielsweise zu Beginn des Vorgangs die Bonität des Kunden zu prüfen wäre, eine weitere Stelle einrichten. In dieser Stelle würden sich Objekte der Art "Kunde" befinden. Ein solche Trennung würde allerdings die Anschaulichkeit der Darstellung erheblich gefährden, da gemeinhin Kunde und Bestellposition zusammen zu einer Bestellung gehören.

Darüber hinaus ist zu berücksichtigen, daß die Beziehungen zwischen Kunde und Bestellposition nicht dargestellt werden können. Genau dies wäre aber zu tun, um die Synchronisation der Prüfung von Kundenbonität und den zugehörigen Bestellpositionen zu gewährleisten. Diese Unzulänglichkeit ist allerdings nicht allein den Prädikat/Ereignis-Netzen anzulasten, sondern auch dem ERM: Eine Bestellung läßt sich nicht vollständig als Entität darstellen. Falls das Datenmodell solche komplexen Objekte zuließe, müßte es das Prädikat/Ereignis-Netz erlauben, Stellen mit maximalen Kapazitäten auszuzeichnen und außerdem Möglichkeiten vorsehen, Kollektionen zu verarbeiten. Auf diese Weise könnte ein Objekt "Bestellung" (nicht zu verwechseln mit dem gleichnamigen Prädikat im

1. Bezeichnend dafür ist die Werkzeugübersicht in Batini/Ceri/Navathe (1992), in der dedizierte Ansätze für die Modellierung dynamischer Zusammenhänge überhaupt nicht als Werkzeugmerkmal erwähnt werden. Demgegenüber sind Werkzeuge für den Entwurf von Petri-Netzen (wie etwa Scheschonk) zumeist exklusiv für diesen Modellierungsansatz vorgesehen.

Beispiel) verwendet werden, das zusammengesetzt ist aus Angaben wie Kunde, Datum und dergleichen, sowie einer Kollektion von Bestellpositionen. Nach erfolgreicher Überprüfung der Kundenbonität könnte in der anschließenden Stelle die Kollektion aufgelöst werden, um eine parallele Verarbeitung der einzelnen Bestellpositionen zu modellieren.

Zur Anbindung an das Funktionenmodell bieten sich vor allem die Prozesse/ Funktionen in Datenflußdiagrammen an. Wenn sie mit dem in einem Petri-Netz abgebildeten Vorgang übereinstimmen, läßt sich leicht eine entsprechende Referenz etablieren. Falls sich solche Korrespondenzen nicht finden, kann versucht werden, eine Anbindung durch gegenseitige Anpassung (beispielsweise mit Hilfe von Kompositionen oder Dekompositionen) zu erreichen. Eine weitere Möglichkeit, die Elemente von Funktionenmodellen mit denen von Petri-Netzen zu verbinden, bieten die in den Ereignissen spezifizierten Bedingungen. Dabei kann es sich um Bedingungen handeln, deren Überprüfung eine mehr oder weniger umfangreiche Funktion voraussetzt. So kann die Prüfung der Bonität eines Kunden mehrere miteinander verknüpfte Vergleiche nötig machen. Auch die Evaluation der im Beispiel dargestellten prädikatenlogischen Präposition erfordert eine Prozedur. Bei solch komplexen Bedingungen kann die Definition entsprechender Funktionen sinnvoll und damit eine Anbindung an das Funktionenmodell möglich sein. Wenn es sich bei einer Bedingung jedoch allein um einen Vergleich zweier Werte handelt, entspricht die Definition einer dedizierten Funktion gewiß nicht der gängigen Granularität von Funktionenmodellen.

Die Integration mit dem Funktionenmodell ist also grundsätzlich möglich. Um ein Prädikat/Ereignis-Netz zur Generierung von Code oder Code-Fragmenten verwenden zu können, müßte allerdings auch eine zufriedenstellende Anbindung an ein ERM gewährleistet sein. Das ist jedoch nicht immer möglich. Es kann also nicht davon ausgegangen werden, daß die in einem Prädikat/Ereignis-Netz abgebildete Semantik durch ein automatisiertes Verfahren in eine implementierungsnähere Repräsentation transformiert werden kann. In jedem Fall bietet der Vergleich der verschiedenen Modelle die Chance, gegenseitige Korrekturen anzuregen. Beispielsweise mag im dynamischen Modell ein Objekt auftauchen, das nicht nur im Datenmodell nicht vorhanden ist, sondern auch gar nicht rekonstruierbar ist.

2.4 Zusammenfassende Beurteilung

Der oben entwickelte Bezugsrahmen wird im folgenden auf einen Modellierungsansatz angewendet, in dem ERM, Datenflußdiagramme und Prädikat/Ereignis-Netze gemeinsam zum Einsatz kommen. Im ersten Teil (strukturelle Aspekte) werden allein fortschrittliche Versionen des ERM und seine Transformation in das relationale Datenmodell bewertet. Dabei werden Werkzeuge, die auf dem

ERM aufbauen und zusätzliche Funktionen (wie etwa die Gestaltung von Aspekten der Benutzerschnittstelle) bereitstellen, ausgeklammert. Im zweiten Teil (funktionale/dynamische Aspekte) ist die Bewertung von Datenflußplänen und Prädikat/Ereignis-Netzen zusammengefaßt. Da es für diese Techniken im Unterschied zum ERM keine gängigen Transformationen in nachfolgende Stufen gibt, wird ein entsprechender Vergleich hinfällig. Die Bewertung der Produktionsumgebung ist vor allem an gegenwärtigen CASE-Werkzeugen und RDBMS orientiert. Die Erläuterung der Bewertungskriterien findet sich oben unter III.2.1.

Strukturelle Aspekte Bewertung

			Modell	nach Übergang in anschließende Repräsentation
Attribute	maschinenorientiert	definierbar als Intervall	ja	teilweise
		durch funktionale Spezifikation	nein	nein
	anwendungsorientiert	durch Enumeration	ja	nein
		durch funktionale Spezifikation	nein	nein
		Entitätstyp	nein	nein
	Kardinalität	minimal = 0	ja	mit Einschränkung
		maximal > 1	ja	nein
	Zugriffsrecht		nein	ja
	Beziehungen		nein	nein
Identität einer Entität vom Zustand determiniert			ja	ja
Historie	generell		nein	nein
	partiell		nein	nein

Strukturelle Aspekte

				Bewertung	
				Modell	nach Übergang in anschließende Repräsentation
Beziehungen zwischen Entitäten	Kardinalität	einfach		ja	nein
		min-max	eingeschränkt	ja	nein
			frei	nein	nein
	Richtung	Rolle		ja	nein
		inverse Beziehung		nein	nein
	Referentielle Integrität			ja	ja
	Aggregation			nein	nein
	sonstige			nein	nein
Generalisierung	einfach			ja	nein
	mehrfach			nein	nein
Views/Design der Benutzerschnittstelle					
Spezifikation von Views auf einen Entitätstyp				ja	ja
Spezifikation von Views auf eine Menge von Entitätstypen				ja	ja
Spezifikation einer prototypischen Benutzerschnittstelle für einen Entitätstyp				nein	ja
dito, für eine Menge von Entitätstypen				nein	ja

Dynamische/funktionale Aspekte		Bewertung
Operationen/Funktionen		
Schnittstellenspezifikation		ja
Preconditions		nein
Postconditions		nein
Trigger		nein
Ausnahmenbehandlung		nein
Systemverhalten		
Zeitliche Aspekte/ Kontroll-strukturen	für Entität	nein
	für Kollektionen	nein
Sonstige Aspekte		
Modula-risierung	Kohärenz	nein
	Assoziierung	mit Einschränkung
Produktionsumgebung		
Datenbankmanage-mentsystem	DDL	ja
	DML	ja
	Standards	ja
	Kontrolle über Ope-rationen	mit Einschränkungen
Werkzeugunterstützung		ja

Zusammenfassend können wir festhalten, daß die gemeinsame Verwendung von Datenmodellen, Funktionenmodellen und dynamischen Modellen (etwa in Form von Petri-Netzen) eine konzeptuelle Modellierung auf einem hohen semanti-

schen Niveau erlaubt. Alle für die Implementierung eines Informationssystems wesentlichen Eigenschaften eines Anwendungsbereichs können abgedeckt werden und durch fortschreitende Dekomposition auch im Detail beschrieben werden. Dabei ist allerdings eine Reihe von Defiziten zu berücksichtigen. Die drei Modelle stehen isoliert nebeneinander. Da unterschiedliche Abstraktionen verwendet werden, ist es für den Modellierer mit einem großen Aufwand verbunden, die Beziehungen zwischen korrespondierenden Teilen der Modelle herauszuarbeiten und im Verlauf des fortschreitenden Entwurfs zu pflegen. Für Datenflußpläne wie auch für Petri-Netze gibt es in aller Regel keine entsprechenden implementierungsnäheren Repräsentationsformen. Es werden also für die Implementierung andere Konstrukte verwendet. Dadurch wird einerseits ein schnelles Prototyping erschwert, andererseits ist die Systempflege mit dem Problem belastet, vielfältige Referenzen zwischen den verschiedenen Repräsentationsformen zu berücksichtigen. Daneben ist festzustellen, daß Petri-Netze und Datenflußpläne für viele Betrachter nicht hinreichend anschaulich sind.

3. Objektorientierte Modellierung

Objektoriente Software-Entwicklung wie auch objektorientierten Systeme erfahren gegenwärtig eine Aufmerksamkeit wie kaum ein anderer softwaretechnischer Ansatz seit der Einführung höherer Programmiersprachen. Die objektorientierte Betrachtung von Informationssystemen geht auf Entwicklungen im Bereich der Programmiersprachen zurück und wurde nicht zuletzt durch Forschungsarbeiten in der sogenannten Künstliche Intelligenz-Forschung angereichert.[1] Nachdem objektorientierte Ansätze lange Zeit nur im Stillen blühten, haben sie mittlerweile nahezu alle Bereiche der Informationssystem-Forschung erobert: Glaubt man den einschlägigen Konferenzbeiträgen, wird in der Forschung nur noch objektorientiert entwickelt. Aber auch in der Praxis der Software-Entwicklung stoßen objektorientierte Ansätze auf großes Interesse. Es dokumentiert sich einerseits in der Verfügbarkeit einer wachsenden Zahl objektorientierter Werkzeuge und System, andererseits in einer kaum noch überschaubaren Vielfalt einführender Publikationen.

Gleichwohl die fortschreitende Kommerzialisierung nicht nur für die Verbreitung, sondern vor auch für die Weiterentwicklung objektorientierter Ansätze von großer Bedeutung ist, hat sie auch ihre Schattenseiten. So ist "objektorientiert" jenseits von konkreten Kriterien zu einem Etikett geworden, das eng mit Fort-

1. Hier ist vor allem an Programmiersprachen wie Ada, Simula und Smalltalk zu denken. In der KI-Forschung sind im Zeitverlauf viele objektorientierte Formalismen entstanden. Besonders hervorzuheben ist dabei das von Minsky (1975) vorgeschlagene Frame-Konzept und die von Hewitt/Bishop/Steiger (1973) präsentierte Sprache zur Beschreibung nebenläufiger Objekte (ACTOR).

schrittlichkeit und Qualität assoziiert ist. Das verleitet die Anbieter mancher Produkte dazu, sich mit falschen Federn zu schmücken.[1] Die wachsende Popularität geht einher mit einer wenig einheitlichen Terminologie und der Verbreitung zum Teil diffuser Vorstellungen über die Leistungsfähigkeit des Ansatzes. In diesem Sinne Rentsch (1982, S. 51) - wobei man die Jahreszahlen m.E. jeweils um zehn inkrementieren sollte:

> "My guess is that object-oriented programming will be in the 1980s what structured programming was in the 1970s. Everyone will be in favor of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is."

Auch wird mitunter der mystifizierende Eindruck geweckt, als seien objektorientierte Ansätze mit einer völlig neuen Sicht des Software-Entwurfs verbunden. Sie sind allerdings gewiß nicht so revolutionär wie die mitunter in leidenschaftlicher Polemik geführten Diskussionen zwischen Anhängern traditioneller Ansätze und den Proponenten der Objektorientierung vermuten lassen. Vielmehr handelt es sich um die (mehr oder weniger) konsequente Anwendung von Prinzipien, über die es im Software-Engineering seit geraumer Zeit einen weiten Konsens gibt.

Im folgenden werden zunächst zentrale Begriffe definiert. Im Anschluß daran wenden wir uns der objektorientierten konzeptuellen Modellierung zu. Dazu wird einleitend eine größere Zahl von Modellierungsansätzen anhand weniger Kriterien dargestellt. Anschließend werden zwei der zur Zeit bedeutendsten Ansätze näher untersucht.

3.1 Terminologie

Mit der Einführung höherer Programmiersprachen war das Ziel verbunden, dem Entwickler die Möglichkeit zu geben, ein Programm in größerer Anlehnung an die Begrifflichkeit des jeweiligen Anwendungsbereichs zu formulieren als dies maschinenorientierte Sprachen gestatten. Die in frühen höheren Programmiersprachen bereitgestellten Datentypen erlauben zwar die Unabhängigkeit von den Spezifika einer bestimmten Prozessor-Architektur, sind allerdings deutlich maschinenorientiert, da sie in der Regel mit internen Darstellungen von Zahlen und Zeichenketten korrespondieren. Auch die Operationen, die auf Daten dieser Art anwendbar sind, sind deutlich an Maschinenoperationen angelehnt. Die Entwicklung sogenannter *abstrakter Datentypen* zielte darauf, Konstrukte bereitzustellen, deren Semantik weniger von der Hardware vorgegeben ist, als vielmehr im Hinblick auf einen Anwendungsbereich definiert werden konnte. Abstrakte Datentypen entstehen durch die Festlegung einer Datenstruktur und einer Menge von Operationen, die auf diese Datenstruktur angewendet werden können. Ein

1. Dazu King (1989, S. 24) in (selbst-) kritischer Ironie: "If I were trying to sell (my cat) ... I would argue that he is object-oriented".

wesentliches softwaretechnisches Ziel dabei war eine gegenüber herkömmlichen Programmiersprachen verbesserte Modularisierung. Daten und darauf operierende Verfahren werden auch vom Compiler als Einheit betrachtet. Es kann festgelegt werden, daß Daten nur durch die Operationen des abstrakten Datentyps, aus dem sie instanziert werden, manipuliert werden können. Dadurch wird ein wirksamer Schutz vor Seiteneffekten gewährleistet.

Das Konzept eines Objekts bzw. einer Klasse setzt unmittelbar auf dem eines abstrakten Datentyps auf. Wie auch für einen abstrakten Datentyp werden für eine Klasse eine Datenstruktur und darauf operierende Verfahren definiert. Im Unterschied zu abstrakten Datentypen kann dabei auf eine Oberklasse referiert werden. Deren Merkmale (strukturelle wie prozedurale) werden dann durch *Vererbung* in die neue Klasse übernommen. Wenn eine Klasse nur aus einer Oberklasse erben kann, spricht man von Einfachvererbung; sind es mehrere, liegt Mehrfachvererbung vor. Analog zu abstrakten Datentypen wird ein Objekt aus einer Klasse instanziert, indem der für die Daten nötige Speicherplatz allokiert wird und eine Referenz zu den mit der Klasse definierten Operationen etabliert wird. Dabei wird der Datenschutz besonders betont: Idealtypisch bleiben die Daten eines Objekts allen anderen Objekten verborgen, sie sind im Objekt *verkapselt*.[1] Der Zugriff erfolgt allein über die *Dienste* eines Objekts. Dienste sind die Schnittstellen der Operationen bzw. Prozeduren, die ein Objekt nach außen hin anbietet. Die Menge der Dienste wird auch als Schnittstelle oder Protokoll eines Objekts bezeichnet.

Neben Verkapselung und Vererbung gibt es für den hier verwendeten Objektbegriff ein weiteres Charakteristikum: *Polymorphie*. Polymorphie bezeichnet den Umstand, daß ein in verschiedenen Klassen in gleicher Weise präsentierter Dienst durch unterschiedliche Operationen/Prozeduren erfüllt werden kann. Dieser Umstand ist letztlich allein bedeutend für die zur Kommunikation zwischen Objekten nötigen Angaben: Die Benennung eines Dienstes allein ist nicht hinreichend, vielmehr muß auch das zugehörige Objekt identifiziert werden. Im Unterschied zu den Prozedur- und Funktionsaufrufen traditioneller Sprachen werden die Dienste eines Objekts genutzt, indem dem Objekt eine *Nachricht* geschickt wird. Die Nachricht besteht aus der Bezeichnung des gewünschten Dienstes und enthält gegebenenfalls Parameter. Mitunter wird als weiteres charakteristisches Merkmal "Objektidentität" genannt. Damit wird betont, daß ein Objekt eine Identität hat, die unabhängig von seinem Zustand ist. Diese Eigenschaft ist also nur dann von Bedeutung, wenn es gilt, Objekte in objektorientierten Datenbankmanagement-Systemen von Tupeln in relationalen Datenbankmanagement-Systemen abzugrenzen.

1. Es gibt objektorientierte Programmiersprachen, wie z.B. C++, die Verkapselung lediglich optional anbieten.

Neben den originär softwaretechnischen Merkmalen ist für unsere Betrachtung vor allem von Bedeutung, daß das Konzept einer Klasse bzw. eines Objekts auf eine besonders anschauliche Beschreibung von Anwendungsbereichen zielt. Die grundlegende Vorstellung dabei ist, daß Objekte unmittelbar mit Gegenständen der abzubildenden Realität korrespondieren. Die Zusammenfassung gleichartiger Objekte zu Klassen entspricht gängigen Konzeptualisierungen. Ähnliches gilt für das Generalisieren über oder das Spezialisieren von Klassen. Polymorphie schließlich entspricht dem gewohnten Umstand, daß ähnliche Operationen, auf unterschiedliche Objekte angewandt, in gleicher Weise benannt werden.

Um von den Spezifika einer bestimmten objektorientierten Programmiersprache unabhängig zu bleiben und gleichzeitig eine anschaulichere Darstellung zu ermöglichen als sie Programmcode bietet, ist für die Entwicklung großer Anwendungssysteme eine konzeptuelle, objektorientierte Modellierung angeraten. Sie beinhaltet einerseits die Beschreibung von Klassen, die neben strukturellen Merkmalen auch prozedurale und funktionale umfaßt. Andererseits dient sie der Darstellung von Beziehungen zwischen Klassen beziehungsweise Objekten.

3.2 Überblick über Modellierungsansätze

In der Literatur findet sich mittlerweile eine beträchtliche Zahl objektorientierter Modellierungsansätze. Der größte Teil dieser Ansätze - zumeist als Forschungsberichte oder Konferenzbeiträge publiziert - präsentiert sich in vorläufiger und rudimentärer Form. Eine Reihe umfassend beschriebener Methoden liegt in Form von Lehrbüchern vor. Im Unterschied zur Datenmodellierung hat sich bisher kein dominanter Ansatz zur objektorientierten Modellierung herausgebildet. Der folgende Überblick, mit dem das Bemühen, nicht aber der Anspruch um Vollständigkeit verbunden ist, zielt nicht darauf, eine vergleichende Beurteilung der Ansätze zu bieten - dazu sind die Darstellungen häufig nicht detailliert genug.[1] Er dient vielmehr als Beleg für den Zustand dieses Forschungsgebiets. Er ist gekennzeichnet durch eine hohe Dynamik und durch viele einzelne Ansätze. Allen Gemeinsamkeiten zum Trotz ist eine Konsolidierung auf eine Referenzmethode nicht in Sicht. Die einzelnen Methoden werden häufig danach differenziert, ob sie eher der Analyse- oder eher der Entwurfsphase zuzurechnen sind. Diese

1. In Monarchi/Puhr (1992) findet sich ein strukturierter, allerdings wenig detaillierter Vergleich von 23 der im folgenden aufgelisteten 39 Methoden. Ein ausführlicherer, aber immer noch oberflächlicher, Vergleich von vier ausgewählten Ansätzen findet sich in Hsieh (1992). Hong/Goor (1993) vergleichen sechs Ansätze, indem sie dazu eine übergreifende "supermethodology" einführen. Dadurch bieten sie einen übersichtlichen Vergleich, beschränken sich aber auf wenige Kriterien, die zudem nicht immer in angemessener Weise ausgewählt sind (so beschreiben sie zum Teil Eigenschaften von Werkzeugen, nicht von Methoden). Ein weiterer Vergleich von objektorientierten Analysemethoden findet sich in De Champeaux/Faure (1992).

Unterscheidung wird im folgenden nicht aufgegriffen, da für uns die Konzeptualisierung von Objektmodellen im Vordergrund steht. Analysemethoden fokussieren vor allem auf das Problem, wie ein Realitätsausschnitt zu untersuchen ist, um die Konzepte zu identifizieren, die für seine objektorientierte Modellierung von Bedeutung sind. In zwei Fällen haben die Verfasser (Coad/Yourdon, Shlaer/Mellor) ihre Methoden in zwei Bänden beschrieben. In diesen Fällen gehen beide Bände zusammengefaßt in die Übersicht ein. Die Jahresangabe bezeichnet das Erscheinungsjahr der zugehörigen Publikation, also nicht unbedingt das Jahr, in dem die Methode fertiggestellt wurde. Nur wenige Methoden sind von ihren Verfassern durch eine eigenständige Bezeichnung gekennzeichnet.

Verfasser	*Bezeichnung*	*Jahr*	*Publikation*
Alabios		1988	A
Alabisco		1988	K
Ackroyd/Daum		1991	A
Berard		1986	H
Bailin		1989	A
Booch		1990	M
Boyd	PAMELA	1987	A
Buhr		1984	M
Cherry	PAMELA 2	1987	H
Coad/Yourdon	OOA/OOD	90/91	M
Cunningham/Beck		1986	K
Desfray		1990	K
Edwards	Ptech	1989	A
Embley et al.		1992	M
ESA	HOOD	1989	H
Felsinger		1987	M
Ferstl/Sinz	SOM	90/91	A
Firesmith		1992	M
Henderson-Sellers/Constantine		1991	A
Jacobson et al.		1992	M
Johnson/Foote		1988	A
Kadie		1986	U
Kappel/Schrefl		1991	K
Lee/Carver		1991	A
Liskov/Guttag		1986	M
Masiero/Germano		1988	A
McGregor/Sykes		1992	M

Verfasser	*Bezeichnung*	*Jahr*	*Publikation*
Mullin		1989	M
Nielsen		1988	H
Odell		1992 b	A
Page et al.		1989	K
Rajlich/Silva		1987	U
Seidewitz/Stark		1987	A
Robinson		1992	M
Shlaer/Mellor		88/92	M
Rumbaugh et al.	OMT	1991	M
Velho/Carapuca	SOM	1992	K
Wasserman et al.		1990	A
Wirfs-Brock/Wilkerson		1990	M

M Monographie
A Zeitschriftenaufsatz
H Handbuch
U Universitäts-/Forschungsbericht
K Konferenzbeitrag

Abb. 13: Ansätze zur objektorientierten Modellierung

Die Vielzahl der Methoden dokumentiert einerseits das rege Forschungsinteresse, andererseits trägt sie dazu bei, daß sich viele an objektorientierten Methoden interessierte Entwickler verunsichert fühlen.[1] Einige Studien (Mannino 1987, Hewlett Packard 1991) zielen auf eine vergleichende Beurteilung, berücksichtigen allerdings jeweils nur wenige (nicht mehr als fünf) Ansätze.

Auch wenn es keine Methode bisher zu einer großen Verbreitung in der Praxis gebracht hat, so sind doch einige Ansätze in der Szene der objektorientierten Entwickler besonders bekannt. Von herausragender Bedeutung sind zur Zeit drei Methoden. Coad/Yourdon (1990, 1991) profitieren dabei vor allem vom Ruhm ihrer konventionellen Methoden. Beide Bücher sind darauf gerichtet, eine leichtverständliche, mit vielen Beispielen angereicherte Einführung in die Methoden zu geben. Sie enthalten eine Reihe hilfreicher Heuristiken. Die Methoden sind

1. Die Zwiespältigkeit dieser Situation artikuliert Berard in einem Beitrag der News-Net-Group "comp.obj" (Oktober 1991) in treffender Weise: "I have good news and bad news. The good news is that there has been a great deal of work in the area of "object-oriented software engineering". The bad news is that there has been a great deal of work in the area of "object-oriented software engineering".

vor allem auf strukturelle, statische Aspekte konzentriert. Auf diese Weise wird jenen Lesern, die in konventioneller Datenmodellierung qualifiziert sind, eine schrittweise Erweiterung vertrauter Sichten geboten. Diese didaktischen Vorteilen gehen einher mit einer zum Teil oberflächlichen Darstellung und einem Mangel an softwaretechnischer Fundierung.

Im Vergleich dazu sind die Ansätze von Booch (1990) und Rumbaugh et al. (1991) umfassender und detaillierter. Auch wenn beide deutliche Gemeinsamkeiten aufweisen, sind sie doch so unterschiedlich strukturiert, daß sie sich gegen eine gemeinsame Darstellung sperren. Sie werden deshalb im folgenden nacheinander beschrieben und anschließend in einer zusammenfassenden Beurteilung miteinander verglichen. Das geschieht nicht zuletzt deshalb, da beide Ansätze die in IV.2.1.1 entworfene Modellierungsmethode wesentlich inspiriert haben. Die in traditionellen Ansätzen übliche Trennung in Analyse- und Entwurfsphase wird von Booch und Rumbaugh et al. weitgehend aufgehoben: Es wird als wesentlicher Vorteil objektorientierter Modellierung angesehen, daß Objekte einerseits eine natürliche Abstraktion realer Objekte darstellen (was sie für die Analyse geeignet macht) und andererseits ein solides softwaretechnisches Konzept darstellen, das über den Entwurf hinaus bis in die Implementierung übernommen werden kann.[1] Damit bleiben für die frühe Analysephase vor allem Heuristiken für das Entdecken von Objekten.[2] Im Unterschied dazu betonen Jacobson et al. (1992), die in jüngster Zeit wachsende Beachtung finden

3.3 Object Modeling Technique

Rumbaugh arbeitete mit einer Reihe von Kollegen in einer Forschungsabteilung von General Electric an einer systematischen Anleitung für objektorientierte Software-Entwicklung. Zur Unterstützung der daraus entstandenden Methode, "Object Modeling Technique" (OMT) genannt, wird mittlerweile von General Electric auch ein dediziertes Werkzeug angeboten. Rumbaugh et al. unterscheiden drei Teilmodelle des Entwurfs: ein statisches Objektmodell, ein funktionales Modell und ein dynamisches Modell.

---

1. In diesem Sinne Rumbaugh et al. (1991, S. 21): "Objects serve two purposes: They promote understanding of the real world and provide a practical basis for computer implementation."
2. So empfehlen Shlaer/Mellor (1988, S. 15) die Suche nach "tangible things", "roles", "events" und "interactions". Coad/Yourdon (1990, S. 62) weisen unter anderem auf "devices", "roles", "locations" und "organizational units" hin. Ein weiterer Ansatz, der in jüngster Zeit zunehmende Beachtung findet, ist vor allem auf die Erfassung und Konzeptualisierung von Anwendungsdomänen gerichtet, beginnt also nicht erst unmittelbar mit dem Entwurf von Klassen (Jacobson et al. 1992). In seinem Mittelpunkt steht die systematische Erfassung von Szenarien, die "use cases" genannt werden.

3.3.1 Das Objektmodell

Das Gerüst der OMT bildet ein Objektmodell. Dazu ist, wie die folgende Charakterisierung (Rumbaugh et al. 1991, S. 6) zeigt, eine Reihe von Begriffen zu berücksichtigen:

> "The *object model* describes the static structure of the objects in a system and their relationships. The object model contains object diagrams. An *object diagram* is a graph whose nodes are object *classes* and whose arcs are *relationships* among classes."

Die grafische Darstellung eines Objektmodells, bzw. eines Teils desselben, erfolgt mit Hilfe von Objektdiagrammen, für die zwei Ausprägungen unterschieden werden: "class diagrams" und "instance diagrams". Instanzendiagramme enthalten konkrete Instanzen einer Klasse und dienen lediglich der beispielhaften Veranschaulichung des Modells. Sie sind sicherlich nicht Bestandteil der eigentlichen konzeptuellen Modellierung, denn - wie Rumbaugh et al. (1991, S. 22) selbst ausführen: "The notion of abstraction is at the heart of the matter." Im Unterschied dazu werden in einem Klassendiagramm Klassen verwendet. Sie stehen stellvertretend für jede einzelne ihrer möglichen Instanzen (Objekte). Eine Klasse wird beschrieben durch eine Menge von Attributen und Operationen. Attribute werden durch einen Datentyp gekennzeichnet. Für uns ist dabei bedeutsam, daß explizit ausgeschlossen wird, daß Attribute selbst Objekte sein können, also Instanzen einer im Modell an anderer Stelle definierten Klasse.

Rumbaugh et al. äußern sich nicht explizit über die mögliche Kardinalität von Attributen. Es hat den Anschein als sei die Kardinalität implizit immer eins. Attribute sind nicht unbedingt durch im Objekt verwaltete Daten beschrieben. Vielmehr sind auch sogenannte "*derived attributes*" vorgesehen, die - durch geeignete Operationen - aus den Werten anderer Attribute ermittelt werden. Die Operationen einer Klasse werden im Klassendiagramm durch ihren Namen sowie die zugehörigen Parameter beschrieben. Es wird außerdem empfohlen, zwischen Operationen, die nur Daten eines Objekts lesen, und solchen, die Daten verändern, zu unterscheiden.

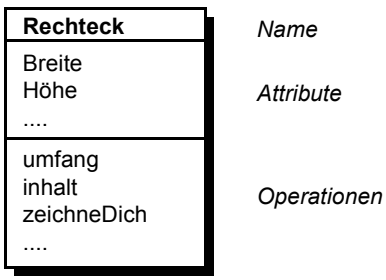


Abb. 14: Darstellung einer Klasse in OMT. Je nach gewünschtem Detaillierungsgrad des Diagramms können weitere Angaben - wie die Datentypen der Attribute oder die Parameter der Operationen - hinzugefügt werden.

Zwischen den Klassen eines Klassendiagramms können Beziehungen ("associations") beschrieben werden. Dabei ist es sinnvoll - auch wenn Rumbaugh et al. dies nicht explizit tun - zwischen Klassen als Abstraktion und Klassen als Stellvertreter der Menge ihrer Objekte zu unterscheiden. Generalisierung beschreibt eine Beziehung zwischen abstrakten Konzepten: Die gemeinsamen Merkmale mehrerer Klassen werden in einer neuen Klasse zusammengefaßt. Falls solche gemeinsamen Merkmale nicht zu einer Abstraktion realweltlicher Objekte führen, können sie als *abstrakte Klassen* gekennzeichnet werden. OMT erlaubt die Darstellung von Mehrfachvererbung.

Andere Beziehungen gelten für beliebige Instanzen einer Klasse. Solche Beziehungen können durch verschiedene Eigenschaften ausgezeichnet werden. Eine Kardinalität ("*multiplicity*") beschreibt, wie viele Instanzen einer Klasse mit einer Instanz der assoziierten Klasse (die dieselbe sein darf) *maximal* verbunden sein dürfen. Für die grafische Darstellung stehen drei Symbole (für 1:1, 1:n und n:m) zur Verfügung. Eine Klasse kann innerhalb einer Beziehung - wie auch beim ERM - durch eine *Rolle* ausgezeichnet werden. Für spezielle Konstellationen stehen zwei weitere Kennzeichnungen zur Verfügung. So können die durch eine Klasse repräsentierten Objekte als geordnet charakterisiert werden. Als Beispiel dafür wird die Beziehung "visible on" zwischen "Window" und "Screen" genannt. Schließlich können Kardinalitäten durch sogenannte "Qualifier" eingeschränkt werden. Während beispielsweise eine Beziehung zwischen Dateiverzeichnis und Datei durch die Kardinalität 1:n gekennzeichnet ist, kann durch den Qualifier "Dateiname" und die Änderung der Kardinalität auf 1:1 ausgedrückt werden, daß innerhalb eines Verzeichnisses ein Dateiname eine Datei eindeutig kennzeichnet.

In OMT können Beziehungen selbst als Klassen modelliert werden. Diese Option wird mit dem plausiblen Hinweis darauf begründet, daß eine Beziehung Merkmale haben mag, die keine originären Eigenschaften der assozierten Objekte darstellen. Wenn also zwei Objekte der Klasse "Person" durch die Beziehung "verheiratet mit" verbunden sind, könnten Merkmale wie "Hochzeitsdatum" der Beziehung zugeordnet werden. Auch wenn es wenig sinnvoll erscheint, ein solches Merkmal jeder Person zuzuordnen, ist die Handhabung von Beziehungen als Objekte (und damit die Einführung von Beziehungsklassen) problematisch. Dazu an dieser Stelle nur soviel: Ein Objekt sollte eine Identität unabhängig von anderen Objekten haben. Für eine Beziehung ist dies gerade nicht der Fall. Die Trennung von Objekten und Beziehungen ist eine wichtige Orientierung für den Entwurf von Modellen: Objekte sollen eben Abstraktionen realweltlicher Objekte darstellen - zwischen denen Beziehungen bestehen können. Durch das - von Rumbaugh et al. nicht forcierte - Motto "alles ist ein Objekt" verliert diese Orientierung an heuristischer Kraft und führt bei konsequenter Anwendung zu

einer erheblichen Steigerung von Komplexität: Wenn Beziehungen Objekte sind, können auch zwischen Beziehungen Beziehungen bestehen.

Zur Modellierung von *Aggregation* steht eine semantisch angereicherte Beziehung zur Verfügung. Sie impliziert einerseits *Transitivität*, andererseits *Anti-Symmetrie*.[1] Beziehungen können grundsätzlich zwischen Objekten beliebig vieler Klassen hergestellt werden. Es wird allerdings empfohlen, sich auf binäre Beziehungen zur beschränken.

In Anlehnung an den Objektbegriff in Smalltalk können Klassen selbst als Objekte ("class descriptor") beschrieben werden. Die Klasse einer solchen Klasse heißt "metaclass". Es können ihnen also Attribute und Operationen zugeordnet werden, die allein für die Klasse, nicht für ihre Instanzen gelten. Typische Beispiele für Operationen sind solche, die eine Instanz erzeugen oder über die Anzahl der Instanzen Auskunft geben.

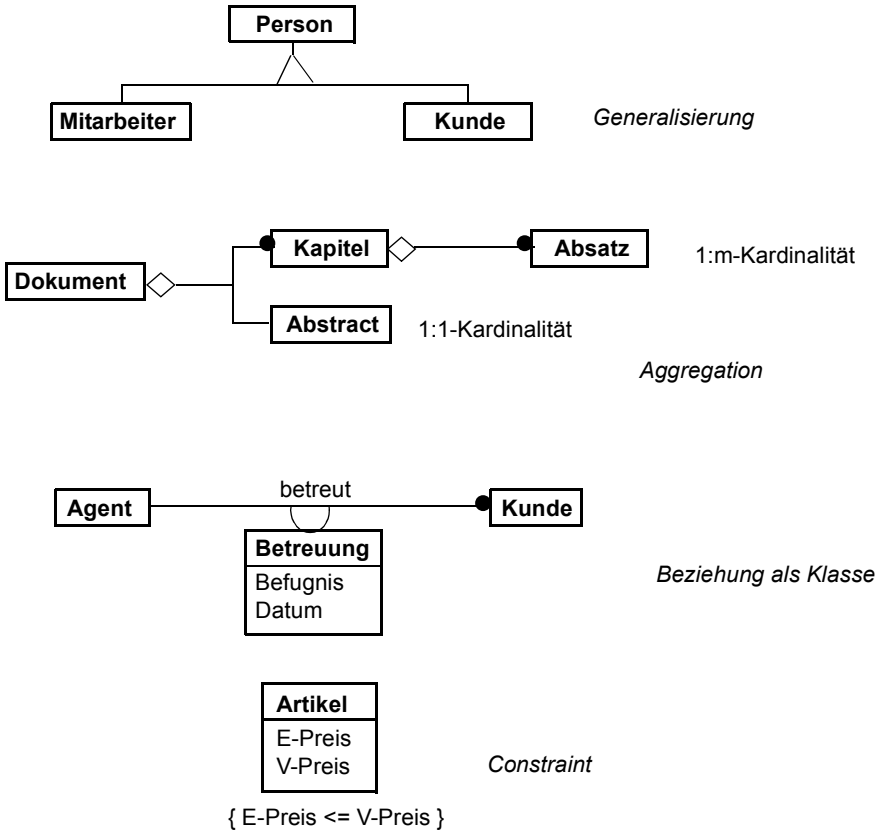


Abb. 15: Darstellung ausgewählter Konzepte in OMT

1. Wenn gilt "A beinhaltet B" und "B beinhaltet C", dann gilt auch "A beinhaltet C" (Transitivität). Anti-Symmetrie besagt, daß duch "A beinhaltet B" "A ist Teil von B" ausgeschlossen wird.

Um ein Objektmodell weiter mit Semantik anzureichern, können "*constraints*" verwendet werden. Solche Integritätsbedingungen schränken die zulässigen Zustände eines Objekts oder eines Attributs in Abhängigkeit von den Zuständen anderer Objekte/Attribute ein. So könnte beispielsweise ein Constraint vorsehen, daß der Verkaufspreis eines Artikels nicht kleiner sein darf als der Einkaufspreis.

Für alle skizzierten Konzepte bietet OMT eine grafische Repräsentation, die zum Teil durch Anmerkungen zu ergänzen ist. Die Darstellung ist offensichtlich durch das ERM inspiriert (vgl. Abb. 15).

Teile eines Objektmodells können zu sogenannten "modules" zusammengefaßt werden. Durch die Definition überschaubarer Module können Implementierung und Pflege erleichtert werden. Dazu sollte eine Klassen möglichst exklusiv einem Modul zugeordnet sein. Auf diese Weise können die Module mit Hilfe übersichtlicher Schnittstellen miteinander verknüpft werden, während die Pflege einzelner Module unabhängig voneinander stattfinden kann. Wenn eine Klassenbezeichnung dennoch in mehreren Modulen verwendet wird, sollte sie jeweils die gleiche Bedeutung repräsentieren.

3.3.2 Das dynamische Modell

Im Objektmodell werden die Dienste der einzelnen Klassen genannt, es wird allerdings in keiner Weise beschrieben, unter welchen Bedingungen und in welcher zeitlichen Reihenfolge sie auszuführen sind. Das dynamische Modell dient dazu, solche Kontrollstrukturen abzubilden. Anders formuliert: Während das Objektmodell darauf gerichtet ist, die zulässigen Zustände der Objekte einzelner Klassen zu beschreiben, zielt das dynamische Modell darauf, Bedingungen für zulässige Systemänderungen darzustellen.

Ein dynamisches Modell setzt sich aus *Zustandsdiagrammen* ("state diagrams") für alle Klassen mit hinreichend komplexem dynamischen Verhalten zusammen. Ein Zustandsdiagramm wird als ein Netz von Ereignissen und Zuständen (und damit implizit: Zustandsübergängen) dargestellt.

Ereignisse können zu Ereignisklassen zusammengefaßt werden. Sie werden dann durch Attribute gekennzeichnet und können Eigenschaften von Oberklassen erben. Es können Bedingungen eingeführt werden, unter denen ein Ereignis zu einer Zustandsänderung führt. Beispiel: Das Ereignis "die Heizung wird eingeschaltet" führe nur unter der Bedingung "wenn die Temperatur weniger als 20 Grad beträgt" zu dem Zustand "die Heizung arbeitet". Auf diese Weise wird eine kombinatorische Explosion des Netzes verhindert und es ergibt sich eine gegenüber Petri-Netzen übersichtlichere Darstellung. Darüber hinaus - und darin ist der wesentliche Unterschied zu Petri-Netzen zu sehen - können in einem Zustandsdiagramm *Aktivitäten* und *Aktionen* abgebildet werden. Eine Aktivität ist einem

Zustand zugeordnet. Um die Integration mit dem Objektmodell zu fördern, ist es sinnvoll, eine Aktivität so zu wählen, daß sie einer Operation der jeweiligen Klasse entspricht. Die Beispiele, die von Rumbaugh et al. genannt werden, entstammen alle einem technischen Kontext. So wird dem Zustand "ringing" eines Telefons die Aktivität "ring bell" zugeordnet. Eine Aktivität beginnt mit dem zugehörigen Zustand und terminiert entweder während der Zustand fortdauert oder mit dem Zustand.

Um die Darstellung von Zustandsdiagrammen übersichtlich zu halten, können sie auf verschiedenen Detaillierungsstufen abgebildet werden. Dazu kann über mehrere Zustände generalisiert werden (Beispiel: "Dokument wird bearbeitet"), um dann bei Bedarf durch eine Dekomposition tieferen Einblick zu erhalten ("nested diagrams").

Im Unterschied zu einer Aktivität hat eine Aktion keine nennenswerte zeitliche Dauer. Beispiele für Aktionen sind das Öffnen eines Fensters auf dem Bildschirm oder die Präsentation eines Pop-Up-Menüs. Aktionen werden durch Ereignisse ausgelöst. Sie geben letztlich nähere Auskunft darüber, wodurch eine Zustandsänderung bewirkt wird.

Die Integration von Zustandsdiagrammen zu einem dynamischen Modell erfolgt über Ereignisse, die in mehreren Zustandsdiagrammen vorkommen ("shared events"). Dabei wird davon ausgegangen, daß alle Objekte nebenläufig sind, ihre Zustandsänderungen also autonom erfolgen können. Es ist zudem möglich, die Operationen eines Objekts als nebenläufig zu modellieren. Abbildung 16 zeigt die grafische Darstellung wichtiger Aspekte von Zustandsdiagrammen. Ähnlich wie Objektmodelle zum Zweck der Veranschaulichung als Instanzdiagramme präsentiert werden können, können dynamische Modelle als sogenannte "scenarios" das Verhalten von in bestimmter Weise initialisierten Objekten beschreiben.

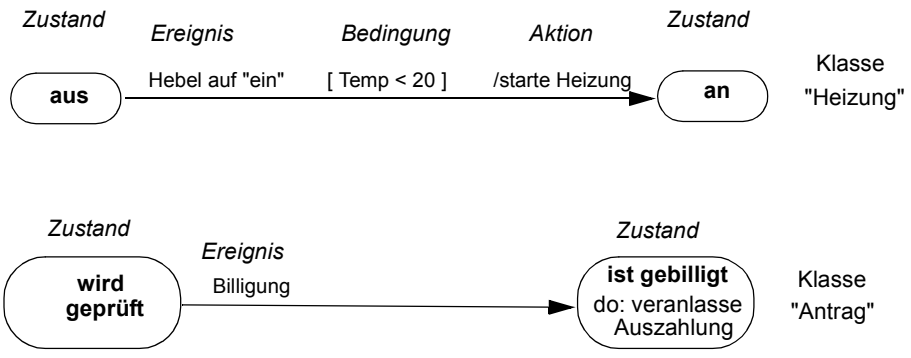


Abb. 16: Beispiele für in Zustandsdiagrammen verfügbare Konzepte

3.3.3 Das funktionale Modell

Im Objektmodell werden die Operationen einer Klasse nur durch einen Namen und gegebenenfalls durch eine Liste von Parametern gekennzeichnet. Im dynamischen Modell werden Operationen neben der namentlichen Kennzeichnung lediglich durch zugeordnete Zustände oder Ereignisse charakterisiert. Bei einfachen Operationen - wie beispielsweise dem Zugriff auf Attributwerte - ist das hinreichend, bei komplexeren Operationen mag eine eingehendere Beschreibung sinnvoll sein. Dazu ist in OMT das sogenannte "*functional model*" vorgesehen. Im funktionalen Modell wird eine Operation (oder ein Constraint des Objektmodells) nicht algorithmisch beschrieben. Vielmehr wird festgelegt, welche Eingabedaten eine Operation benötigt und welche möglichen Resultate sie produziert. Um eine detailliertere Beschreibung zu erhalten, kann eine Operation durch Dekomposition in mehrere Teiloperationen zerlegt werden.

Das funktionale Modell wird mit Hilfe von Datenflußdiagrammen realisiert. Sie unterscheiden sich von konventionellen Datenflußdiagrammen lediglich dadurch, daß die Schnittstellen bzw. Interfaces "*actor objects*" genannt werden und daß Ablagen "*data store objects*" heißen. Prozesse können in bekannter Weise zerlegt werden. OMT beinhaltet allerdings keine Methode zur Spezifikation von Prozessen. Es werden lediglich mögliche Ansätze (wie Pseudo-Code, Pre- und Postconditions, Entscheidungstabellen und dergleichen genannt).

Dabei fällt - wie auch im Objektmodell - auf, daß Rumbaugh et al. Objektorientierung nicht mit größter Konsequenz verwenden. Analog dazu, daß die Attribute von Objekten selbst keine Objekte sein dürfen, wird der Informationsfluß zwischen Objekten auf Daten reduziert. Dies ist eine erhebliche Einschränkung der expressiven Möglichkeiten des Modells.

Um ein Objektmodell mit einem funktionalen und einem dynamischen Modell zu verbinden, ist es zunächst erforderlich, daß die drei Teilmodelle nicht unabhängig voneinander entworfen werden. Vielmehr sollte von Anbeginn darauf geachtet werden, für korrespondierende Elemente gleiche Namen zu verwenden. Die Prozesse in Datenflußdiagrammen entsprechen - bei koordiniertem Entwurf - den Operationen einer Klasse im Objektmodell. Ein Datenfluß kann mit einer Menge von Attributen - von einer oder von mehreren Klassen - korrespondieren. Aktivitäten und Aktionen des dynamischen Modells können ebenfalls als Operationen von Klassen im Objektmodell beschrieben sein.

Abbildung 17 zeigt, daß die in den drei Teilmodellen beschriebene Semantik auf einzelne Klassen bezogen werden kann, so daß grundsätzlich eine Zusammenfassung bzw. Verkapselung der drei Aspekte möglich ist.

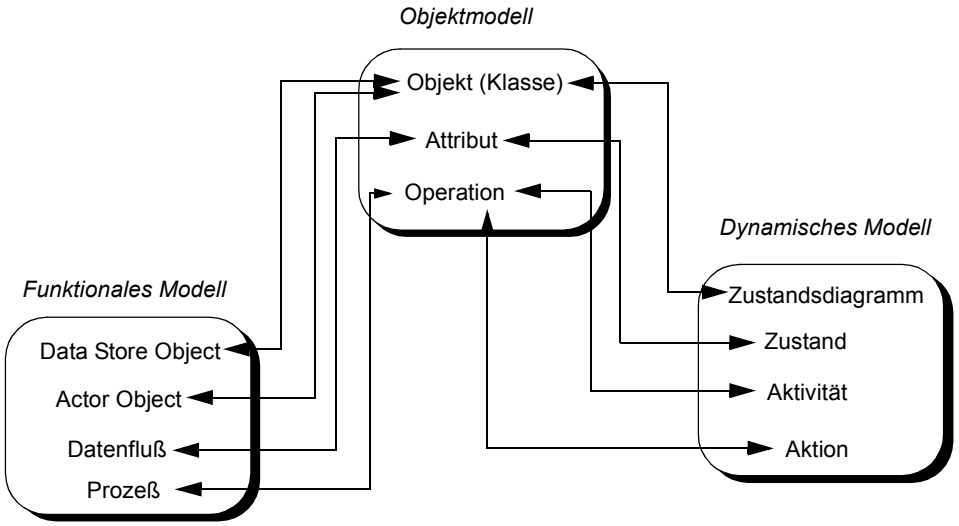


Abb. 17: Beziehungen zwischen den drei Teilmodellen der OMT

3.4 Die Methode von Booch

Booch gehört als früher Protagonist der Ada-Szene zu den Pionieren objektorientierter Modellierung. Nach einer Reihe vorbereitender Publikationen in den achtziger Jahren[1] hat er seine Entwurfsmethode in einem umfangreichen Werk (Booch 1990) präsentiert. Auch wenn Booch die Semantik von Objekten in ähnlicher Weise differenziert wie Rumbaugh et al., nimmt er keine Trennung in drei Teilmodelle vor. Seine Methode konzentriert sich vielmehr auf die Beschreibung von Objekten bzw. Klassen. Für einzelne Aspekte dieser Beschreibungen werden dann weitere Darstellungsformen eingeführt. Über den konzeptuellen Entwurf ("logical view") hinaus beinhaltet die Methode auch Techniken, die stärker auf die Implementierung ausgerichtet sind ("module architecture", "process architecture"). Wir richten im folgenden den Fokus vor allem auf die logische Sicht. Die Seitenangaben beziehen sich allesamt auf Booch (1990).

3.4.1 Klassen- und Objektdiagramme

Anders als im Objektmodell des OMT unterscheidet Booch in der grafischen Darstellung zwischen Klassen- und Objektdiagrammen. Diese Unterscheidung ist nicht zu verwechseln mit der zwischen Klassen und Instanzen: Auch die Elemente in Objektdiagrammen sind Abstraktionen realer Objekte. Die feinere und

1. Vgl. Booch (1982), Booch (1986 a), Booch (1986 b)

letztlich konsistentere Begriffsdifferenzierung bei Booch reflektiert vielmehr unterschiedliche Verwendungskontexte. Sie werden deutlich, wenn man die verschiedenen Beziehungen betrachtet, die zwischen Klassen bzw. Objekten bestehen können.

Zwischen Klassen können nach Booch vier Arten von Beziehungen unterschieden werden (S. 97 ff.). "*Inheritance*" bzw. Generalisierung ist sowohl für Einfach- als auch für Mehrfachvererbung darstellbar. Vererbung drückt aus, daß eine Klasse die Eigenschaften ihrer Oberklasse nutzen kann. Darüber hinaus gibt es Nutzungsbeziehungen ("*using relationships*") zwischen Klassen. Wenn zur Beschreibung einer Klasse auf eine andere Klasse zurückgegriffen wird - entweder zur Deklaration von Parametern in den Schnittstellen der Klasse oder zur Implementierung der Klasse. Dabei ist vor allem an die Instanzierung eines Objekts aus einer anderen Klasse, das allein lokal in einer Operation benötigt wird zu denken - also beispielsweise der Zugriff auf die aktuelle Zeit als Instanz einer Klasse "Zeit". Eine Nutzungsbeziehung kann durch die Angabe von Kardinalitäten angereichert werden. Es liegt auf der Hand, daß die Verwaltung solcher Beziehungen von großer Wichtigkeit für die konsistente Pflege eines Objektmodells ist. Darüber hinaus ist zu berücksichtigen - auch wenn Booch darauf nicht explizit hinweist, daß die Attribute (oder - in der Diktion von Booch - "fields") einer Klasse ebenfalls Nutzungsbeziehungen zu Klassen etablieren, denn sie sind durch Klassen deklariert, nicht - wie in OMT - durch Datentypen.

Es gibt Klassen, deren Objekte als "Container" für Objekte dienen. Dazu gehören Vektoren, Mengen oder auch Verzeichnisse[1] wie sie in Smalltalk verfügbar sind. Im Hinblick auf die Integrität eines Informationssystems ist es wünschenswert, die Klassen der Objekte, die einem Container zugeordnet werden dürfen, soweit wie möglich einzuschränken. Um diese Beziehung zwischen einer Container-Klasse und den Klassen der zugehörigen Objekte ausdrücken zu können, führt Booch die sogenannte "*instantiation relationship*" ein: Die Initialisierung einer Container-Klasse impliziert die Instanzierung von Objekten der so assoziierten Klasse(n). Zur weiteren Einschränkung dieser Beziehung können Kardinalitäten verwendet werden. Wenn für eine Klasse eine Metaklasse vorgesehen ist (die Klasse also selbst als Objekt betrachtet wird), besteht zwischen beiden eine "*metaclass relationship*". Eine solche Beziehung bedeutet, daß die Metaklasse über Operationen zur Instanzierung und Initialisierung der Klasse verfügt. Anders formuliert: Die Semantik einer Klasse, für die eine Metaklasse definiert ist, ist in gewissem Umfang variabel.

Klassen und die zwischen ihnen bestehenden Beziehungen können mit Hilfe von Klassendiagrammen grafisch visualisiert werden. Dazu werden Klassen mit Hilfe

1. Vgl. dazu die Darstellung in V.2.1.1.

von wolkenartigen, gestrichelten Symbolen ("amorphous blob") abgebildet. Für die Darstellung der Beziehungsarten sind verschieden Pfeile vorgesehen (vgl. Abb. 18), an die gegebenenfalls Kardinalitäten angetragen werden können.

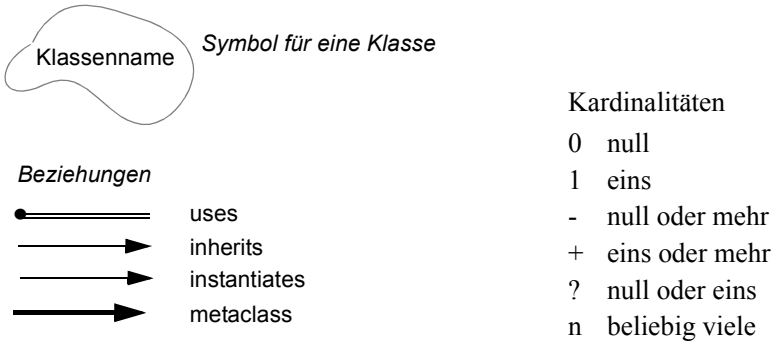


Abb. 18: Elemente zur Darstellung von Klassendiagrammen (S. 159)

Beziehungen zwischen Klassen beschreiben ein Verhältnis zwischen Konzepten. Booch unterscheidet davon Beziehungen, die zwischen einzelnen Objekten bestehen. Anders als in den Instanz-Modellen der OMT werden keine konkreten Instanzen verwendet. Objekte stellen vielmehr Platzhalter für beliebige Instanzen einer bestimmten Klasse dar. Booch unterscheidet auf dieser Ebene zwei Arten von Beziehungen (S. 88 ff.): Aggregation ("*containing relationship*") und Interaktion ("*using relationship*"). Eine Interaktionsbeziehung zwischen zwei Objekten bedeutet lediglich, daß beide Objekte Nachrichten austauschen können. Dabei unterscheidet Booch drei Rollen, die die beteiligten Objekte wahrnehmen können. Ein "*Actor*" bezeichnet ein Objekt, das Dienste eines anderen Objekts in Anspruch nimmt, selbst aber keine Dienste anbietet. Ein "*Server*" ist demgegenüber ein Objekt, das Dienste anbietet, sie aber nicht nachfragt. Ein "*Agent*" schließlich kann gleichzeitig die beiden anderen Rollen wahrnehmen. Aggregation ist eine Spezialisierung der Interaktionsbeziehung, die durch das Hinzufügen von Transitivität und Anti-Symmetrie entsteht.

Um den Nachrichtenaustausch zwischen Objekten zu modellieren, gibt Booch die Differenzierung in Interaktion und Aggregation wieder auf und betrachtet nur noch den generellen Fall: "A relationship between two objects simply means that the objects can send messages to one another." (S. 170) Die grafische Darstellung solcher Beziehungen bezeichnet Booch als *Objektdiagramme*. Objekte werden darin wie Klassen in Klassendiagrammen dargestellt, allerdings mit durchgezogener Linie. An die Verbindungslinien werden - durch eine Richtung ergänzt - Nachrichtennamen angetragen. Die Nachrichtennamen entsprechen den Namen

der jeweils angeforderten Operation. Da an anderer Stelle die Parameter einer Operation definiert werden, wird auf diese Weise implizit der mögliche Informationsfluß zwischen Objekten beschrieben. Objektdiagramme sind also in ihrer Funktion vergleichbar mit Datenflußdiagrammen in OMT. Im Unterschied zu Datenflußdiagrammen sind sie allerdings deutlich daran orientiert, wie Kommunikation zwischen Objekten tatsächlich stattfindet: durch das Versenden von Nachrichten, mit den Dienste aufgerufen werden. Auf diese Weise stellt sich die Integration mit anderen Teilen des Modells als unproblematisch dar. Im Objektdiagramm sind durch die Objekte die zugehörigen Klassen benannt, die Nachrichten beziehen sich unmittelbar auf Operationen dieser Klassen.

Booch sieht zudem vor, Objektdiagramme mit weiteren Angaben auszustatten, die eher der Implementierung zuzurechnen sind. So kann eine Nachricht durch eine von fünf Synchronisationsformen ("simple", "synchronous", "balking", "timeout" und "asynchronous") gekennzeichnet werden (S. 126 f.). Zudem kann - neben anderem - angegeben werden, ob eine Nachricht periodisch oder aperiodisch gesendet wird. Diese Angaben sind gegebenenfalls auf Kompatibilität mit entsprechenden Angaben bei der Klassenbeschreibung zu überprüfen.

3.4.2 Die Konzeptualisierung von Klassen

Die in den Diagrammen dargestellten Klassen und Objekte können sehr viel detaillierter beschrieben werden als in der OMT. Zu diesem Zweck führt Booch ein "class template" (S. 164) mit folgendem Aufbau ein:

Name:	identifier
Documentation:	text
Visibility:	exported | private | imported
Cardinality:	0 | 1 | n
Hierarchy	
Superclasses:	list of class names
Metaclass:	class name
Generic parameters:	list of parameters
Interface | Implementation (Public/Protected/Private):	
Uses:	list of class names
Fields:	list of field declarations
Operations:	list of operation declarations
Finite state machine:	state transition diagram
Concurrency:	sequential | blocking | active
Space Complexity:	text
Persistence:	static | dynamic

Um die Komplexität großer Modelle zu reduzieren, kann eine Menge von Klassen zu Kategorien zusammengefaßt werden. Durch die Eigenschaft *Visibility* wird festgelegt, ob die Klasse allein innerhalb dieser Kategorie verfügbar sein

soll, exportiert werden darf oder ob sie aus einer anderen Kategorie importiert ist. Man fragt sich allerdings, wieso in dem Template die Angabe der jeweiligen Kategorie nicht vorgesehen ist. Die *Kardinalität* einer Klasse gibt an, wie viele Instanzen erzeugt werden dürfen (0: es muß keine Instanz existieren; 1: es muß genau eine Instanz existieren; n: es dürfen beliebig viele Instanzen existieren). Die Zahl der zugeordneten Oberklassen hängt davon ab, ob für eine konkrete Modellierung bzw. Implementierung Einfach- oder Mehrfachvererbung unterstellt wird. Wenn die intendierte Implementierungssprache es erlaubt, kann - wie auch in OMT - optional eine Metaklasse angegeben werden. "Generic Parameters" sind - neben anderen - ein Beleg dafür, daß Booch stark von seinen Arbeiten an Ada geprägt ist. Generische Parameter sind variable Teile einer generischen Klasse (wie sie in Ada oder Eiffel möglich sind). Um aus einer generischen Klasse eine verwendbare Klasse zu machen, müssen die Parameter instanziert werden (beispielsweise mit Operationen).

Die Klassen, die von der beschriebenen Klasse benutzt werden, können als Liste eingefügt werden. Danach wird die Liste der Attribute (von Booch "fields" genannt"), die ein Objekt der beschriebenen Klasse aufweist, erfaßt. Im Unterschied zur Klasse selbst sieht Booch für Attribute nicht die Angabe von Kardinalitäten vor. Desweiteren können Operationen eingetragen werden. Attribute, Operationen und Beziehungen können in drei verschiedenen Formen verfügbar gemacht werden. "*Private*" heißt, daß auf sie von außen nicht zugegriffen werden kann, "*protected*" bedeuted, daß es Schutzmechanismen gibt und "*public*" schließlich heißt, daß sie allgemein verfügbar sind. Die Auflistung assoziierter Klassen ist allerdings redundant, da diese Zuordnung bereits in der grafischen Darstellung sichtbar gemacht wird. Zudem ist die Unterscheidung zwischen Schnittstelle und Implementierung m.E. verwirrend und ohnehin überflüssig: Es werden hier - wie auch an anderer Stelle - Implementierungsentscheidungen vorweggenommen, ohne daß der Klasse auf diese Weise zusätzliche Semantik zugefügt wird. Booch weist denn auch selbst darauf hin, daß die Angaben zur Implementierung zunächst offen gelassen werden sollten (S. 165). Attribute werden durch ihre Klasse (also nicht wie in OMT durch einen Datentyp) beschrieben. Sie können unterschieden werden in solche, die allein der Klasse (wenn man sie ebenfalls als Objekt betrachtet) zur Verfügung stehen (Klassenvariablen), und solche, die an die Instanzen weitergereicht werden (Instanzvariablen). Für Operationen wird diese Unterscheidung nicht (explizit) getroffen.

Durch die Zuordnung eines Zustand/Ereignis-Diagramms kann das dynamische Verhalten der Instanzen der Klasse beschrieben werden. Der mit "*concurrency*" umschriebene Eintrag bezieht sich darauf, in welcher Weise die Kommunikation mit anderen Objekten synchronisiert werden kann. Sequentialisierten Prozessen ist die Synchronisation der zwischen Objekten versendeten Nachrichten gleich-

sam inhärent. Wenn Parallelverarbeitung möglich ist, erlaubt Booch die Unterscheidung von zwei Fällen. "Blocking objects" sind Objekte, die passiv sind, das heißt nur dann eine Operation ausführen, wenn ihnen eine entsprechende Nachricht geschickt wird. Demgegenüber haben "active objects" ein autonomes Verhalten. Letztlich verbirgt sich hinter dieser Unterscheidung, daß die Operationen passiver Objekte sequentialisiert sind (wobei mehrere Objekte parallel eine Operation ausführen können), während die Operationen eines aktiven Objekts parallel ausgeführt werden können.

Wieder im Hinblick auf die Implementierung können unter "*space complexity*" Angaben über den Speicherbedarf eines Objekts der Klasse abgegeben werden. Für den Entwurf sind solche Angaben m.E. nicht eben hilfreich, zumal sie ohnehin weitgehend automatisch ermittelt werden können (mit der Einschränkung, daß sie völlig unabhängig von der jeweiligen Implementierungsebene sind). Die *Persistenz* eines Objekts kann entweder als dynamisch oder statisch gekennzeichnet werden - je nachdem, ob seine Lebensdauer an die des jeweiligen Programms gebunden ist oder nicht. Dabei ist einschränkend darauf hinzuweisen, daß eine solche Auszeichnung durchaus eine Implementierungsentscheidung sein kann.

Neben Klassen können auch sogenannte "class utilities" erfaßt werden. Sie sind eine Konzession an entsprechende Konstrukte in einigen Programmiersprachen wie Ada und C++ ("free subprograms"). Wir müssen hier nicht weiter auf sie eingehen, da sie dem objektorientierten Paradigma im engeren Sinne nicht entsprechen.

Operationen können mit Hilfe eines weiteren dedizierten Templates im Detail beschrieben werden (S. 166):

Name:	identifier
Documentation:	text
Category:	text
Qualification:	text
Formal Parameters:	list of parameter declarations
Result:	class name
Preconditions:	PDL / object diagram
Actions:	PDL / object diagram
Postconditions:	PDL / object diagram
Exceptions:	list of exception declarations
Concurrency:	sequential | guarded | concurrent | multiple
Time Complexity:	text
Space complexity:	text

In Anlehnung an die Gruppierung der Methoden einer Klasse in Smalltalk sieht Booch die Möglichkeit vor, eine Operation einer *Kategorie* zuzuordnen. "*Qualification*" ist eine Konzession an Programmiersprachen (wie etwa CLOS), die

verschiedene Arten von Operationen (wie ":before", ":after", ":around") zulassen. Die Liste der Parameter-Deklarationen besteht aus den Klassen der Parameter, die der Operation beim Aufruf des zugehörigen Dienstes zu übergeben sind. Das *Resultat* der Operation - hier wird wohl implizit unterstellt, daß es sich bei einer Operation immer um eine Funktion handelt - wird durch die Klasse des zurückgereichten Wertes beschrieben. Zur Beschreibung der Ablauflogik einer Operation bietet Booch verschiedene Darstellungsformen an. Sie können einzeln oder einander ergänzend verwendet werden. Während *Pre-* und *Postconditions* lediglich die Eingangs- bzw. Ausgangsbedingungen festlegen, steht "*actions*" für den eigentlichen Ablauf der Operation. Dabei kann jeweils entweder auf eine *Programming Description Language* (PDL), also Pseudocode, oder auf ein *Objektdiagramm* (vgl. III.3.4.1) zurückgegriffen werden. Dabei läßt Booch offen, wie detailliert solche Beschreibungen sein sollten: "... an appropriate level of detail for each operation should be chosen, given the needs of the eventual reviewer." (S. 167) Ausnahmen, die während der Ausführung der Operation auftreten können (ohne innerhalb der Operation kontrollierbar zu sein), werden jeweils durch eine Ausnahmeart, die für das gesamte Modell einheitlich definiert sein sollte, deklariert - also beispielsweise: "Drucker defekt" oder "Lesefehler beim Zugriff auf einen peripheren Speicher". Damit wird die Voraussetzung für eine einheitliche Ausnahmebehandlung geschaffen.

Desweiteren kann festgelegt werden, ob die Operation zu einer Zeit immer nur eine Anfrage bearbeiten kann, oder ob mehrere Anfragen nebenläufig möglich sind - in verschiedener Weise synchronisiert. Diese auf eine Operation bezogenen Angaben müssen mit den in der Klassenbeschreibung gemachten Angaben über Nebenläufigkeit kompatibel sein. "Time complexity" und "space complexity" bezeichnen Angaben über die zeitliche Dauer (eventuell differenziert nach unterschiedlichen Eingangsdaten) und den Speicherbedarf.

3.4.3 Dynamische Aspekte

Auch wenn die Beschreibung von Operationen sowie die des Nachrichtenaustausches sehr detailliert erfolgen kann, so wird dadurch noch nichts darüber ausgesagt, in welcher zeitlichen Folge und unter welchen Bedingungen einzelne Operationen ausgeführt werden. Um diesen Aspekt abzudecken, verwendet Booch neben Zeitdiagrammen ("timing-diagrams") Zustand/Ereignis-Diagramme ("state transition diagrams").

Wie auch in der OMT sind Zustand/Ereignis-Diagramme dafür vorgesehen, die möglichen Zustandsänderungen der Objekte *einer* Klasse zu beschreiben. Die Zustände sollten sich möglichst auf bestimmte Ausprägungen der Attribute (bzw. "fields") eines Objekts beziehen. Die Zustandsübergänge werden durch ein auslösendes Ereignis und eine Transition beschrieben. Dabei korrespondiert die

Transition mit einer der Operationen der betrachteten Klasse.

Um das dynamische Verhalten eines Informationssystems oder wesentlicher Teile eines solchen zu beschreiben, reicht es gewöhnlich nicht hin, allein die Objekte einer Klasse zu betrachten. Vielmehr ist es erforderlich, das unter Umständen interdependente Verhalten der Objekte mehrerer Klassen zu berücksichtigen. Die von Booch vorgeschlagenen Zeitdiagramme stellen die zeitliche Abfolge der Ausführung von Operationen mehrerer Objekte einander gegenüber (Abb. 19).

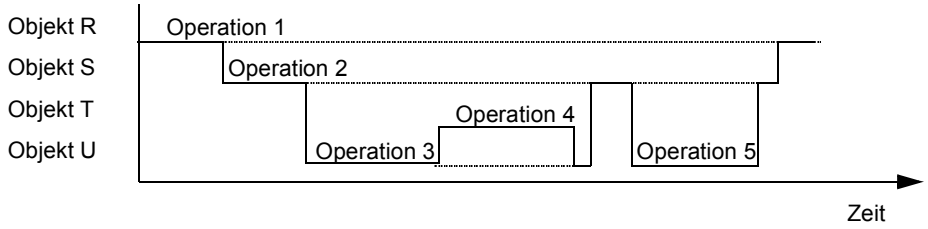


Abb. 19: Beispiel für ein "Timing Diagram" - nach Booch (1990, S. 174)

Ein solches Diagramm, wie es auch zur Veranschaulichung der Zuordnung von Prozessorkapazität bei der Prozeßverwaltung in Betriebssystemen verwendet wird, beschreibt also, in welchem Umfang einzelne Operationen unabhängig voneinander ablaufen können. Ein Umstand, der für die Imlementierung vor allem dann bedeutsam wird, wenn die Möglichkeit besteht, Objekte Prozessoren zuzuordnen. Es enthält allerdings keine Informationen darüber, wodurch eine Operation ausgelöst wird. Es ist also eine Ergänzung zu den anderen Darstellungsformen. Die Integration ist wiederum unproblematisch. Sie erfolgt durch die Verwendung gemeinsamer Namen für die jeweils dargestellten Operationen.

3.5 Vergleichende Beurteilung

Sowohl OMT als auch die Methode von Booch sind geeignet, die wesentlichen Aspekte des Entwurfs von Informationssystemen abzudecken. Booch präsentiert dabei eine konsequentere Objektorientierung. Dies artikuliert sich einerseits in seiner Terminologie[1], andererseits darin, daß die Integration (und damit: Verkap-

1. Dabei ist allerdings die Defintion einer Klasse bei Booch (1990, S. 93) m.E. nicht überzeugend. Danach ist eine Klasse "a set of objects that share common structure and a common behavior". Eine solche Definition kann leicht mißverstanden werden (nämlich als Menge existierender Instanzen anstatt als Menge möglicher Instanzen). Darüber hinaus greift sie nicht mehr für abstrakte Klassen (also Klassen, die gerade dadurch gekennzeichnet sind, daß sie keine Instanzen haben können), die Booch explizit zuläßt.

selung) der Teilmodelle besser angeleitet wird als bei Rumbaugh et al.: Klassen und ihre Merkmale dienen als gemeinsame Referenzen für alle Beschreibungssichten. Dieser Vorteil wird jedoch dadurch eingeschränkt, daß Booch die Objektorientierung durch Konzessionen an Spezifika einzelner Programmiersprachen ("class utilities", "generic parameters") aufweicht.

Während Rumbaugh et al. an manchen Stellen eine größere Detaillierung vermissen lassen[1], wirkt die Methode von Booch mitunter überladen: Er beschränkt sich nicht nur auf den Entwurf, sondern bezieht zum Teil Implementierungsentscheidungen mit ein. Das mag daran liegen, daß Boochs Sichtweise deutlich durch den Entwurf und die Anwendung von Programmiersprachen geprägt ist. Grundsätzlich ist die frühzeitige Berücksichtigung von Details, die vielleicht erst während der Implementierung von Bedeutung sind, kein Fehler. Um die Anschaulichkeit des Entwurfs nicht zu gefährden, wäre es m.E. allerdings besser, dazu eine weitere Abstraktionsebene einzuführen. Die Unterscheidung zwischen Klassen und Objekten als Platzhalter für beliebige Instanzen ist im Hinblick auf die differenzierte Betrachtung von Beziehungen gewiß angemessen. Die von Booch vorgenommene Differenzierung von Beziehungen ist allerdings in Teilen verwirrend. So stellt sich beispielsweise die Frage nach dem Unterschied zwischen einer "using relationship" auf Klassenebene und einer "containing relationship" auf Objektebene.[2]

Der folgende Vergleich der Methoden greift auf das oben entwickelte Beurteilungsschema zurück. Er dient deshalb auch der Gegenüberstellung der objektorientierten Ansätze mit den zuvor beurteilten konventionellen Ansätze. Im Unterschied zur Datenmodellierung ist die Transformation in eine implementierungsnähere Repräsentationsform nicht expliziter Bestandteil der Methoden und kann deshalb nicht in die Beurteilung einfließen.

1. Hsieh (1992, S. 26) kommt hier, m.E. allerdings ohne überzeugende Begründung, zu einem anderen Urteil: "By far we feel that Rumbaugh's methodology is the most comprehensive and complete one ...".
2. Die Antwort lautet letztlich, daß der eine Fall ein wenig spezieller sein mag als der andere. Es bleiben allerdings Zweifel daran, ob eine solche Differenzierung für den Entwurf hilfreich ist.

Strukturelle Aspekte

Bewertung

			Rumbaugh et al.	Booch
Attribute	maschinen-orientiert	definierbar als Intervall	ja	ja
		durch funktionale Spezifikation	nein	nein
	anwendungs-orien-	durch Enumeration	ja	nein
		durch funktionale Spezifikation	nein	nein
		Entitätstyp	nein	ja
	Kardinalität	minimal = 0	nein	ja
		maximal > 1	nein	ja
	Zugriffsrecht		nein	ja
	Beziehungen		nein	nein
Identität einer Entität vom Zustand determiniert			nein	nein

Strukturelle Aspekte

Bewertung

				Rumbaugh et al.	Booch
Historie			generell	nein	nein
			partiell	nein	nein
Beziehungen zwischen Entitäten	Kardinalität		einfach	ja	-
		min-max	eingeschränkt	nein	ja
			frei	nein	nein
	Richtung		Rolle	ja	nein
			inverse Beziehung	nein	nein
	Referentielle Integrität			ja	ja
	Aggregation			ja	ja
	sonstige			ja	ja
Generalisierung	einfach			ja	ja
	mehrfach			ja	ja
Views/Design der Benutzerschnittstelle					
Spezifikation von Views auf einen Entitätstyp				ja	ja
Spezifikation von Views auf eine Menge von Entitätstypen				ja	ja
Spezifikation einer prototypischen Benutzerschnittstelle für einen Entitätstyp				nein	nein
dito, für eine Menge von Entitätstypen				nein	nein

Dynamische/funktionale Aspekte Bewertung

		Rumbaugh et al.	Booch
Operationen/Funktionen			
Schnittstellenspezifikation		ja	ja
Preconditions		nein	ja
Postconditions		nein	ja
Trigger		nein	nein
Ausnahmenbehandlung		nein	ja
Systemverhalten			
Zeitliche Aspekte/ Kontroll-strukturen	für Entität	ja	ja
	für Kollektionen	nein	mit Einschränkung
Sonstige Aspekte			
Modula-risierung	Kohärenz	ja	ja
	Assoziierung	-	-
Produktionsumgebung			
Datenbankmanage-mentsystem	DDL	nein	nein
	DML	nein	nein
	Standards	nein	nein
	Kontrolle über Ope-rationen	ja	ja
Werkzeugunterstützung		ja	ja

Zusammenfassend kann festgestellt werden, daß beide Methoden eine umfassende Modellierung auf einem hohen semantischen Niveau erlauben - mit gewissen Einschränkungen bei der OMT, da Attribute durch Datentypen - nicht durch Klassen - und der Informationsfluß zwischen Objekten mit Hilfe von Datenstrukturen beschrieben wird. Die Integration der Teilmodelle ist in beiden Methoden möglich, wird bei Booch jedoch wirksamer unterstützt. Im Hinblick auf die Implementierung sind beide Ansätze an einer Umsetzung mit Hilfe einer Programmiersprache orientiert, wobei Rumbaugh et al. auch Beispiele für die (fragmentarische) Transformation in ein relationales Schema geben. Für den Einsatz beider Methoden stehen einige Werkzeuge zur Verfügung. Während General Electric mit "OMT-Tool" einen speziellen Entwurfseditor anbietet und Booch ein dediziertes Werkzeug ("Rational Rose") zur Unterstützung seiner Methode verkauft, ist eine Reihe von Entwurfsumgebungen am Markt, die wahlweise die Notation von Booch oder die von OMT zulassen.

Der Entwurf von Benutzerschnittstellen wird lediglich auf einem technischen Niveau unterstützt, also beispielsweise durch den Einsatz von Zustand/Ereignis-Diagrammen. Vor dem Hintergrund der zunehmenden Verbreitung von Bibliotheken zur Gestaltung grafischer Benutzerschnittstellen erscheint ein solches Niveau nicht immer angemessen. Für den Entwurf könnte vielmehr auf vorhandene Konstrukte (Windows, Widgets etc.) zurückgegriffen werden. Dazu sollte ein Modellierungsansatz einerseits eine geeignete Auswahl solcher Konstrukte anbieten, andererseits deren Integration mit anderen Modellebenen unterstützen. Die Analyse organisatorischer Alternativen und deren betriebswirtschaftliche Bewertung liegt außerhalb des Blickwinkels beider Ansätze.

Beide Methoden bieten gegenüber konventionellen Ansätzen der Datenmodellierung und deren Erweiterungen deutliche softwaretechnische Vorteile. Dennoch dürfte ihr Nutzen für viele Entwickler zweifelhaft sein. Beide Methoden, besonders die von Booch, sind sehr komplex. Ihre Rezeption ist trotz einer gelungenen Darstellung mit einem erheblichen Aufwand verbunden. Schließlich ist einschränkend zu berücksichtigen, daß die grafische Darstellung der funktionalen/ dynamischen Teilmodelle für viele Anwender - deren Kommentare ja auch während des Entwurfs wichtig sein können - nicht eben anschaulich sind.

4. Unternehmensmodelle und Informationssystem-Architekturen

Die Verwendung objektorientierter Ansätze für die Modellierung von Informationssystemen erleichtert die Etablierung eines hohen Integrationsniveaus erheblich: Als gemeinsame Referenzen für die Kommunikation zwischen Teilen eines Informationssystems dienen nicht allein wenige Datentypen und aus ihnen zusammengefügte Strukturen, sondern Klassen, die ein hohes Maß an Anwen-

dungssemantik beinhalten können. Generalisierung und Spezialisierung sind zudem wirksame Mittel, um die Wiederverwendung vorhandener Software zu unterstützen. Dennoch ist der Verweis auf Methoden für den objektorientierten Enwurf allein kaum hinreichend, um der zu Beginn unserer Untersuchung entfalteten Vision von Integration und Wiederverwendbarkeit näher zu kommen: Die für den Entwickler verbleibenden (und zum Teil völlig neuen) Anforderungen sind so komplex, daß sie die Verheißungen von Wiederverwendbarkeit zu konterkarieren drohen. Darüber hinaus leisten die Methoden eo ipse keinen Beitrag zur unternehmensübergreifenden Integration: Selbst wenn innerhalb eines Unternehmens der Entwurf und die Implementierung eines hochintegrierten Informationssystems gelingt, ist damit noch kein gemeinsames Referenzssystem für mehrere Unternehmen geschaffen. Isoliert betriebene Entwicklungen lassen kaum auf brauchbare Gemeinsamkeiten hoffen, denn "... the design process is not deterministic: different designers can produce different enterprise models of the same enterprise." (Hawryszkiewycz 1991, S. 115).

4.1 Überblick

Um Gemeinsamkeiten innerhalb eines Unternehmens und auch zwischen Unternehmen darzustellen, ist letztlich eine Gesamtsicht des Unternehmens anzustreben. Das Bemühen um eine solche Perspektive hat in der Betriebswirtschaftslehre eine lange Tradition und ist gleichsam konstituierend für die betriebswirtschaftliche Organisationstheorie.

Es war denn auch ein Forschungsvorhaben im Rahmen der Organisationtheorie, in dem bereits zu Beginn der siebziger Jahre ein an den Erfordernissen der Datenverarbeitung orientiertes Gesamtmodell von Industrieunternehmen entwikkelt wurde. Das Kölner Integrationsmodell (KIM) zielte darauf, DV-Organisatoren und Entwicklern eine Grundlage für die Systemgestaltung und -pflege zu liefern. Dazu wurde angestrebt, eine *vollständige* Liste aller *automatisierbaren Aufgaben* sowie den zwischen ihnen auszutauschenden *Daten* zu erstellen.[1] Die Arbeiten führten zu einem umfangreichen Modell (Grochla et al. 1974). In einem ersten Schritt wurden ungefähr 400 Aufgaben erfaßt. Jede Aufgabe wurde mit einem Schlüssel und einer Benennung versehen und durch eine kurze, stichwortartige Beschreibung charakterisiert. In einem weiteren Schritt wurde eine Liste der Datenströme erstellt, die in die Aufgaben eingehen bzw. von ihnen produziert werden. Diese Liste umfaßt fast 1500 sogenannte Kanäle, die in natürlichsprachlicher Form als Menge von Feldtypen beschrieben und ebenfalls durch einen eindeutigen Schlüssel gekennzeichnet sind. Die Zuordnung von Kanalinhalten zu Aufgaben wird in einem komplexen grafischen Schaubild dargestellt.

1. Eine ausführliche Darstellung der methodischen Grundlagen findet sich in den Beiträgen in Grochla et al. (1974).

Die Fülle der im KIM verzeichneten Aufgaben und Daten wird wohl von keiner anderen publizierten Dokumentation übertroffen. Aus heutiger Sicht sind die Schwächen des Ansatzes allerdings nicht zu übersehen. Dabei ist einerseits daran zu denken, daß beim Entwurf des KIM wichtige Anforderungen an die Datenmodellierung nicht erfüllt wurden. So ist die Darstellung der Kanäle durch ein hohes Maß an Redundanz gekennzeichnet. Die Beschreibung der Felder erfolgt natürlichsprachlich, es wird nicht auf eine Menge von Datentypen zurückgegriffen. Angesichts der vielfältigen Beziehungen zwischen Kanälen - hier ist vor allem an Generalisierung und Aggregation zu denken - ist die im KIM betriebene Modellbildung durch vollständige Enumeration unbefriedigend. Zudem ist die Beschreibung der Aufgaben teilweise vage und mehrdeutig. Das KIM bleibt allerdings - vor allem, wenn man an die Entstehungszeit denkt - ein beeindruckendes Dokument einer unternehmensweiten *Anforderungsanalyse*.

In jüngerer Zeit ist eine Reihe unterschiedlich motivierter Ansätze zu einer unternehmensweiten Betrachtung von Informationssystemen zu verzeichnen. So werden in einer wachsenden Zahl von Arbeiten die Vorteile einer unternehmensweiten Datenmodellierung herausgestellt und Ansätze zur schrittweisen Entwicklung solcher Modelle präsentiert.[1] Damit zusammenhängend werden Architekturen vorgeschlagen, die die softwaretechnische Einbindung solcher Modelle in Informationssysteme beschreiben.[2] Mit einem Blickwinkel, der stärker die Verwendung von Information betont, sind Ansätze entstanden, die ein unternehmensweit geplantes und koordiniertes Informationsmanagement beschreiben (Parker/Benson 1985; Marchand/Horton 1986; Österle 1989). Sie behandeln vor allem Aspekte der organisatorischen Verankerung eines unternehmensweiten Informationsmanagements. Daneben gibt es Arbeiten, die darauf gerichtet sind, die Zusammenhänge zwischen einer an der Verwendung orientierten Sicht auf Informationen und deren technischer Verwaltung deutlich zu machen, um so die gängige isolierte Behandlung der beiden Sichten zu überwinden. Dazu werden mitunter Architekturen entworfen, die der Veranschaulichung dieser Zusammenhänge dienen (Krcmar 1990; Österle/Brenner/Hilber 1990; Hildebrand 1992).

Eine Reihe von Verfassern präsentiert Datenmodelle von Branchen oder auch von Funktionsbereichen - zum Teil in Form von Übersichtsartikeln (Nießen/Zwickler 1990; Thanheiser 1990; Wiborny 1992), zum Teil in ausführlicher und detaillierter Darstellung (Deßaules 1992). Das Esprit-Projekt ITHACA (Integrated Toolkit for Highly Advanced Computer Applications) basierte wesentlich auf

1. Im deutschsprachigen Raum ist hier besonders an die Arbeiten von Scheer und Vetter zu denken.

2. Dies gilt einerseits für die Datenbankforschung, andererseits für Entwürfe großer Anbieter, wie beispielsweise das vor allem mit IBM verknüpfte Repository-Konzept (vgl. dazu Sagawa 1990 oder Hofinger 1991).

der Annahme, daß für einzelne Branchen sogenannte "generic applications " existieren: "A 'generic application' factors out and packages the common aspects of a class of specific applications" (Pröfrock et al. 1989, S. 7). Ziel von ITHACA war es, dem Anwendungsentwickler sogenannte "generic application frames" zur Verfügung zu stellen. Der Inhalt eines solchen generischen Anwendungsrahmens wird allerdings in den vorliegenden Veröffentlichungen nicht im Detail beschrieben. Er wird allein durch einen wenig detaillierten Aufbau charakterisiert: Neben dem Code einer generischen Anwendung gehören dazu Entwurfsdokumente sowie Informationen darüber, "wie aus dem generischen Rahmen eine spezifische Anforderung geformt werden kann" (Ader et al. 1990, S. 13). Der wesentliche Teil der Forschungsaktivitäten war allerdings auf den Entwurf und die Implementierung einer objektorientierten Entwicklungsumgebung gerichtet. Die Arbeiten an branchenspezifischen Anwendungsrahmen beschränkten sich auf rudimentäre Anforderungsanalysen.[1]

Unter dem Etikett "Open Distributed Processing" (ODP) hat sich eine Gruppe von Forschungseinrichtungen zusammengefunden, um Standards für den Entwurf offener verteilter Systeme zu etablieren (Geihs 1990; Popien/Spaniol/De Meer 1991; Marques/Navarro/Sarmiento 1993). Dabei wird besonders betont, daß die Bedürfnisse der Anwender solcher Systeme - also vor allem von Unternehmen - im Vordergrund stehen sollen. Deshalb wird im ODP-Referenzmodell ein Unternehmensmodell ("enterprise viewpont) skizziert, das allerdings zur Zeit noch nicht über rudimentäre Beschreibungen hinausgeht. Von der ISO[2] werden unter dem Oberbegriff "Information Resources Dictionary System" eine Reihe von Standards angestrebt, die eine Basis für die Unternehmensmodellierung liefern sollen. Die einzelnen Vorschläge sind zur Zeit nur als "Draft" verfügbar. Auffällig ist, daß man nicht allein technische Probleme fokussiert, sondern explizit auch eine betriebswirtschaftliche und organisatorische Sicht betont:

> "An Information Resources Dictionary System (IRDS) provides the users with the capability of defining, maintaining and presenting a complete model of an enterprise, including both its business environment (i.e. its business functions, information, procedures etc.) and the information resources used to support the business environment. The enterprise model in an IRDS encompasses the strategic, tactical, and operational levels." (ISO/IEC 1992, S. 1)

Auch wenn der in ISO/IEC (1992) unterbreitete Vorschlag einen Umfang von fast 400 Seiten aufweist, beschränkt er sich doch im wesentlichen auf die Analyse von Repräsentationsformalismen, bewegt sich also allein auf der Ebene von Metamodellen (wie ERM, konzeptuelle Graphen etc.).

Auch große Anbieter versuchen mehr und mehr, Referenzmodelle zu entwerfen.

1. Eine solche Analyse für einen kleinen Funktionsbereich von Versicherungsunternehmen findet sich in Bruni et al. (1990).
2. unter Mitwirkung einer größeren Zahl renommierter Anbieter und Anwender.

Sie zielen einerseits darauf, grundlegende Modellierungskonzepte und die zwischen ihnen bestehenden Beziehungen zu beschreiben. Auf diese Weise wird ein Bezugsrahmen für den Entwurf von CASE-Werkzeugen und die Integration der mit ihnen entwickelten Systeme geschaffen. Von besonderer Bedeutung ist das mit dem von IBM propagierten Entwicklungs- und Wartungskonzept einhergehende "information model" (Devlin/Murphy 1988; Hazzah 1990). Andererseits gibt es eine Reihe von Referenzmodellen für einzelne Branchen. Während solche Modelle in der Regel zum wohlgehüteten Geheimnis von Softwarehäusern zählen, verkünden einige große Anbieter - mit Blick sowohl auf Software-Partner wie auch auf Kunden - die Bereitstellung von Branchenmodellen. So werden bei IBM unter anderem Referenzmodelle für Banken und Versicherungen entwickelt (IBM 1990, Barthelmess 1991). Für unsere Betrachtung ist dabei besonders bemerkenswert, daß solche Aktivitäten nicht allein auf Software-Entwürfe zielen. Vielmehr sollen die Modelle auch zu einem besseren Verständnis von Geschäftsabläufen beitragen. Zur Erreichung der Ziele soll eine *weltweite Anforderungsanalyse* durchgeführt werden. Um eine systematische Erfassung bzw. Entwicklung der Anwenderbedürfnisse zu ermöglichen, ist die Etablierung sogenannter "Costumer Advisory Boards" vorgesehen. Entwürfe und Implementierungen für einzelne Branchen sollen in eine einheitliche Architektur eingebettet werden (Mercurio et al. 1990).

Jacobson et al. (1992, S. 17 f.) betonen den gegenüber reiner Systementwicklung erweiterten Anspruch der Unternehmensmodellierung. Sie sprechen dabei von "enterprise development":

> "Enterprise development can be viewed as a generalization of system development. Instead of developing a system, a whole enterprise is developed. In enterprise development, the company is seen from several different perspectives. The aim is to identify the problem areas and suggest alternative solutions. One result may be to introduce an information system."

Ihre weiteren Ausführungen beschränken sich allerdings im wesentlichen auf objektorientierte Systemanalyse und -entwurf.

Wir werden im folgenden drei Ansätze zur ganzheitlichen Unternehmensmodellierung näher betrachten. Sie unterscheiden explizit - wenn auch mit unterschiedlichen Zielen - verschiedene Sichten auf das Unternehmen.

4.2 Scheer: Architektur integrierter Informationssysteme

Vor dem Hintergrund zunehmender Durchdringung der Unternehmen mit DV-Systemen und der damit zusammenhängenden wachsenden Bedeutung dieser Systeme für wirtschaftliche Unternehmensführung schlägt Scheer (1990 b) eine "EDV-orientierte Betriebswirtschaftslehre" vor. Als deren wesentlichen Gegenstand betrachtet er den Entwurf unternehmensweiter Informationssysteme. In

Scheer (1991) wird ein Bezugsrahmen für den Entwurf und die Einordnung von Anwendungsprogrammen in betriebliche Informationssysteme vorgestellt: "Die entwickelte ARIS-Architektur legt fest, wie Anwendungssysteme beschrieben werden." (Scheer 1991, S. 41).

4.2.1 Sichten auf das Unternehmen

Als Ausgangspunkt für die Beschreibung eines Unternehmens schlägt Scheer die Erfassung und Analyse von Vorgangsketten vor. Vorgangsketten sind wichtige Prozesse der betrieblichen Leistungserstellung, die aus mehreren Vorgängen zusammengesetzt sind. Als Beispiele werden "eine geschlossene Auftragsbearbeitung" oder "die Entwicklung eines Produktes" genannt (Scheer 1991, S. 4). Vorgangsketten werden auf drei Abstraktionsebenen betrachtet. Die erste Ebene beinhaltet konkrete Vorgangsketten (Instanzen). Auf der zweiten Ebene wird über gleichartige Vorgangsketten generalisiert und es entsteht ein Modell von Vorgangskettentypen. Auf der höchsten Abstraktionsstufe, der "Meta-Ebene", werden die gemeinsamen Merkmale aller Vorgangskettentypen beschrieben. Es handelt sich also gleichsam um ein generisches Vorgangskettenmodell. Zur systematischen Erfassung von Vorgangsketten schlägt Scheer "Vorgangskettendiagramme" vor (Scheer 1991, S. 58 f.).

Die "Architektur integrierter Informationssysteme" (ARIS) sieht vor, ein Vorgangskettenmodell (zunächst) der Ebene 2 durch zwei orthogonale Strukturierungsdimensionen in insgesamt neun Teilmodelle zu differenzieren. Dazu werden auf der einen Seite drei Konzeptualisierungsebenen unterschieden. Auf der Ebene des *Fachkonzepts* werden "die einzelnen Sichten des Anwendungssystems" (Scheer 1991, S. 16) modelliert. Scheer spricht in diesem Zusammenhang von "requirements definition". Im *DV-Konzept* ("design specification") werden die Fachkonzepte so transformiert, daß sie von den jeweils eingesetzten Werkzeugen verarbeitet werden können. Die *technische Implementierung* schließlich umfaßt nach Scheer neben der Programmierung auch die Festlegung physischer Datenstrukturen. Auf der anderen Seite werden drei "Sichten" unterschieden. Die *Organisationssicht* umfaßt organisatorische Einheiten, Rollen von Mitarbeitern, Aufgaben und dergleichen. Die *Funktionssicht*[1] setzt sich vor allem aus hierarchisch geordneten Funktionen zusammen, die wiederum Zielen zugeordnet sein können. In der *Datensicht* werden die im Vorgangskettenmodell erfaßten Informationsobjekte und ihre Beziehungen beschrieben.

Scheer legt sich nicht auf bestimmte Darstellungsformen für die Modellierung der einzelnen Architekturkomponenten fest. Stattdessen wird eine Reihe unter-

1. Scheer setzt dabei, in ungewöhnlicher Terminologie, Funktionen mit Vorgängen gleich.

schiedlicher Modellierungstechniken ohne eingehende Evaluation nebeneinandergestellt. Um einige zu nennen: ERM, Funktionsbäume, Organigramme, Petri-Netze, Modelle der Linearen Programmierung oder auch Diagramme der Structured Analysis. Die Verwendung sichtenspezifischer Repräsentationsformen ist offensichtlich nicht der eigentliche Gegenstand des Ansatzes. Vielmehr geht es darum, die in den einzelnen Sichten verwendeten Konzepte in *einheitlicher Weise* zu modellieren[1]. Dazu wird eine erweiterte Form von ERM verwendet.[2] Auf diese Weise werden Konzepte, die üblicherweise mit unterschiedlichen Repräsentationsformen assoziiert sind (beispielsweise Gegenstände, Prozesse, Ereignisse), allesamt auf Entitäts- und Beziehungstypen abgebildet. Dabei wird von wesentlichen Teilen der Semantik solcher Konzepte abstrahiert.

Um eine Integration der verschiedenen Sichten zu erreichen, werden die sichtenspezifischen Konzepte durch Beziehungen verbunden. ARIS sieht vor, diese Beziehungen in einem weiteren Teilmodell darzustellen, der sogenannten *Steuerungssicht*. Während der Begriff Steuerung üblicherweise mit einer Abbildung von Kontrollstrukturen assoziiert wird, beschränkt er sich in ARIS auf die Zuordnung der unterschiedlichen Konzepte (Ereignistypen, Entitätstypen, Vorgangstypen und dergleichen) mit Hilfe eines ERM.

Durch die datenorientierte Abbildung aller Konzepte und ihrer Zusammenhänge wird die Voraussetzung dafür geschaffen, die Konzepte selbst sowie ihnen zugeordnete Repräsentationen mit Hilfe eines geeigneten Datenbankmanagementsystems - Scheer spricht von Repository - zu verwalten. In einem früheren Werk (Scheer 1988 b) wird für die systematische Entwicklung von Unternehmensdatenmodellen eine Unterteilung in Funktionsbereiche (wie Produktion, Beschaffung, Verkauf, Personal etc.) vorgeschlagen, die in Form kleinerer Beispiele ("Bereichsmodelle") beschrieben werden. Für ARIS selbst gibt es kein veröffentlichtes Unternehmensmodell. An anderer Stelle (Scheer/Hars 1992) wird allerdings auf ein umfangreiches Modell verwiesen, das in Anlehnung an ARIS entstanden ist.

1. Ähnliche Ansätze finden sich in Olle et al. (1988) und Hilbers (1989).
2. Dazu gehört vor allem die Verwendung von Konzepten für die Darstellung von Generalisierung und Aggregation.

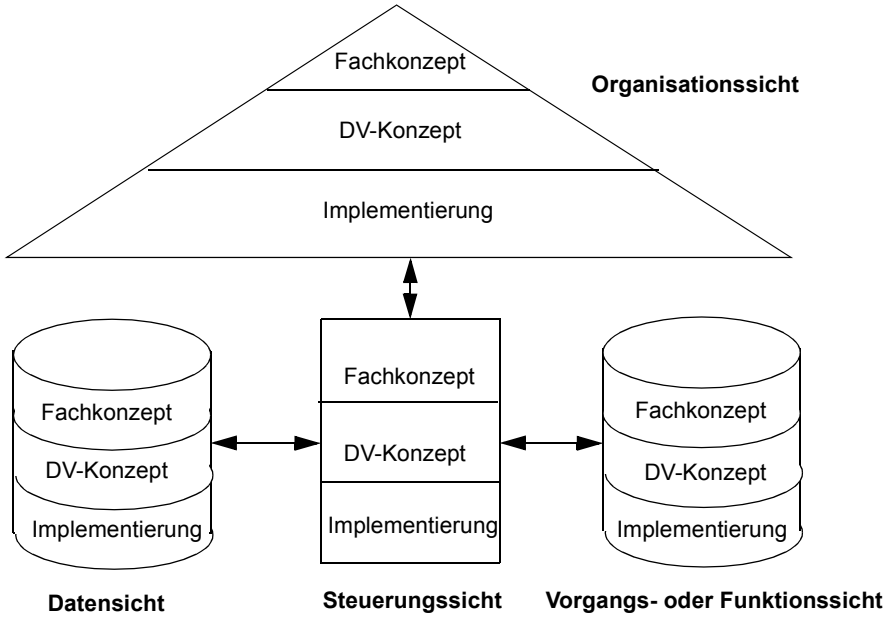


Abb. 20: Elemente der "Architektur integrierter Informationssysteme" (Scheer 1991, S. 18)

4.2.2 Meta-Informationsmodell und Integration der Sichten

Die Verbindung von Konzepten verschiedener Entwurfs- und Implementierungssichten ist notwendige Voraussetzung für eine werkzeuggestützte Verwaltung. Sie ist denn auch - in mehr oder weniger elaborierter Form - essentieller Bestandteil von CASE-Systemen. Der Anspruch von ARIS zielt nun darauf, ein generelles Referenzmodell für den rechnergestützten Entwurf betrieblicher Informationssysteme bereitzustellen. Dazu werden nicht - wie in der konzeptuellen Modellierung üblich - abstrahierende Beschreibungen realer Sachverhalte erfaßt. Vielmehr werden die Konzepte, die jeweils zur Abbildung solcher Sachverhalte dienen, dargestellt und miteinander assoziiert. Es wird also ein Meta-Informationsmodell angestrebt. Die Meta-Konzepte werden mit Hilfe eines ERM abgebildet. Es ist damit vergleichbar mit dem von IBM unterstützten "information model" (Hazzah 1990).

Es ist unstrittig, daß es einen großen Bedarf an solchen Metamodellen gibt. Nicht nur, daß sie den Entwurf von Informationssystemen anleiten und eine Architektur für deren Implementierung anbieten, sie stellen zudem einen Beitrag zur Pflege der zur Zeit wenig einheitlichen Fachterminologie dar. ARIS leistet sicherlich einen Beitrag in diesem Sinne. Das auf einem Faltblatt der Größe DIN-

A2 präsentierte Modell enthält allerdings einige Schwächen. So ist die Abbildung der Konzepte teilweise wenig differenziert, in anderen Teilen (beispielsweise bei der Beschreibung von Berichten) sehr speziell. Mitunter wird eine unangemessene Begrifflichkeit gewählt: "Trigger erzeugt Modul". Während das Konzept "Datenmodell" explizit im Metamodell enthalten ist, sucht man Konzepte für funktionale oder dynamische Modelle vergeblich - obwohl die Integration dieser verschiedenen Sichten ein explizit mit ARIS verbundenes Anliegen ist. Vor allem aber werden in dem Diagramm die einzelnen Teile der Architektur nicht in überzeugender Weise voneinander abgegrenzt. Die Datensicht des Fachkonzepts enthält unter anderem "Informationsobjekt" und "Attribut", in der Steuerungssicht des Fachkonzepts finden sich unter anderem "Objektklasse" und "Instanzvariable". Das Konzept Objektklasse ist dabei durch eine nicht näher charakterisierte Beziehung mit dem Konzept Informationsobjekt verbunden. Eine solche Differenzierung ist widersprüchlich: Wenn objektorientiert entworfen wird, gibt es neben der "Objektsicht" keine dedizierte "Datensicht". Für den Fall, daß konventionell, also mit Hilfe von Daten- und Funktionenmodellen, entworfen wird, stellt die Verknüpfung von Daten und Funktionen mit Hilfe virtueller Objektklassen in der Steuerungssicht eine reizvolle Idee dar. Es muß allerdings berücksichtigt werden, daß die Einführung solcher Klassen mit einem erheblichen Aufwand verbunden ist. Der in Olle et al. (1988) entwickelte terminologische Bezugsrahmen ist sehr viel detaillierter als der mit ARIS vorgestellte - allerdings wird jener Rahmen nicht in Form eines ganzheitlichen Architekturkonzepts präsentiert.

Jenseits von den durch geeignete Modifikationen zu heilenden Mängel von ARIS bleibt für unsere Betrachtung festzuhalten, daß die Abbildung der verschiedenen Sichten weniger darauf gerichtet ist, den Perspektiven der an der Entwicklung und Nutzung von Informationssystemen beteiligten Personen gerecht zu werden. Das Bemühen um eine für die verschiedenen Gruppen *anschauliche* Modellierung des Unternehmens liegt weitgehend außerhalb des Ansatzes. Auch wenn sich die Entwicklung von ARIS vor dem Hintergrund einer betriebswirtschaftlichen Betrachtung vollzieht (nicht zuletzt durch die Betonung der Analyse von Vorgangsketten), ist das Ergebnis doch weitgehend an den Bedürfnissen von Software-Ingenieuren ausgerichtet. ARIS ist in einer Reihe von Fällen zur Erstellung von Unternehmensmodellen eingesetzt worden (Scheer/Hars 1992). Das unterliegende softwaretechnische Konzept ist an einer getrennten Daten- und Funktionenmodellierung orientiert. Damit wird - vor dem Hintergrund heutiger Datenbanktechnologie und deren künftigen Erweiterungen - dem gegenwärtigen Bedarf der Praxis Rechnung getragen. Diesem Vorteil stehen die in III.2.4 dargestellten Defizite einer solchen Trennung gegenüber. Zwar enthält ARIS auch objektorientierte Konzepte, sie sind allerdings nicht überzeugend integriert.

4.3 CIM-Open System Architecture

Nach der Einführung des Begriffs CIM Anfang der siebziger Jahre wurde bald deutlich, daß die Umsetzung dieser attraktiven Vision viele Anwender und Anbieter überforderte. Als Reaktion darauf wurde von bedeutenden Anwendern und Anbietern - zum Teil in Kooperation - eine Reihe von Projekten iniitiert, die darauf zielten, CIM-Konzepte und -Architekturen bereitzustellen.[1] Keines dieser Projekte führte zu dem intendierten Ziel: einem für möglichst viele Anwender brauchbaren Referenzmodell. Der gegenwärtig wohl bekannteste Ansatz dazu, ein solches Referenzmodell doch noch zu realisieren, ist in dem ESPRIT-Projekt CIM-OSA (Open System Architecture) zu sehen. Für unsere Betrachtung ist dieser Ansatz aus mehreren Gründen bedeutsam. So wird ein Bezugsrahmen entwickelt, der neben einer Architektur eine Vielzahl zum Teil detailliert beschriebener Konzepte enthält. Daneben gehört die Förderung von Wiederverwendbarkeit und Integration zu den zentralen Projektzielen. Dazu wird die Etablierung verbindlicher, unter Umständen standardisierter Richtlinien angestrebt. Deshalb wird der frühzeitigen Einbindung wichtiger Interessengruppen - nicht nur im Rahmen des Projektkonsortiums - besondere Aufmerksam zuteil.

4.3.1 Ziele und Bezugsrahmen

Die Realisation von CIM-Systemen ist mit vielfältigen Herausforderungen verbunden. Produktionsabläufe und sie umgebende organisatorische Prozesse sind neu zu gestalten. Die zu automatisierenden Aufgaben und die zwischen ihnen bestehenden Zusammenhänge sind zu spezifizieren. Auf dieser Grundlage ist ein System zu entwerfen und zu implementieren. Die damit verbundene Komplexität ist von vielen potentiellen Anwendern nicht allein zu bewältigen. Eine umfassende Unterstützung durch qualifizierte Berater und Softwarehäuser ist häufig mit zu hohen Kosten verbunden. Diese Situation bildete den Anlaß für das Projekt CIM-OSA. Der Komplexität der Herausforderungen entsprechend sind die Ziele des Projekts vielfältig. Im Hinblick auf die Interessen der Projektbeteiligten wird betont, daß sowohl Anwender als auch Anbieter profitieren sollen. So wird den Anwendern ein Bezugsrahmen versprochen, der die Zusammenhänge zwischen den Komponenten eines CIM-Systems und organisatorischen Abläufen erhellt. Außerdem soll, gleichsam als technischer Bezugsrahmen, eine Architektur entwickelt werden, die den Aufbau von CIM-Systemen beschreibt. Diese Architektur ist als eine generische Referenzarchitektur gedacht, aus der spezielle Architekturen für ein konkretes Unternehmen in komfortabler Weise abgeleitet

1. Zu den bedeutendsten Vorhaben zählen: "Factory of the Future" (US-Air Force), "Purdue Workshop's Reference Model for CIM", ISO Technical Commitee 184, "Automated Manufacturing Research Facility" (National Institute of Standards and Technology). Vgl. dazu Savage (1990, S. 136)

werden können. Dadurch daß die Referenzarchitektur für viele Anwender eine wirtschaftliche Nutzung von CIM-Systemen erst ermöglichen wird, sollen auch die Anbieter profitieren. Kosanke und Vlietstra (1989) stellen ein drastisch erhöhtes Marktvolumen in Aussicht.

Es wird nachhaltig betont, daß die Konzentration auf technische Aspekte für die erfolgreiche Einführung von CIM-Systemen nicht hinreicht: "CIM-OSA embraces the complete enterprise operation, and describes it as an integrated system." (Kosanke/Vlietstra 1989, S. 663). Dazu werden drei Integrationsebenen genannt. Auf der physikalischen Ebene ("physical system integration") soll die Integration von Maschinen, Netzwerken und dergleichen gefördert werden. Auf einer darüberliegenden Schicht sollen Anwendungen integriert werden und schließlich soll der Bezugsrahmen auch der Integration von CIM in übergeordnete Geschäftsprozesse sowie der Abstimmung mit Unternehmenszielen dienen ("business integration").

Da an der Entwicklung und softwaretechnischen wie organisatorischen Implementierung von CIM-Systemen Personen mit unterschiedlichen Sichten beteiligt sind, ist ein multidisziplinärer Bezugsrahmen intendiert, der verschiedene Konzeptualisierungen und Repräsentationen enthält:

> "Many different representations of the manufacturing enterprise content and structure are required to satisfy the needs of the different users of such an architecture. CIM-OSA provides the necessary constructs to enable these multiple representations." (ESPRIT Consortium AMICE 1991, S. 3)

Der generelle Bezugsrahmen, visualisiert mit Hilfe eines Würfels (vgl. Abb. 21), enthält drei Differerenzierungsdimensionen. Zunächst werden vier Sichten auf das Unternehmen unterschieden. Im "functional view" werden Geschäfts- bzw. Produktionsprozesse abgebildet. Dazu gehört die Beschreibung von Kontrollstrukturen und relevanten organisatorischen Regeln. Der "information view" beinhaltet die Informationen, die von den Prozessen benötigt bzw. erzeugt werden. Der "resource view" ist vorgesehen für die Erfassung von Ressourcen, die nicht Informationen sind, aber für die Realisation eines Systems von wesentlicher Bedeutung sind. Dabei ist vor allem an Maschinen und Werkstoffe gedacht. Im "Organization View" schließlich werden Organisationsstrukturen, Zuordnungen von Rollen zu Aufgaben, Informationen und Ressourcen beschrieben. Diese Sichten werden in der zweiten Dimension auf *Phasen* des Entwicklungsprozesses projiziert. Dabei wird unterschieden zwischen Anforderungsanalyse, Spezifikation und Implementierung. Die dritte Dimension dient der Differenzierung von drei Generalisierungs- bzw. Spezialisierungsstufen. Die generische Ebene zielt darauf, die größte Menge von Charakteristika zu erfassen, die für alle Industriebetriebe gelten. Auf dem "partial level" finden sich speziellere Beschreibungen für bestimmte Branchen oder auf andere Art abgegrenzte Gruppen von Unternehmen. Das "particular level" schließlich dient der Spezialisierung auf konkrete

Unternehmen und liegt damit außerhalb des von CIM-OSA angebotenen Referenzmodells.

Die vier Sichten sollen durch einen Prozeß entwickelt werden, der "stepwise generation" genannt wird, was wohl heißen soll, daß sie irgendwie in kleinen Schritten zu erfassen und beschreiben sind.[1] Für die Entwicklung von der Anforderungsanalyse hin zu der Implementierung wird ein Prozeß der schrittweisen Ableitung der Repräsentationen einer Phase aus denen der davorliegenden Phase propagiert. Die Spezialisierung von generischen Beschreibungen hin zu den für ein bestimmtes Unternehmen geltenden Festlegungen wird als "stepwise instantiation" bezeichnet - gewiß eine gewöhnungsbedürftige Terminologie.

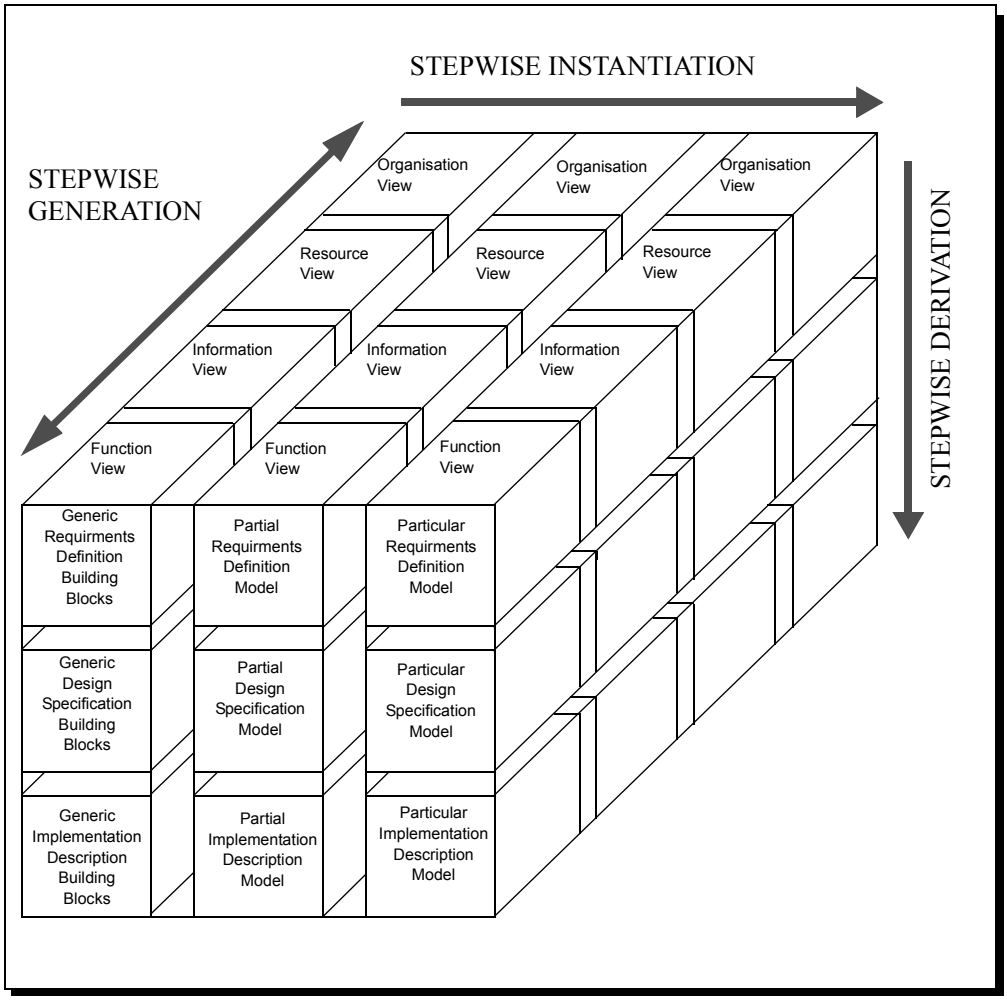


Abb. 21: Der sogenannte CIM-OSA-Würfel (ESPRIT Consortium AMICE 1991, S. 9)

1. "The Generation Process is a CIM-OSA architectural principle which forces users to think about their total enterprise in terms of Function, Information, Resource and Organisation Views." (ESPRIT Consortium AMICE 1991, S. 94).

Die durch die Differenzierung der drei Dimensionen entstehenden Teile des Würfels werden explizit als Komponenten, als "building blocks", bezeichnet. Das Anliegen von CIM-OSA besteht einerseits darin, diese einzelnen Blöcke zu beschreiben, andererseits darin, die Transformationen zwischen korrespondierenden Blöcken zu erleichtern. Auf diese Weise soll ein "modular framework for supporting the development of CIM systems throughout the whole systems life cycle" (Moss 1988, S. 238) entstehen. Dabei spielt die Orientierung an wiederverwendbaren Software-Komponenten eine wichtige Rolle. Es ist allerdings nicht explizites Ziel des Projekts, solche Komponenten bereitzustellen:

> "A strong implicit linkage (the derivation process) exists between the Requirements Definition Modelling constructs and the Implementation Description Modelling constructs to ensure realizations of the Requirements Definition Modelling constructs by the CIM vendors. Thus there will be a fair chance that system components required will exist in the market place." (ESPRIT Consortium AMICE 1991, S. 9)

4.3.2 Die Komponenten des Bezugsrahmens und ihre Repräsentation

Die funktionale Sicht wird anders als in einschlägigen Verfahren der konzeptuellen Modellierung nicht allein durch die Fokussierung auf funktionale Aspekte gekennzeichnet. Vielmehr dient sie der Darstellung von Geschäftsprozessen ("enterprise functions") aus der Sicht der Anwender ("business user's view"). Dazu soll das Unternehmen zunächst in Bereiche ("domains") unterteilt werden, um eine Lokalisierung einzelner Prozesse im Gesamtunternehmen zu unterstützen. In einem Geschäftsprozeß wird beschrieben, welche Aufgaben in welcher zeitlichen Folge erfüllt werden müssen, um bestimmte Ziele zu erreichen. Prozesse können aus Teilprozessen, Aufgaben aus Teilaufgaben komponiert werden. Zur systematischen Erfassung eines Prozesses wird eine Unterteilung in drei Aspekte vorgenommen (ESPRIT Consortium AMICE 1991, S. 24 ff.).

Im "functional part" werden einzelne Funktionen bzw. Aufgaben erfaßt. Eine Funktion wird natürlichsprachlich beschrieben. In gleicher Form werden Ziele und Randbedingungen ("Constraints") dargestellt. Außerdem werden Ein- und Ausgangsdaten festgelegt. Eine grafische Darstellung ist nicht vorgesehen. Im "behaviour part" wird das Zusammenwirken der Funktionen beschrieben. Dazu werden folgende Konstrukte bereitgestellt: Ziele, Randbedingungen, Ereignisse, "procedural rules" und termininierende Zustände. "Procedural rules" werden als Tupel von Ereignissen, Bedingungen und Aktionen dargestellt. Dabei werden verschiedene Ausprägungen unterschieden. Ein "ending status" legt fest, wann ein Prozeß beendet wird. Zur grafischen Repräsentation von Prozessen werden an anderer Stelle (ESPRIT Consortium AMICE 1991, S. 156) Zustandsdiagramme vorgeschlagen. Im "structural part" eines Geschäftsprozesses wird die Stellung des Prozesses in der Hierarchie beschrieben. Es werden also die übergeordneten und die untergeordneten Prozesse benannt. Darüber hinaus wird eine Zuordnung zu Unternehmensbereichen vorgenommen.

Aus der funktionalen Sicht der Anforderungsanalyse soll eine implementierungsorientierte Spezifikation abgeleitet werden. Sie führt zu einer Menge von "specified functional operations". Dabei wird allerdings offengelassen, wie sich eine solche Ableitung ("derivation") vollzieht. Zur Spezifikation wird eine detaillierte Konzeptualisierung von Geschäftsprozessen vorgegeben, die im folgenden in Teilen dargestellt ist (ESPRIT Consortium AMICE 1991, S. 80):

Enterprise Function	
Type:	[relevant category - select from list <name>]
Identifier:	[FU-<Unique Identifier>]
Name:	[name of Enterprise Function]
Design Authority:	[name of a person and department]
A. Functional Description	
OBJECTIVES:	[list of identifiers of all Objectives which must be fulfilled by this enterprise function]
CONSTRAINTS:	[list of identifiers of all Constraints applicable to this Enterprise Function]
DECLARATIVE RULES:	[list of identifiers and names of Declarative Rules applicable to this Enterprise Function]
.....	

Durch die Verwendung gemeinsamer Listen von Beschreibungsmerkmalen wird einerseits der Verwendung redundanter Merkmalsarten entgegengewirkt, andererseits wird das Entstehen eines einheitlichen Namensraums gefördert. Die Beschreibung von Prozessen beinhaltet auch den Verbrauch und die Produktion von Ressourcen.

Auf der Implementierungsebene schließlich wird eine Server/Client-Architektur skizziert, in der die spezifizierten Operationen als Dienste bereitgestellt werden. Die Darstellungen der drei Entwicklungsstufen beschränken sich auf die Konzeptualisierung der Modellierungskonstrukte und enthalten darüber hinaus keine Anwendungssemantik.

Der Erfassung der Informationssicht dienen vor allem "enterprise objects" und zwischen ihnen bestehende Beziehungen. Beziehungstypen sind Generalisierung, Aggregation und "particularisation" (entspricht der "instantiation relationship" von Booch, vgl. III.3.4.1). Außerdem hat der Modellierer die Möglichkeit, weitere Beziehungstypen zu definieren. Zur Beschreibung von "enterprise objects" wird auf der Ebene der Anforderungsanalyse ein objektorientierter Ansatz gewählt. Für die Spezifikation des Entwurfs wird diese Konzeptualisierung wieder aufgegeben - zugunsten einer Datenmodellierung, die dem ERM entspricht.[1] Damit wird dem Umstand Rechnung getragen, daß die Implementierung in der nächsten Zeit noch mit konventionellen Datenbankmanagement-

Systemen erfolgen wird. Sobald objektorientierte Datenbankmanagement-Systeme in der nötigen Reife verfügbar sind, soll die Konzeptualisierung von Entitäten für die Spezifikation angepaßt werden.

Sowohl die Ressourcen- als auch die Organisationssicht sind bisher noch nicht in nennenswerter Weise konzeptualisiert. Es wird lediglich vorgeschlagen, Ressourcen, die für bestimmte Aufgaben verwendet werden, in sogenannten "logical cells" zu gruppieren. Auf der organisatorischen Ebene werden den "logical cells" "organisational cells" zugeordnet, um die Zuständigkeit zu beschreiben.

Der mit CIM-OSA verbundene Anspruch ist hoch. Es wäre deshalb verfehlt, den gegenwärtigen Stand des Projekts allein an diesem Anspruch zu messen. Die vorliegenden Arbeiten stellen einen umfangreichen und in Teilen detaillierten Bezugsrahmen dar. Dieser Rahmen beschränkt sich bisher allerdings, ähnlich wie ARIS, auf die Definition eines Metamodells. Das heißt, es werden für einzelne (nicht für alle) Teile des Würfels Konzepte vorgeschlagen, die jeweils eine angemessene Beschreibung ermöglichen sollen. Abgesehen von einigen Beispielen wird keine Anwendungssemantik dargestellt - etwa in Form von "generic functions" oder "generic enterprise objects".

Während einerseits eine ganzheitliche Sicht auf das Unternehmen bewußt gegen eine einseitige Orientierung an der Gestaltung von Produktionssystemen betont wird, sind andererseits die vorliegenden Berichte deutlich an CIM-Systemen im engeren Sinne orientiert. Diese Fokussierung wurde auch schon in einer frühen Phase des Projekts explizit vorgenommen: "Although company activities range from management to research and from sales to quality assurance, presently we have restricted our work to the manufacturing area." (Moss 1988, S. 243)

Neben den skizzierten Lücken, die noch gefüllt werden können bzw. zum Teil vielleicht schon gefüllt sind (die vorliegenden Publikationen dokumentieren nicht den aktuellen Stand des Projekts) sind allerdings einige konzeptionelle Schwächen zu bemängeln. Sie betreffen zum einen widersprüchliche oder unklare Projektziele. So wird aus den vorliegenden Darstellungen nicht deutlich, wie weit die Detaillierung bzw. Formalisierung des Modells fortschreiten soll. Einerseits wird darauf verwiesen, daß das Referenzmodell nur "generic characteristics" (ESPRIT Consortium AMICE 1991, S. 8) enthalten sollte, andererseits werden teilweise technische Details behandelt (wie etwa Inter-Prozeß-Kommunikation durch "message passing" oder "shared memory", Kosanke/Vlietstra 1989, S. 769), von denen auf einem generischen Betrachtungsniveau abstrahiert werden müßte. Während an einigen Stellen explizit eine formale Beschreibung der Architekturelemente gefordert wird, enthalten die verwendeten Konzepte - wie beispielsweise Geschäftsprozesse - Teile, für die eine natürlichsprachliche Beschreibung vorgesehen ist.

1. Sie wird allerdings "Entity-Relationship-Attribute approach" genannt.

Das Konzept der "building blocks" zielt darauf, für abgrenzbare Aspekte von der Analyse bis zur Implementierung einander entsprechende Konstrukte zu verwenden. Ein solches Ziel empfiehlt einen objektorientierten Ansatz. Tatsächlich ist bei einigen Mitarbeitern des Projekts eine ausgeprägte Affinität zu objektorientierten Ansätzen zu verzeichnen (Emond/Hermans 1988). Daß dennoch für Spezifikation und Implementierung (zunächst) von einer Trennung in Daten- und Funktionssicht ausgegangen wird, macht Sinn: Das Modell soll schließlich unter gegenwärtig vorherrschenden Bedingungen in der Praxis verwendbar sein. Eine solche Trennung ist mit einer besonderen Herausforderung für die Integration von Sichten und Entwicklungsphasen verbunden. An einigen Stellen werden in konsequenter Weise Voraussetzungen für Integration geschaffen. Dabei ist vor allem an die Verwendung gemeinsamer Merkmalstypen für die Spezifikation von Funktionen und Entitäten zu denken. Die Transformation ("derivation") von objektorientierter Analyse hin zu konventionellem Entwurf wird allerdings nicht hinreichend beschrieben (beispielsweise: Wie wird die prozedurale Semantik einer Klasse auf das Funktionenmodell abgebildet?).

Die Differenzierung in verschiedene Sichten und verschiedene Entwicklungsstufen ist grundsätzlich zu begrüßen. Die Dimensionen Spezialisierung ("stepwise instantiation") und Entwicklung ("stepwise derivation") sind nach gängiger Terminologie zwar nicht orthogonal zueinander (Spezialisierung kann den Entwicklungsprozeß selbst kennzeichnen) - aber als analytisch motivierte Differenzierung sind sie durchaus brauchbar. Die Zuordnung von Rollen zu Sichten ist problematisch. So wird die funktionale Sicht auf das Unternehmen als Perspektive der Anwender gekennzeichnet (s.o.) Dies trifft allerdings allenfalls für die Anforderungsanalyse zu (und da vor allem für das "particular model"). Insofern scheint es angemessener, nicht die vier Sichten jeweils allein zur Zuordnung von Betrachterrollen zu verwenden, sondern mit Hilfe der anderen Dimensionen weiter zu differenzieren. Wichtiger ist jedoch der Umstand, daß für die Abbildung der Anwendersicht keine anschaulichen Repräsentationsformen vorgeschlagen werden.

4.3.3 Maßnahmen zur Standardisierung

Der im Rahmen des Projekts entwickelte Bezugsrahmen soll zu einem Referenzmodell für weite Teile der Industrie werden. Es wurde deshalb von Beginn[1] an darauf gezielt, ein verbindliches Engagement wichtiger Anbieter und Verwender zu erreichen. Zu diesem Zweck wurde das AMICE[2]-Konsortium gegründet, dem

1. Die ersten Planungen gehen zurück auf das Jahr 1984 (Jorysz/Vernadat 1989 a, S. 144).
2. AMICE ist das reversierte Akronym für "European Computer Integrated Manufacturing Architecure".

ungefähr zwanzig Unternehmen und einige Forschungseinrichtungen (Eversheim/Klevers 1989) angehören. Zu den beteiligten Unternehmen gehören große Anbieter (neben anderen IBM, SNI, Hewlett-Packard, DIGITAL) ebenso wie bedeutende Industrieunternehmen als potentielle Verwender (so etwa VW, FIAT, AEG).

Die systematische Öffentlichkeitsarbeit hat mittlerweile zu einem hohen Bekanntheitsgrad in Fachkreisen geführt. An dem Projekt sind ca. 50 Mitarbeiter beteiligt - allerdings bis auf einen hauptamtlichen Koordinator in Brüssel nur zu einem kleinen Teil ihrer Gesamtarbeitszeit. Insgesamt ist von ca. fünf Mannjahren pro Jahr die Rede.[1] Die im Projekt erarbeiteten Teilmodelle werden regelmäßig zur Diskussion gestellt, um eine kritische Revision anzuregen. Das geschieht bisher vor allem in Form von Berichten. Es ist allerdings erklärtes Ziel, auch frühe Prototypen und Simulatoren zu entwickeln. Emond und Hermans (1988) stellen einen (in Smalltalk implementierten) Prototyp vor, der es gestattet, durch das Geflecht der definierten Konzepte zu navigieren.

Auch wenn namhafte Unternehmen an dem Projekt beteiligt sind, gibt es dennoch Schwierigkeiten, die Verbreitung von CIM-OSA zu forcieren. Das liegt zum einen an der unzureichenden Unterstützung durch die beteiligten Großunternehmen. Wie in anderen Standardisierungsgruppen auch ist es bei einigen Beteiligten offenkundig, daß man vor allem die Aktivitäten der Konkurrenz beobachten möchte. Darüber hinaus entsteht mitunter der Eindruck, daß die Beteiligung gerade darauf gerichtet ist, das gemeinsam formulierte Ziel, die Etablierung eines standardisierten Modells, zu blockieren.[2] Zum anderen ist zu berücksichtigen, daß die meisten der für den zu erwartenden CIM-Markt so bedeutenden mittleren und großen Industrieunternehmen nicht beteiligt sind.

Die in CIM-OSA gemachten Erfahrungen zeigen auf zwei wesentliche Gesichtspunkte, die zu beachten sind, wenn gemeinsame Referenzmodelle angestrebt werden. Einerseits muß das bereits erwähnte Standardisierungsdilemma berücksichtigt werden. Die mit einer Standardisierung verbundenen Anreize werden erst nach Etablierung der Standards wirksam - die Beteiligung an Standardisierungsprozessen ist allerdings mit Kosten und Risiken verbunden:

> "It is a well known difficulty for proposals of computer architectures to reach a sufficient credibility before they are supported by some implementations." (Emond/Hermans 1988, S. 266)

Wenn die Entwicklung komplexer Referenzmodelle mit Engagement betrieben wird, ergibt sich für die Beteiligten das Problem, ihre Eigenleistung wirksam zu

1. Diese Angaben stammen aus persönlichen Gesprächen mit Projektbeteiligten.
2. So gibt es bei einigen Anbietern, parallel zu der bescheidenen Beteiligung in AMICE, sehr viel umfangreichere Anstrengungen zur Entwicklung proprietärer CIM-Modelle.

schützen. Darüber hinaus ist zu berücksichtigen, daß die Etablierung von Standards nicht für alle Betroffenen ein erstrebenswertes Ziel ist. Für uns bleibt festzuhalten, daß Konsortien, die auf die Etablierung gemeinsamer Referenzmodelle gerichtet sind, mit erheblichen Problemen zu rechnen haben, wenn die Beteiligten im wirtschaftlichen Wettbewerb stehen.

4.4 Der Bezugsrahmen von Zachman

Seit Mitte der sechziger Jahre war John Zachman bei IBM im Marketingbereich mit der strategischen Planung von Informationssystemen beschäftigt. Sein Umgang mit verschiedenen Beteiligten - wie Vertriebsbeauftrage, Manager und Software-Ingenieure - hatte ihm verdeutlicht, daß es eine Reihe unterschiedlicher Sichten auf Informationssysteme sowie mit ihnen verbundene Konzeptualisierungen und Fachsprachen gibt. Als Reaktion darauf entwickelte Zachman (1987) einen Bezugsrahmen, der wesentliche Sichten und ihre Beziehungen zueinander verdeutlichen sollte. Gleichwohl er als "Information System Architecture" eingeführt wurde, dient der Bezugsrahmen doch vor allem der Förderung des Verständnisses von Informationssystemen und ihrer Einbettung in Unternehmen. Der Bezugsrahmen hat nicht unmittelbar Eingang in das mit AD-Cycle propagierte "information model" (Devlin/Murphy 1988) gefunden, ist aber - nicht nur innerhalb von IBM - auf große Aufmerksamkeit gestoßen. Besondere Attraktivität für unsere Betrachtung erhält der Bezugsrahmen durch eine Erweiterung, die Zachman unlängst zusammen mit dem Logiker Sowa vorlegte. Sie schließt konzeptionelle Lücken und beinhaltet einen ambitionierten Ansatz zur Integration der verschiedenen Sichten des Bezugsrahmens.

4.4.1 Information System Architecture

Zachman entwickelt die erste Version seines Bezugsrahmens in Analogie zur Architektur eines Hauses. Anhand verschiedener Rollen (wie "owner", "architect", "contractor", "builder") macht er deutlich, daß jeweils unterschiedliche Konzeptualisierungen des Hauses und seiner Teile im Vordergrund stehen. Die mit den Rollen verbundenen Sichten differenziert er weiter in drei Aspekte, die durch die Interrogative "was" ("material"), "wie" ("function") und "wo" ("location") gekennzeichnet werden. Die Übertragung dieser Vorüberlegungen auf betriebliche Informationssysteme führt zu einem Bezugsrahmen mit fünf Sichten und drei Betrachtungsaspekten. Jede der auf diese Weise entstehenden fünfzehn Teilsichten wird mit einer bestimmten Repräsentationsform assoziiert.

Der "ballpark view" (auch "scope description" genannt) entspricht einer ersten Wunschliste des Bauherrn. Wie die anderen Sichten auch wird er unterteilt in die Aspekte "data description" ("was"), "process description" ("wie") und "network description" ("wo"). Die drei Teilsichten werden mit Hilfe von Listen repräsen-

tiert, die die wichtigsten Begriffe enthalten. Im "owner's view" (oder "model of the business") werden diese Begriffe näher beschrieben und - beschränkt auf die einzelnen Teilsichten - zueinander in Beziehung gesetzt. Als zugehörige Darstellungsformen nennt Zachman beispielhaft ERM-Diagramme, Funktionsdiagramme und "logistic networks", in denen die zwischen einzelnen Standorten bestehenden Beziehungen abgebildet werden. Der "designer's view" ("model of the information system") kann mit Hilfe von ERM-Diagrammen[1], Datenflußdiagrammen und Plänen zur Darstellung der physischen Verteilung eines Informationssystems abgebildet werden. Der "builder's view" ("technology model") schließlich enthält implementierungsorientierte Darstellungen. Die letzte Sicht ("detailed description") wird mit Hilfe von Implementierungssprachen konkretisiert. Zachman erwähnt sie lediglich der Vollständigkeit halber ("out of scope").

Schon in der ersten Darstellung des Bezugsrahmens weist Zachman (1987, S. 282) darauf hin, daß noch weitere Aspekte zu berücksichtigen seien: "The implications are that there must be at least "WHO", "WHEN", and "WHY" descriptions as well." Diese Aspekte werden denn auch in die Weiterentwicklung des Bezugsrahmens (Sowa/Zachman 1992) aufgenommen. Unter dem Aspekt "people" werden organisatorische Einheiten und Rollen sowie deren Zuordnung zu Aufgaben beschrieben. Die Abbildung zeitlicher Abläufe und der diese festlegenden Konstrollstrukturen erfolgt unter dem Aspekt "Time". Schließlich werden, zusammengefaßt unter dem Aspekt "motivation", auf verschiedenen Abstraktionsstufen Ziel/Mittel-Relationen dargestellt. Zudem werden dem "ballpark view" und der Implementierungssicht explizit Rollen zugeordnet, nämlich "planner" und "subcontractor". Alle Sichten werden zusätzlich durch die jeweils im Vordergrund stehenden Randbedingungen gekennzeichnet (Sowa/Zachman 1992, S. 592):

Perspective	*Constraint*	*Model*
Planner	Financial/external	Scope
Owner	Usage/Policy	Enterprise model
Designer	Structure/operation	System model
Builder	Technology	Technology model
Subcontractor	Implementation	Out-of-context models

Dabei fällt die restriktive Auslegung des Begriffs "enterprise model" auf. Ein solche Diktion ist insofern verwunderlich, als Zachman und Sowa (1992, S. 600) explizit darauf verweisen, daß "Each column represents an abstraction from the real-world enterprise for convenience of design." Die den einzelnen Teilsichten zugeordnenten Repräsentationsformen werden wiederum lediglich als Beispiele eingeführt - sie können bei der konkreten Anwendung des Bezugsrahmens spezi-

1. Hierbei handelt es sich im Unterschied zum "owner's view" um ein ERM der Ebene 2, also ein Datenmodell (vgl. III.2.2.1).

fiziert werden. Um die Anwendung anzuleiten, formulieren Zachman und Sowa sieben Prinzipien. Dazu gehört die Regel, daß jedem Aspekt ein für alle Sichten verwendbares Metamodell zuzuordnen ist. Ein solches Metamodell beinhaltet Konzepte zur Beschreibung eines Aspekts, die für die verschiedenen Sichten in unterschiedlicher Detaillierung verwendet werden können. Für die Datensicht wird beispielhaft das ERM genannt. Bei näherer Betrachtung des Bezugsrahmens (vgl. Abb. 22) reduziert sich die Beschreibung eines Metamodells auf die Festlegung von jeweils zwei charakteristischen Konzepten (wie "entity" und "relationship" oder "link" und "node"). Eine weitere Regel besagt, daß sich die Basis-Konzepte der verschiedenen Aspekte voneinander unterscheiden müssen. Damit soll redundanten Repräsentationen entgegengewirkt werden.

Der Bezugsrahmen ist - anders als bei Scheer oder CIM-OSA - nicht vordergründig darauf gerichtet, den Entwicklern von Informationssystemen detaillierte Beschreibungskonzepte zur Verfügung zu stellen. So gehören auch Personen, die im Umgang mit Informationstechnik nicht versiert sind, zu Zachmans Zielgruppe - was sich beispielsweise in der didaktisch motivierten Analogie zum Hausbau ausdrückt. Zachman zielt vor allem darauf, wichtige Perspektiven auf Informationssysteme zu identifizieren und ein Medium zur Vermittlung zwischen diesen Perspektiven bereitzustellen. Der einzelne Betrachter wird dadurch in die Lage versetzt, seine Sicht in einen größeren Kontext einzuordnen, wodurch sie letztlich auch besser kommunizierbar wird. Die Beschreibungskonzepte werden auf einer Ebenen eingeführt, die von konkreten Darstellungstechniken (wie ERM oder Datenflußdiagramme) abstrahiert[1] - sie werden lediglich als Beispiele genannt. Es handelt sich also gleichsam um ein "Meta-Metamodell".

Angesichts der Vielfalt möglicher Perspektiven auf Informationssysteme scheint die Differenzierung von insgesamt 30 Teilsichten nicht unangemessen. Es stellt sich allerdings die Frage, ob die mit der geforderten eindeutigen Differenzierung verbundene Komplexität das Anliegen, Anschaulichkeit und Verständlichkeit zu fördern, nicht konterkariert. Dieser Einwand gilt umso mehr als nicht nur der "subcontractor’ view" - wie Zachman selbst einräumt - vernachlässigt werden kann. Wenn man sich auf die konzeptuelle Modellierung beschränkt, ist zudem die Sicht der Programmierer ("builder’s view") entbehrlich. Auch der Aspekt "network" scheint in der vorliegenden Form nicht notwendig: Bei der Modellierung (und möglichst auch bei der Implementierung) sollte von der konkreten physischen Verteilung eines Systems ohnehin abstrahiert werden. Damit ist nicht gesagt, daß räumliche Verteilung grundsätzlich nicht zu berücksichtigen ist. Sie ist allerdings vor allem bei der Analyse von Geschäftsprozessen von Bedeutung.

1. Peters/Schultz (1993) propagieren die Verwendung des Modells von Sowa und Zachman als heuristische Hilfe für die objektorientierte Unternehmensmodellierung - allerdings ohne näher auf das Procedere einzugehen.

	DATA Entity Relationship	FUNCTION Function Argument	NETWORK Node Link	PEOPLE Agent Work	TIME Time Cycle	MOTIVATION Ends Means
SCOPE Planner	**List of things important to the business** entity: class of business thing	**List of processes the business performs** function: class of business process	**List of locations** node: major business location	**List of organizations/ people** agent: major organization unit	**List of events** time: major business event	**List of business goals** ends/means: major business
ENTERPRISE MODEL Owner	**e.g., "ERM-Diagram"** entity: business entity	**e.g., "process flow"** function: business process	**e.g., "logistics network"** node: business location	**e.g., "organizat. chart"** agent: organization unit	**e.g., "master schedule"** time: business event cycle: bus. cycle	**e.g., "business plan"** ends: business objective
SYSTEM MODEL Designer	**e.g., "data model"** entity: data entity rel.: data relationship	**e.g., "data flow diag."** function: appli. function	**e.g., "distributed system architecture"** node: I/S function link: line characteristics	**e.g., "human interface archi."** agent: role work: deliverable	**e.g., "processing struc."** time: system event cycle: process cycle	**e.g., "knowledge archi."** ends: criterion means: option
TECHNOLOGY MODEL Builder	**e.g., "data design"** entity: segment/row rel.: pointer/key	**e.g., "structure chart"** func.: computer function	**e.g., "system archi."** node: hardware/sys. softw.	**e.g. "human/technology interface"** agent: user work: job	**e.g., "control structure"** time: execute cycle: component	**e.g., "knowledge design"** ends: condition means: action
COMPONENTS Subcontractor	**e.g., "data definition description"** entity: field rel.: address	**e.g., "program"** function: lang. statement	**e.g., "network architecture"** node: address link: protocol	**e.g., "security architecture"** agent: identity work: "transaction"	**e.g., "timing definition"** time: interrupt cycle: machine cycle	**e.g., "knowledge definition"** ends: subcontition means: step

Abb. 22: Der erweiterte Bezugsrahmen von Zachman (Sowa/Zachman 1992, S. 600 f.)

Es ist positiv zu vermerken, daß Sowa und Zachman im Unterschied zu ARIS oder CIM-OSA strategische Planung als einen eigenständigen Aspekt einführen.

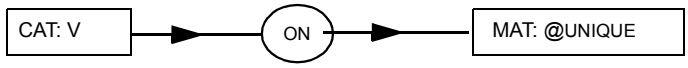
4.4.2 Die Integration der Teilsichten

Im Hinblick auf die werkzeuggestützte Verwaltung der mit den in dem Bezugsrahmen differenzierten Sichten assoziierten Repräsentationen ist es erforderlich, den Zusammenhang zwischen den jeweils verwendeten Konzepten zu verdeutlichen. Für die verschiedenen Sichten eines Aspekts geschieht dies durch die Vorgabe gemeinsamer Konzepte. Auf diese Weise wird die Identifikation korrespondierender Teile in verschiedenen Sichten unterstützt. Der Bezugsrahmen enthält allerdings keine konkreten Hinweise darauf, wie die einzelnen Aspekte (die ja durch je unterschiedliche Konzepte gekennzeichnet sind) miteinander verbunden werden können.

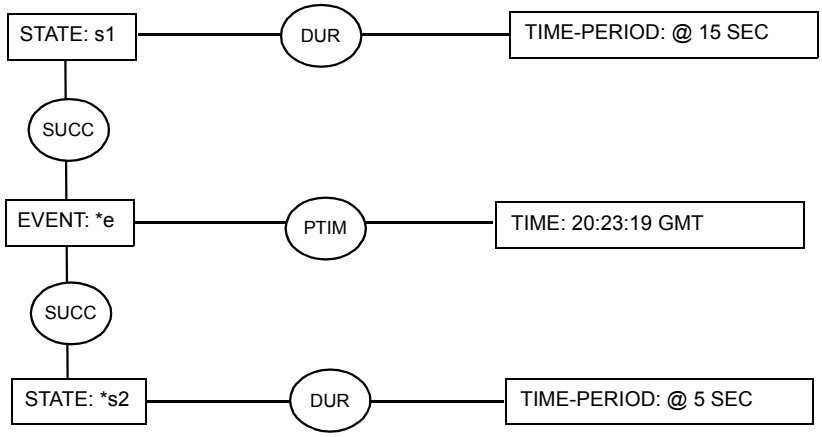
Als erste Maßnahme zur Integration der Teilsichten, wird in Sowa/Zachman (1992) ein ERM eingeführt, in dem die jeweils verwendeten Konzepte zueinander in Beziehung gesetzt werden - also beispielsweise das Konzept "Input/Output" des Aspekts "Funktion" und das Konzept "Entity" bzw. "Attribute" des Aspekts "Daten". Auf diese Weise ergibt sich eine Darstellung, die der von ARIS ähnelt. Solche Beziehungen zwischen den verwendeten Konzepten sind hilfreich, um gewisse Formen der referentiellen Integrität zu ermöglichen. Sowa und Zachman trachten allerdings nach einem höheren Integrationsniveau. Dazu soll die Semantik der in den einzelnen Teilsichten verwendeten Konzepte mit Hilfe einer Metasprache rekonstruiert werden. Auf diese Weise entsteht ein gemeinsames semantisches Referenzsystem, in dem alle Konzepte in einheitlicher Form beschrieben sind. Es würde in seiner Gesamtheit eine vollständige Beschreibung des Informationssystems und der relevanten Sichten auf dasselbe bieten - gleichsam ein "Superschema". Die natürliche Sprache wäre mächtig genug, um eine solche Darstellung zu ermöglichen. Sie ist allerdings nicht angemessen, wenn eine automatische Interpretation angestrebt wird. Genau dies ist die Vision, die Sowa und Zachman entwickeln. Durch die Abbildung der einzelnen Teilmodelle auf eine gemeinsame Sprache können alle Modelle in einem einheitlichen Repository verwaltet werden. Außerdem können Transformationsregeln zwischen den verschiedenen Repräsentationsformen unter Verweis auf die gemeinsame Meta-Ebene definiert werden. Dadurch daß die Semantik der zur Modellierung verwendeten Konzepte auf einer höheren Ebene beschrieben ist, ergibt sich zudem eine größere Flexibilität: Wenn einzelne Konzepte erweitert oder durch andere ersetzt werden (wie beispielsweise Entitätstypen durch Klassen), wird diese Veränderung in der Meta-Sprache beschrieben - ohne daß die Verfahren, die diese Meta-Sprache interpretieren, geändert werden müßten.

Als Meta-Sprache käme ein Formalismus wie beispielsweise die Prädikatenlogik

in Frage. Sowa und Zachman wollen aber über die semantische Integration hinaus auch eine Vereinheitlichung der Repräsentationsformen der einzelnen Sichten ermöglichen. Sie schließen die Prädikatenlogik deshalb aus, da ihre Darstellung nicht hinreichend anschaulich ist. Stattdessen empfehlen sie die Verwendung von "conceptual graphs". Solche konzeptuellen Graphen bieten Ausdrucksmöglichkeiten, die denen der Prädikatenlogik entsprechen, in einer anschaulichen grafischen Form. Sie sind in Anlehnung an die Arbeiten des Logikers Pierce und an semantische Netze, wie sie in der Künstliche Intelligenz Forschung verwendet werden, entstanden. In einem konzeptuellen Graph werden Objekte (oder in der Diktion der Prädikationlogik: Subjekte) mit Hilfe von Prädikaten ausgezeichnet. Da Prädikate mehrstellig sein können, dienen sie auch der Darstellung von Beziehungen zwischen Subjekten, die wiederum mit Quantoren ausgezeichnet werden können.



a) "Jede Katze gehört zu genau einem Halter"



b)
DUR "dauert" SUCC
SUCC "wird gefolgt von"
PTIM "erfolgt zum Zeitpunkt"

Abb. 23: Beispiel für konzeptuelle Graphen zur Ersetzung von ERM (a) und von Zustandsdiagrammen (b). Die Darstellung der zeitlichen Zuzsammenhänge erfolgt dabei auf Instanzebene. (Sowa/Zachman 1992, S. 607 u. 609)

Abbildung 23 zeigt Beispiele konzeptueller Graphen. Grundsätzlich sind konzeptuelle Graphen geeignet, "any level of the framework" abzubilden. Einschränkend ist jedoch darauf hinzuweisen, daß es sich dabei aus der Sicht der Betrach-

ter vor allem um eine Vereinheitlichung der verwendeten grafischen Symbole handelt. Die an die Prädikate und Subjekte geknüpfte Bedeutung ist von Sicht zu Sicht verschieden. Sowa und Zachman treten allerdings nicht für eine bedingungslose Ersetzung bisheriger Repräsentationsformalismen ein. Vielmehr soll es den Präferenzen des Modellierers überlassen bleiben, vertraute Repräsentationsformen beizubehalten oder konzeptuelle Graphen zu verwenden. Es kann allerdings in jedem Fall eine Repräsentation in Form konzeptueller Graphen erzeugt werden, da die Konzepte aller Teilsichten nach festen Regeln in konzeptuelle Graphen transformiert werden können. Darüber hinaus wird eine die grafische Darstellung ergänzende Repräsentation in Aussicht gestellt: Da ein konzeptueller Graph in seinem Aufbau einer vereinfachten Syntax der natürlichen Sprache gleicht, können (hölzern wirkende) Sätze der natürlichen Sprache generiert werden.

Die skizzierte Anwendung der konzeptuellen Graphen ist bewußt visionär gehalten.[1] Sie erinnert an die - in anderer Intention - entwickelte "Theory of conceptual dependency", in der Schank (1972) sogenannte "semantic primitives" zur Rekonstruktion der Bedeutung natürlicher Sprachen einführt. Sowa und Zachman gehen nicht weiter auf die vielfältigen Herausforderungen der Rekonstruktion von Modellierungskonzepten ein. Zudem wecken sie den unangemessenen Eindruck, daß mit Hilfe von konzeptuellen Graphen auch Sachverhalte beschrieben werden können, die - wie beispielsweise Unternehmensziele - nicht immer formalisierbar[2] sind. Dennoch ist der Vorschlag einer Meta-Sprache für unsere Betrachtung eine reizvolle Anregung: Da nach der hier entwickelten Begrifflichkeit Integration mit der Semantik der gemeinsamen Referenzen zunimmt, ist die Einführung einer Meta-Sprache eine Orientierung für das Erreichen höherer Integrationsstufen.[3]

5. Zusammenfassung: Ansatzpunkte und Forschungspotentiale

Die Beurteilung der untersuchten Ansätze zur Unternehmensmodellierung ist abhängig von den im Einzelfall zu berücksichtigenden Randbedingungen. So können die Methoden zur Unterstützung der Analyse und des Entwurfs aus der Sicht der Anwender kaum ohne die sie begleitenden Werkzeuge und Laufzeitum-

1. Sowa/Zachman (1992, S. 614) selbst betonen, daß konzeptuelle Graphen der skizzierten Art in nächster Zukunft nicht in einem (IBM-) Produkt verfügbar sein werden.
2. In Frank (1988) wird ein ähnlicher Vorschlag, "formale Geschäftssprache" genannt, entwickelt. Dabei werden die damit verbundenen Herausforderungen im Detail erörtert.
3. Wobei in einem fortschreitenden Regreß auch Meta-Sprachen von Meta-Sprachen gebildet werden können.

gebungen (wie vor allem Datenbankmanagementsysteme) bewertet werden. Objektorientierte Ansätze bieten konzeptionell deutliche Vorteile gegenüber traditionellen Ansätzen, in denen Daten und Funktionen getrennt behandelt werden. Eine objektorientierte Abstraktion verspricht eine authentischere Abbildung realweltlicher Gegenstände und Konzepte als eine Datenabstraktion. Dadurch werden die Friktionen zwischen den Phasen des Lebenszyklus gemildert, denn es können von der Analyse bis zur Implementierung die gleichen Konstrukte (wenn auch in unterschiedlicher Detaillierung und Genauigkeit) verwendet werden. Die gegenüber Datenstrukturen reichhaltigere Semantik von Klassen unterstützt die Etablierung gemeinsamer Referenzsysteme auf einem hohen semantischen Niveau und trägt damit zu einer höheren Integration von Informationssystemen bei. Vererbung unterstützt die Vermeidung von Redundanz und leistet damit einen wichtigen Beitrag für die Integrität und Wartbarkeit von Systemen. Verkapselung hilft, unerwünschte Seiteneffekte zu vermeiden. Schließlich sind Klassen ein besseres Medium für die Bereitstellung wiederverwendbarer Software als Funktionen und Datenstrukturen: Verkapselung schützt vor Mißbrauch und Spezialisierung in Unterklassen trägt zu einer sicheren Anpassung an individuelle Anforderungen bei.

Diesen Vorteilen stehen allerdings Nachteile gegenüber. Aus der Sicht der Praxis ist hier in erster Linie an die mangelnde Verfügbarkeit geeigneter Entwurfs- und Laufzeitumgebungen zu denken sowie daran, daß die Ersetzung der Vielzahl bestehender Anwendungen durch objektorientierte Systeme kurzfristig mit einem kaum zumutbaren Aufwand verbunden ist. Jenseits solcher Kriterien, die die gegenwärtigen ökonomischen Randbedingungen betreffen, ist den objektorientierten Methoden entgegenzuhalten, daß ihre Anwendung mit einem hohen Aufwand verbunden ist. Der Entwurf eines Gesamtsystems bleibt dem Entwickler überlassen, es werden lediglich elementare Konstrukte zur Verfügung gestellt. Dadurch beschränkt sich der Beitrag zu Integration und Wiederverwendbarkeit auf eine grundlegende softwaretechnische Ebene.

Unternehmensmodelle oder Informationssystem-Architekturen zielen darauf, den Entwurf von Gesamtsystemen zu erleichtern. Dazu werden - detaillierter als in den Entwurfsmethoden - verschiedene Perspektiven unterschieden. Dabei werden auch - mehr oder weniger pointiert - die Sichten von solchen Personen berücksichtigt, die nicht unmittelbar mit der Systemgestaltung befaßt sind. Die Vielfalt der in den drei betrachteten Modellen vorgeschlagenen Perspektiven stellt einen wichtigen Fundus für zukünftige Ansätze zur Unternehmensmodellierung dar. Es sind aber auch Defizite erkennbar. So bleibt die Abbildung der Organisationssicht auf rudimentäre Aspekte beschränkt. Außerdem sind keine Möglichkeiten vorgesehen, organisatorische Alternativen zu modellieren und zu bewerten.[1] Ähnliches gilt für die Einordnung der Gestaltung von Informationssy-

stemen in längerfristige Planungsstrategien. Zachman betont zwar die Bedeutung einer solchen Perspektive, geht allerdings nicht im Detail auf sie ein. Es werden zudem keine Repräsentationen von Teilsichten angeboten, die auch für Nicht-Entwickler anschaulich wären.

Unternehmensmodelle dienen nicht zuletzt dazu, eine anschauliche Beschreibung wichtiger Sachverhalte und Zusammenhänge in einem Unternehmen zu liefern. Auch wenn damit vordergründig auf eine Unterstützung der Informationssystemgestaltung gezielt wird, soll darüber hinaus - jedenfalls vor dem Hintegrund des hier vertretenen Integrationsbegriffs - ein Medium geschaffen werden, das das Verständnis der Zusammenhänge fördert und den Diskurs über das Unternehmen erleichtert. Dazu ist gewiß nicht immer eine formalisierte Darstellung nötig. Mitunter wäre sie hinderlich und zum Teil ist sie gar nicht möglich. Unternehmen lassen sich allerdings auch in nicht formalisierter Weise gehaltvoll beschreiben - man denke nur an die Betriebswirtschaftslehre und andere Disziplinen in ihrem Umfeld (wie die Rechtswissenschaften oder die Soziologie). Die dargestellten Bezugsrahmen zur Unternehmensmodellierung vernachlässigen solche nicht formalisierten Beschreibungen. Sie sind vielmehr auf eine formale Rekonstruktion von Unternehmen gerichtet - der durchgehend verwendete Begriff "Architektur" mag als Beleg dafür gelten. Auch wenn eine solche Fokussierung im Hinblick auf die Implementierung von Software sinnvoll erscheint, kann doch zweierlei nicht übersehen werden: Ein Unternehmensmodell kann durch die Integration nicht formalisierter Darstellungen angereichert werden. Die Integration dazu geeigneter Konzepte (wie Texte, Bilder etc.) erfordert deren formale Beschreibung in der Architektur.

Die betrachteten Gesamtmodelle verfügen über ein mehr oder weniger elaboriertes Metamodell, in dem die Beschreibungskonzepte der verschiedenen Perspektiven zueinander in Beziehung gesetzt werden. Ein solches Metamodell ist eine wesentliche Voraussetzung für die systematische Integration der Teilsichten zum Zweck der Implementierung und eine conditio sine qua non für den Entwurf zugehöriger CASE-Umgebungen. Scheer und vor allem die Verfasser von CIM-OSA lassen zwar eine Präferenz für objektorientierte Ansätze erkennen, gehen allerdings letztlich doch von einem traditionellen Ansatz aus.[1] Die oben skizzierten Vorteile objektorientierter Modellierung werden also erkannt, aber - im Hinblick auf die Randbedingungen in der Praxis mit gutem Grund - allenfalls verhalten genutzt.

---

1. Es gibt zwar eine große Zahl von Methoden und Techniken zur Organisationsanalyse (eine vergleichende Gegenüberstellung findet sich in Frank/Kronen 1991), aber sie sind eben nicht in die oben untersuchten Modelle integriert.

1. Sowa und Zachman können hier ausgenommen werden, da sie softwaretechnische Konzepte auf einem anderen Abstraktionsniveau betrachten.

Während alle drei Modelle bzw. Architekturen explizit einen Beitrag zur Integration leisten wollen, wird das Ziel Wiederverwendbarkeit nur in CIM-OSA mit Nachdruck betont. Dabei wird zweierlei deutlich: Um Wiederverwendbarkeit zu ermöglichen, sollten die Anforderungen der potentiellen Verwender berücksichtigt werden. Dazu ist es angeraten - wie in CIM-OSA vorgesehen - möglichst viele Verwender frühzeitig zu einer engagierten und verbindlichen Mitwirkung zu bewegen. Die Erfahrungen in CIM-OSA zeigen aber auch, wie schwierig ein solcher Prozeß zu gestalten ist, wenn elementare wirtschaftliche Interessen der Beteiligten berührt werden.

Um eine Alternative zu den dargestellten Ansätzen zu entwickeln, die sich einerseits ekletizistisch an diesen bedient, andererseits die skizzierten Lücken (in Teilen) füllt, werden wir im folgenden von einigen Restriktionen der gegenwärtigen Anwendungspraxis absehen. Dazu gehören die Qualifikation der eingesetzten Entwickler, die installierten und am Markt verfügbaren Werkzeuge und Laufzeitumgebungen sowie die große Zahl alter Anwendungen. Damit ist keine Mißachtung der in der Praxis bestehenden Probleme verbunden. Es wird lediglich von ihnen abstrahiert, um die Chance zu verbessern, innovative Lösungen zu skizzieren. In gewissem Maße ist eine solche Abstraktion auch Ausdruck einer notwendigen Berücksichtigung zeitlicher Zusammenhänge: Erst dann, wenn neue Ansätze zur Gestaltung betrieblicher Informationssysteme in gehaltvoller Weise beschrieben sind, lassen sich sinnvolle Migrationsstrategien entwickeln.

IV. Ein Bezugsrahmen für die Gestaltung von Unternehmensmodellen

"Der Gegenstand des Denkens wird fortschreitend deutlicher durch die Vielfalt der Perspektiven, die sich auf ihn richten."

Peter L. Berger und Thomas Luckmann

"There are potentially at least as many ways of dividing up the world into object systems as there are scientists to undertake the task."

Clyde H. Coombs, Howard Raiffa, Robert M. Thrall

Die bisherige Untersuchung hat ergeben, daß fortschrittliche Analyse- und Entwurfstechniken - wie etwa objektorientierte Ansätze - die *softwaretechnische* Qualität betrieblicher Informationssysteme zu verbessern versprechen. Sie geben allerdings nur eine geringfügige Unterstützung für die zum Entwurf unternehmensweit integrierter Informationssysteme notwendige Strukturierung bzw. Systematisierung. Die oben dargestellten Unternehmensmodelle oder Informationssystem-Architekturen zielen darauf, dem Entwickler (von Werkzeug- wie auch von Anwendungssystemen) sowie anderen Beteiligten eine angemessene Strukturierung zu bieten. Dies geschieht durch die Abgrenzung und weitere Differenzierung sogenannter "Views" oder "Sichten". Der in diesem Kapitel zu entwickelnde Bezugsrahmen ist darauf gerichtet, eine Alternative für den Entwurf von Unternehmensmodellen zu bieten, die in einigen Punkten über CIM-OSA, ARIS und den Vorschlag von Sowa und Zachman hinausweist. Um die wichtigsten Charakteristika dieser Alternative zu nennen:

- *Betonung von Perspektiven*: Wie in II.3 gefordert, sollten Unternehmensmodelle ein Medium liefern, das dazu beiträgt, die durch unterschiedliche Auffassungen, Wahrnehmungen, Prioritäten und dergleichen verursachten Sprachbarrieren zu überwinden. Das Konzept der Perspektive wird in einer Akzentuierung verwendet, die diesem Umstand besser gerecht zu werden verspricht als die Unterscheidung von Sichten, wie sie in anderen Ansätzen vorgenommen wird.
- *Dediziertere Berücksichtigung betriebswirtschaftlicher Zusammenhänge*: In bisherigen Ansätzen zur Unternehmensmodellierung wird die Bedeutung betriebswirtschaftlicher Zusammenhänge zwar mehr oder weniger nachhaltig betont, eine spezifische Rekonstruktion solcher Zusammenhänge unterbleibt jedoch.
- *Konsequente Objektorientierung*: Die dargestellten Modelle bzw. Architekturen sind - wegen ihrer Orientierung an gegenwärtig vorherrschenden Randbe-

dingungen mit gutem Grund - eher daten- als objektorientiert. Das Bemühen um eine durchgängige objektorientierte Modellierung soll dazu beitragen, die Potentiale eines solchen Vorgehens für den zukünftigen Einsatz zu erarbeiten. Gegenwärtige Methoden zur objektorientierten Modellierung präsentieren sich weder in einheitlicher Form noch sind sie über jede Kritik erhaben. Es wird deshalb eine eigene objektorientierte Analyse- und Entwurfsmethode vorgestellt, die anders als generelle Methoden speziell in den Kontext der Unternehmensmodellierung eingebunden ist.

- *Integration der Perspektiven*: Die Vermittlungsfunktion eines multiperspektivisches Unternehmensmodells hängt wesentlich davon ab, in welcher Weise die einzelnen Perspektiven integriert sind. Das dazu nötige gemeinsame semantische Referenzsystem wird in Form eines Metamodells dargestellt.
- *Werkzeugorientiert*: Die Definition des Metamodells ist nicht zuletzt darauf ausgerichtet, eine gehaltvolle Grundlage für den Entwurf von Entwicklungsumgebungen zu liefern, die die verschiedenen Perspektiven gemeinsam zu behandeln gestatten.
- *Evolution von Unternehmensmodellen*: Das Bemühen um ökonomische Wiederverwendbarkeit legt es nahe, Referenzmodelle für möglichst viele Unternehmen zu entwickeln. In VI wird skizziert, wie die damit verbundenen Herausforderungen angegangen werden können.

1. Abgrenzung des Gegenstands

Die skizzierten Merkmale legen eine Reihe von Fragen nahe. Zunächst ist vor allem zu klären, wie der Begriff "Perspektive" (oder auch "Sicht") in der hier vertretenen Terminologie zu profilieren ist. Anschließend sind - unter Berücksichtigung von in der Literatur verwendeten Perspektiven auf das Unternehmen bzw. auf betriebliche Informationssysteme - diejenigen Perspektiven auszuwählen und zu konkretisieren, die hier für die Unternehmensmodellierung vorgeschlagen werden.

1.1 Zur Verwendung des Begriffs "Perspektive"

Die in ARIS, CIM-OSA oder von Sowa und Zachman unterschiedenen Sichten sind darauf gerichtet, Ausschnitte oder Aspekte des Unternehmens zu identifizieren, denen besondere Aufmerksamkeit zuteil werden sollte. In diesem Sinn sind die vorgeschlagenen Differenzierungen durchaus überzeugend: Es läßt sich kaum leugnen, daß die einzelnen Sichten (wie etwa Organisationssicht oder Funktionssicht) bedeutend sind. Die Plausibilität der drei unterschiedlichen Ansätze gibt gleichzeitig einen Hinweis darauf, daß die Abgrenzung von Sichten kaum nach eindeutigen intersubjektiv akzeptierten Kriterien erfolgen kann. Für unsere

Betrachtung ist aber vor allem bedeutsam, wie die jeweils unterschiedenen Sichten präsentiert werden. Dabei fällt auf, daß in allen Ansätzen die verschiedenen Sichten nach jeweils gleichartigen Merkmalen strukturiert werden - wodurch sich eine Matrix- bzw. Würfeldarstellung ergibt.

Eine solche Matrix-artige Differenzierung verspricht gewichtige Vorteile. Zum einen ist der Gesamtaufbau eines Modells besser darstellbar und damit leichter vermittelbar. Schwerer als dieser didaktische Vorteil wiegt jedoch der Umstand, daß eine Matrixdarstellung eine Unterteilung in Komponenten oder abstrakter: "building blocks" fördert. Solche "building blocks" werden in allen Sichten in jeweils unterschiedlicher Abstraktion verwendet. Damit ist die wichtige Frage nach den Beziehungen zwischen den Teilsichten und möglichen Transformationen weitgehend beantwortet. Dennoch sind solche Matrixdarstellung mit Problemen behaftet: Die jeweils gleichartige Differenzierung von Sichten wirkt mitunter aufgesetzt. So reflektiert die in ARIS vorgeschlagene Unterteilung der Organisationssicht in "Fachkonzept", "DV-Konzept" und "Implementierung" gewiß keine typischen Sichten auf die Unternehmensorganisation. Ähnliches gilt für die Behandlung der Organisationssicht in CIM-OSA oder bei Sowa und Zachman.

Eine allein analytisch motivierte Abgrenzung und Strukturierung von Sichten führt darüber hinaus leicht zu einer wenig übersichtlichen Vielfalt von Teilsichten (CIM-OSA: 36; Sowa/Zachman: 30). Zusammen mit der teilweise künstlich wirkenden Strukturierung von Sichten trägt diese Vielfalt dazu bei, daß die am Entwurf, der (organisatorischen) Implementierung und der Nutzung von Informationssystemen beteiligten Personen Schwierigkeiten haben dürften, die ihren Vorstellungen am besten entsprechende Sicht zu finden. Der Grund dafür liegt darin, daß die in den dargestellten Ansätzen vorgeschlagenen Differenzierungen weniger darauf ausgerichtet sind, die Vorstellungen der verschiedenen Beteiligten zu berücksichtigen, als vielmehr daran, das Unternehmen aus der Sicht von Systementwicklern zu strukturieren. Im Unterschied dazu werden wir die interpersonellen Unterschiede bei der Betrachtung von Unternehmen besonders betonen. Eine Perspektive ist danach weniger als ein bestimmter Ausschnitt zu kennzeichnen, sondern vielmehr als ein Konglomerat von *Auffassungen*, P*rädispositionen* und *Präferenzen*, die die Wahrnehmung einzelner Personen oder Personengruppen in bestimmten Kontexten wesentlich bestimmen. Eine Perspektive beschreibt also einerseits eingeschränkte, subjektiv verfärbte Konzeptualisierungen von Realität, andererseits ist sie - positiv gewendet - Ausdruck einer notwendigen Reduktion von Komplexität. In diesem Sinne stellt Luhmann (1977, S. 182), in Anlehnung an die Husserl'sche Phänomenologie, fest:

> "Grundlegend vereinfacht sich das System seine Umweltsituation dadurch, daß es die *objektive Situation durch eine subjektive ersetzt*, das heißt sein Handeln nicht unmittelbar durch die Wirklichkeit bestimmen läßt, sondern es nach seiner Vorstellung von der Wirklichkeit ausrichtet. Die unfaßbare Komplexität der Welt wird dadurch in eine Per-

spektive gefaßt, kann in Ausschnitten Erlebnisthema werden in der Form einer immer schon bestimmten und weiter bestimmbaren Unbestimmtheit."

Die folgende Verwendung des Begriffs "Perspektive" (wahlweise auch als "Sicht" bezeichnet) ist durch diese Akzentuierung geprägt, schließt dabei aber die oben dargestellte, im Bereich des Systementwurfs übliche Bedeutung nicht aus. Ein Vertriebsmanager hat in diesem Sinn eine andere Perspektive auf die Zusammenhänge im Unternehmen als ein DV-Leiter. Das äußert sich eben nicht allein in der Auswahl bestimmter Ausschnitte des Unternehmens (die können in beiden Fällen gleich sein), sondern in einer - um es tiefsinnig zu wenden - unterschiedlichen "Weltanschauung".

Wir suchen also nach Perspektiven auf das Unternehmen, die geeignet sind, individuellen Auffassungen zu entsprechen (um durch die Integration der Perspektiven zwischen ihnen vermitteln zu können). Angesichts der offenkundigen Vielfalt solcher Auffassungen und ihrer mitunter subtilen Konturen ist es für den Zweck der Unternehmensmodellierung nicht sinnvoll, möglichst viele individuelle Perspektiven im Detail zu erfassen. Außerdem dienen Perspektiven nicht allein der Abbildung vorhandener Sichten. Sie sind zwar an ihnen orientiert, sollten aber durchaus auch analytischen Anforderungen an die Betrachtung von Unternehmen gerecht werden. Diese Überlegungen sprechen dafür, prototypische Sichten oder Perspektiven zu suchen, deren Darstellung den vorherrschenden Realitätskonzeptualisierungen bedeutender Gruppen weitgehend entsprechen. Um eine der Anschaulichkeit von Modellen abträgliche Vielfalt zu vermeiden, sollte die Zahl der Perspektiven klein gehalten werden. Damit stellen sich zwei konkrete Fragen:

- Welche Perspektiven auf das Unternehmen bieten sich neben einer vorrangig DV-orientierten Sicht für die Modellierung im Kontext unserer Untersuchung an?
- Wie können sie zum Zweck der Modellierung rekonstruiert werden?

1.2 Die Kontingenz von Perspektiven

Perspektiven auf das Unternehmen können in analytischer Absicht als bewußte Abstraktion von irrelevanten Aspekten explizit eingeführt werden, sie können aber auch mehr oder weniger unbewußt durch berufliche Spezialisierung, individuelle Erfahrungen, Präferenzen und Prädispositionen entstehen. Das Bemühen um Anschaulichkeit empfiehlt die Berücksichtigung faktisch vorfindbarer Perspektiven. In der Literatur zur Systementwicklung und zur organisatorischen Implementierung findet sich eine Reihe von Ansätzen zur Differenzierung solcher Perspektiven. Die zum Teil subtile Vielfalt individueller Perspektiven wird dabei durch die Einführung prototypischer, an bestimmte Rollen- oder Berufsbilder geknüpfter Perspektiven eingeschränkt. Olle/Hagelstein et al. (1988, S. 7)

unterscheiden neun Rollen. Dazu zählen unter anderem:

- executive responsible
- development coordinator
- business analyst
- designer
- user
- constructor

Wollnik (1986, S. 227 f.) differenziert zehn Rollen von am Prozeß der organisatorischen Einführung von Informationssystemen beteiligten Personen. Dazu gehören unter anderem:

- "obere Führungskräfte"
- "Leiter von Ressorts, Fachabteilungen oder kleineren Organisationseinheiten, in deren Zuständigkeit das zu implementierende Informationssystem fällt ..."
- "Mitarbeiter, von denen Informationsverarbeitungs- oder Informationsbearbeitungsleistungen im Rahmen des Informationsverarbeitungsverfahrens erwartet werden ..."
- "Systemplaner"
- "Programmierer"

Wollnik beschreibt die Perspektiven allerdings nicht näher. Er faßt sie vielmehr zu drei Hauptsichten ("Benutzerbereichsleiter", "Informationsbenutzer", "Systembediener") zusammen, die dann im Rahmen einer empirischen Untersuchung tatsächlicher Implementierungsprozesse näher beleuchtet werden.

In der anwendungsorientierten Informatik gibt es eine Reihe von Ansätzen, die Untersuchung und Modellierung von Domänen durch die Einführung von Perspektiven anzuleiten. Sie sind weniger an bestimmten Rollen orientiert als daran, einen systematischen Entwurf zu unterstützen. In Kapitel III wurden die wichtigsten Perspektven der konzeptuellen Modellierung ausführlich behandelt. Beispielhaft seien darüber hinaus die Differenzierungen angeführt, die Bracchi/Pernici (1984) in einem Übersichtsartikel für die Büroautomatisierung nennen. Sie unterscheiden einerseits drei grundlegende Entwurfsperspektiven: eine technische, eine organisatorische und eine sozio-technische. Daneben unterscheiden sie fünf Ansätze für die Rekonstruktion der Perspektiven: "procedural models, "information flow models", "database models", "decision making models", "behavioral models".

Die Einbettung von Informationssystemen in ein Unternehmen empfiehlt die Berücksichtigung von Zielen, Handlungsspielräumen und wesentlichen Bewertungskriterien - kurz: einer mehr oder weniger differenzierten betriebswirtschaft-

lichen Perspektive. Die drei oben beschriebenen Architekturen bzw. Bezugsrahmen versuchen allesamt, diesem Umstand Rechnung zu tragen. Dabei beschränken sie sich allerdings auf die rudimentäre Darstellung einer "Organisationssicht", die von Sowa und Zachman noch um eine, ebenfalls nicht im Detail beschriebene strategische Sicht ergänzt wird.

Ein Blick auf die Betriebswirtschaftslehre eröffnet eine Reihe differenzierterer Sichten auf das Unternehmen. Die Unterteilung in Sach- oder Funktionsbereiche (wie "Beschaffung", "Produktion", "Finanzierung" und dergleichen) reflektiert traditionelle - und in Teilen von der Betriebswirtschaftslehre selbst geprägte - Formen der Arbeits- bzw. Kompetenzteilung. Daneben gibt es Perspektiven, die bestimmte Abstraktionen des gesamten Unternehmens empfehlen, um eine zweckbezogene systematische Betrachtung zu ermöglichen. Das gilt beispielsweise für das Marketing als "Lehre der marktorientierten Unternehmensführung" oder in jüngerer Zeit für das Informationsmanagement, vor allem aber für die Sicht der Organisationstheorie, die selbst wieder - in unterschiedlicher Weise - differenziert wird (wie etwa in Aufbauorganisation, Ablauforganisation, formale und informale Organisation). Andere Unterscheidungen zielen auf eine Klassifizierung von Handlungen - wie etwa die in "Planung - Ausführung - Kontrolle". Ein weiterer Ansatz zur Bildung von Perspektiven ist in der - häufig didaktisch motivierten - Fokussierung auf bestimmte Teilaspekte zu sehen, wie sie sich beispielsweise in Management-By-Konzepten artikuliert. Auf methodologischer Ebene gibt es darüber hinaus eine Reihe von - mehr oder weniger elaborierten - Perspektiven für die wissenschaftliche Annäherung an Unternehmen. Um einige Orientierungen zu nennen: Verhaltenstheorie, Systemtheorie, Handlungstheorie, Entscheidungstheorie.

Die Liste der skizzierten Perspektiven auf das Unternehmen ist gewiß nicht vollständig. Sie deutet allerdings an, daß Perspektiven - als mehr oder weniger exakt beschriebene Abstraktionen des Unternehmens - in vielfältiger Weise möglich sind. Ihre Festlegung hängt einerseits davon ab, welche Untersuchungs- oder Handlungsziele jeweils im Vordergrund stehen. Andererseits spielen allerdings auch - was sich nicht zuletzt in der mitunter zu verzeichnenden Apologetik methodologischer Auseinandersetzungen spiegelt - individuelle Prädispositionen eine Rolle. Wir können also sagen, daß Perspektiven auf das Unternehmen in der Regel kontingent sind: Sie sind (denk-) möglich, aber nicht notwendig.

1.3 Auswahl der Perspektiven

Die skizzierten, gewiß nicht vollständigen, Beispiele für Perspektiven auf das Unternehmen machen deutlich, daß die Auswahl von Perspektiven - so sorgfältig sie begründet sein mag - nie ganz frei sein kann von subjektiven Attitüden. Um diesem Umstand Rechnung zu tragen, empfiehlt es sich einerseits, solche Per-

spektiven zu wählen, die etabliert und damit "wirklich" sind, das heißt "von einem größeren Kreis von Personen für relevant gehalten" (Wollnik 1986, S. 44) werden. Andererseits sollten die eine Perspektive kennzeichnenden Abstraktionen in dem Sinne flexibel sein, daß den Verwendern der unter Rückgriff auf diese Perspektiven entworfenen Modelle Raum für individuelle Ausdifferenzierungen gelassen wird.

Vor dem Hintergrund der in II.1.1 dargestellten Anforderungen sind neben der Integration der Komponenten eines Informationssystems vor allem dessen Integration in die Unternehmensorganisation sowie dessen Abstimmung mit der strategischen Planung von herausragender Bedeutung. In Anlehnung an diese Dimensionen von Integration beschränken wir uns im folgenden auf drei Perspektiven:

- Die *strategische Perspektive* ist auf die Formulierung von Unternehmenszielen, den Entwurf und die Bewertung langfristiger Strategien gerichtet. Es ist also keine Eingrenzung auf die strategische Planung von Informationssystemen intendiert.
- Die *organisatorische Perspektive* behandelt die Gestaltung und Durchführung der kooperativen Handlungen im Unternehmen.
- Die *Informationssystem-Perspektive* ist auf den Entwurf, die Implementierung und den Betrieb von Informationssystemen gerichtet.

Es ist wohl unstrittig, daß diese drei Perspektiven für den Entwurf, die (organisatorische) Implementierung und die wirtschaftliche Nutzung von Informationssystemen von wesentlicher Bedeutung sind. Darüber hinaus sind die Interdependenzen zwischen den Perspektiven unverkennbar, ihre Berücksichtigung in einem integrierten Modell also wünschenswert.

Es handelt sich allerdings nicht allein um eine analytisch motivierte Differenzierung. Vielmehr können den einzelnen Sichten ohne Schwierigkeiten existierende Rollen- bzw. Berufsbilder zugeordnet werden (wobei natürlich Überschneidungen nicht ausgeschlossen sind). Im Unterschied zu den dargestellten Matrix-artigen Strukturierungen werden die drei Perspektiven in einer für die jeweilige Perspektive angemessenen Weise modelliert. Im Hinblick auf die Integration der Perspektiven ist es dabei erstrebenswert, diese spezifischen Detaillierungen nicht völlig unabhängig voneinander durchzuführen, sondern mögliche Korrespondenzen deutlich zu machen.

Um die Perspektiven (im folgenden zum Teil auch als "Ebenen" bezeichnet) zur Modellbildung zu verwenden, ist es erforderlich, die mit ihnen verbundenen Abstraktionen zu beschreiben. Das geschieht in Form charakteristischer Begriffe (oder Konzepte) und den zwischen ihnen bestehenden Beziehungen. In diesem Sinne Wollnik (1986, S. 42 f.):

"Jede Perspektive läßt sich an einem eigenen Sprachstil, an besonderen Schlüsselbegriffen, an typischen Etikettierungen und Darstellungen, kurz: an einer *spezifischen Rhetorik* kommunikativ erkennbar machen. ... Die Auffassungen reflektieren ein bestimmtes praktisches Eingestelltsein auf Informationssysteme. Sie sind insbesondere auf unterschiedliche Interessen, Handlungsabsichten, Aufträge, Berührungspunkte usw., mit einem Wort: *auf unterschiedliche pragmatische Relevanzen* zurückzuführen."

Zunächst geht es uns dabei nicht - wie von Wollnik intendiert - um eine empirisch fundierte sprachliche Rekonstruktion. Es wird allerdings noch darauf einzugehen sein, in welcher Weise faktische Auffassungen berücksichtigt werden können.

In Abbildung 24 ist skizziert, wie die drei Perspektiven mit Hilfe semantischer Netze konkretisiert werden könnten. Es liegt auf der Hand, daß es zwischen den drei Ebenen vielfältige Beziehungen gibt. Dabei sollte tendenziell die beeinflussende Wirkung von oben nach unten stärker ausgeprägt sein. Auch wenn die Konkretisierung einer Ebene in der Regeln nicht aus einer darüber liegenden auf formal-logischem Weg abgeleitet werden kann, so ist doch unstrittig, daß eine Begründung oder vielleicht besser: Sinnattribuierung wünschenswert ist. Die Beschreibung der Objekte eines Informationssystems erhält Sinn durch die Verwendung dieser Objekte auf der organisatorischen Ebene. Die Ausgestaltung der organisatorischen Ebene wiederum sollte sich unter Hinweis auf übergeordnete strategische Richtlinien begründen lassen. Davon abgesehen scheint es realistisch, von wechselseitigen Abhängigkeiten auszugehen (die in Abb. 24 nur angedeutet sind): Jede Sicht ist für die andere von Bedeutung und weist gleichzeitig spezifische Randbedingungen auf, die von außen nur teilweise beeinflußt werden können.

Die Strukturierung der einzelnen Sichten wird im folgenden für jede Sicht entwickelt. Erst danach werden die Beziehungen zwischen den Sichten behandelt. Angesichts der Komplexität der einzelnen Ebenen wäre es vermessen, Vollständigkeit anzustreben. Vielmehr soll die Detaillierung der Perspektiven und der zwischen ihnen bestehenen Zusammenhänge dazu beitragen, eine diskursfähige Vorstellung von multiperspektivischen Unternehmensmodellen zu vermitteln. Auf diese Weise soll nicht zuletzt verdeutlicht werden, wie die dargestellten Modelle weiterentwickelt werden können.

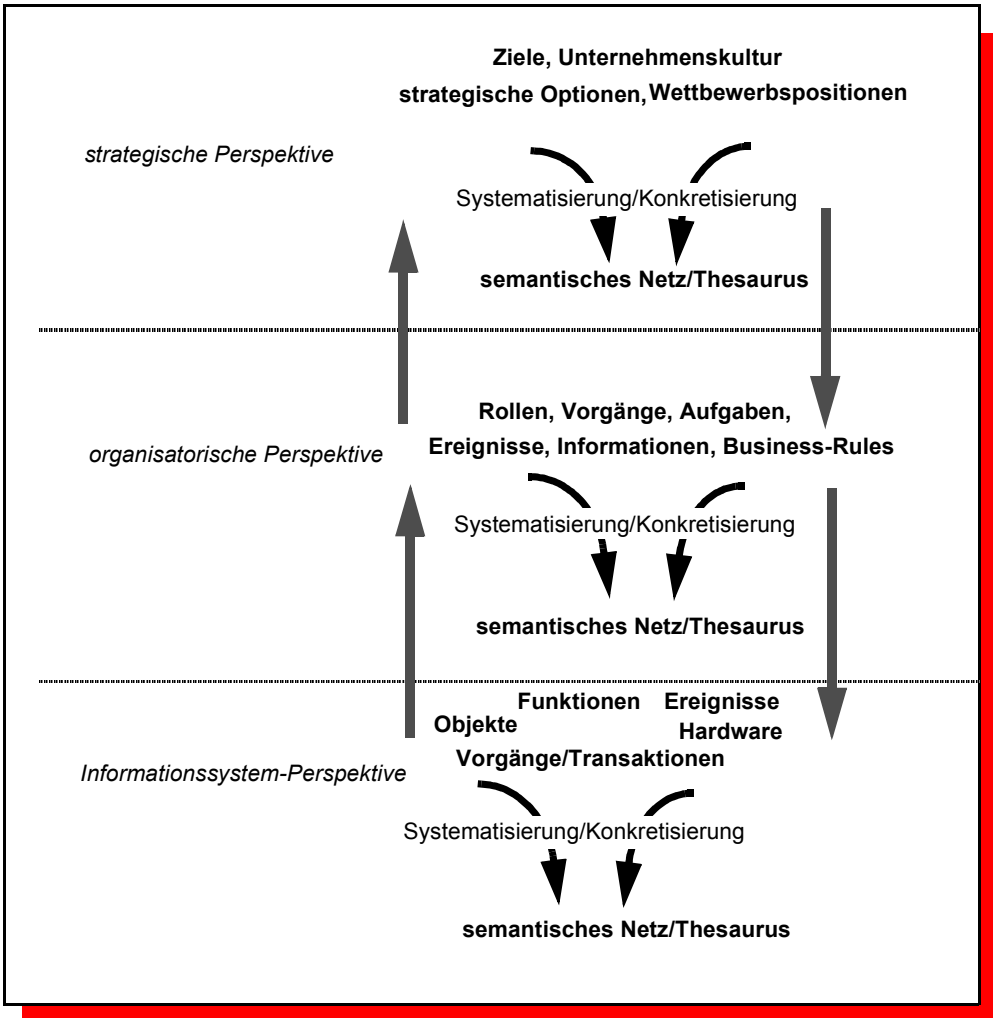


Abb. 24: Beispielhafte Konkretisierung von Perspektiven durch Einführung zentraler Begriffe.

2. Die Informationssystem-Perspektive

Die Bedeutung der drei ausgewählten Perspektiven hängt - dies ist eine inhärente Eigenschaft des hier vertretenen Perspektivenbegriffs - vor allem vom jeweiligen Betrachter ab. Auch wenn deshalb im folgenden alle drei Perspektiven sorgfältig untersucht werden, hat für uns dennoch die Informationssystem-Perspektive eine herausragende Bedeutung. Das liegt zum einen daran, daß der Entwurf, die Implementierung und die organisatorische Einführung von Informationssystemen den Ausgangspunkt unserer Untersuchung markieren.

In einem Unternehmen gibt es - vor allem dann, wenn auch Software-Entwicklung betrieben wird - kaum eine einheitliche Sicht auf das Informationssystem.

Unter den vielen möglichen Konzeptualisierungsmustern, mit denen Informationssyteme von den verschiedenen Betrachtern wahrgenommen werden, werden im folgenden zwei prototypische Perspektiven ausgewählt. Diese Perspektiven werden dann allerdings nicht so beschrieben, daß sie gängigen Sichtweisen entsprechen. Vielmehr wird eine Vorstellung von Informationssystemen entwickelt, die unter anderem durch eine konsequente Objektorientierung gekennzeichnet ist. Daraus ergeben sich Darstellungen der zwei Sichten, die als Vorschläge für eine Neuorientierung zu verstehen sind:

- Die Perspektive der *Entwickler* wird gekennzeichnet durch konzeptuelle Modelle des Informationssystems.
- Die Sicht der *Informationssystem-Manager* (wie beispielsweise DV-Leiter, aber auch Informationsmanager und andere) ist geprägt durch die Architektur und die für die Verwaltung von Hardware und Software wichtigen Zusammenhänge. Dazu gehört auch eine gewandelte Vorstellung von Anwendungen und Informationsobjekten.

Es liegt auf der Hand, daß es sich dabei um eine analytisch motivierte Differenzierung handelt. Realistischerweise können die zwei Teilperspektiven den einzelnen Rollen kaum überschneidungsfrei zugeordnet werden.[1] Darüber hinaus bestehen enge Beziehungen zwischen den jeweils verwendeten Konzepten.

Die zwei skizzierten Sichten auf das Informationssystem werden im folgenden detailliert dargestellt. Anschließend werden in einem kurzen Szenario die Implikationen für die Anwender beschrieben.

2.1 Das konzeptuelle Metamodell

Im Hinblick auf Ziele wie softwaretechnische Integration, Wartbarkeit und Portierbarkeit kommt dem konzeptuellen Modell[2] eines Informationssystems herausragende Bedeutung zu (vgl. III.2). Für unsere Betrachtung gilt dies auch in anderer Hinsicht: Die werkzeuggestützte Erstellung und Verwaltung von Unternehmensmodellen erfordert die Beschreibung aller dabei berücksichtigten Perspektiven mit den Ausdrucksmitteln der konzeptuellen Modellierung.

Die bisherige Untersuchung hat gezeigt, daß eine objektorientierte Modellierung

1. Dies gilt auch für das Verhältnis der Informationssystem-Perspektive zu der organisatorischen und strategischen Perspektive: So mag ein Anwender eine Sicht auf das Unternehmen haben, die in Teilen sowohl der organisatorischen als auch der strategischen Perspektive entspricht.
2. Der Begriff "konzeptuelles Modell" wird hier in der im Software-Engineering üblichen Weise verstanden. Dieser Umstand ist zu betonen, da die Modellierung aller in dieser Arbeit berücksichtigten Teilsichten in dem Sinne konzeptuell ist, daß reale Sachverhalte mit Hilfe geeigneter Abstraktionen auf Konzepte des Modells abgebildet werden.

gegenüber konventionellen Ansätzen - jenseits gegenwärtiger Restriktionen - deutliche Vorteile verspricht. Die hier vorgeschlagene Entwickler-Sicht wird deshalb im folgenden mit Hilfe eines objektorientierten Modellierungsansatzes beschrieben. Er ist einerseits durch das Bemühen gekennzeichnet, die Vorteile bisheriger Ansätze - hier ist vor allem an die oben analysierten Arbeiten von Booch und Rumbaugh et al. zu denken - zu vereinen. Andererseits weicht er in einigen Punkten von diesen Ansätzen ab, geht über sie hinaus. So werden Aspekte der Benutzerschnittstelle in die Modellierung mit einbezogen. Dadurch werden nicht zuletzt bessere Voraussetzung für ein schnelles Prototyping geschaffen. Daneben wird die vorwiegend an den Anforderungen der Implementierung orientierte Darstellung von dynamischen und funktionalen Modellen durch ein Konzept für die Modellierung von Vorgängen ersetzt. Es ist deutlich anschaulicher und integriert Aspekte der dynamischen und der funktionalen Modellierung.

Die im folgenden entwickelten Modelle stellen nur in Teilen eine konzeptuelle Beschreibung eines Realitätsausschnitts dar. Daneben sind sie vor allem Modelle der zur Modellierung verwendeten Konzepte oder - anders formuliert - Metamodelle. Einzelne Teile der Metamodelle werden in V.2 zum Zweck des Werkzeugentwurfs noch weiter differenziert.

2.1.1 Das Objektmodell

Die Grundlage eines objektorientierten konzeptuellen Modells bildet das Objektmodell. Ein Objektmodell besteht aus der Beschreibung von Objekten und den zwischen ihnen bestehenden Beziehungen. Da in der konzeptuellen Modellierung grundsätzlich von konkreten Exemplaren eines Realitätsbereichs abstrahiert wird, werden genau genommen Klassen beschrieben. Diese triviale Feststellung wird allerdings für die Modellierung von Beziehungen zu differenzieren sein (vgl. auch III.3.4).

2.1.1.1 Die Konzeptualisierung von Objekten

Objekte sind Beschreibungen realweltlicher Gegenstände oder Sachverhalte. Eine Klasse generalisiert über mögliche Eigenschaften einer Menge von Objekten. Dabei kann es sich um alle invarianten Eigenschaften einer Menge von Objekten oder Teilen davon handeln.[1] Die Einschränkung, daß nicht über alle Objekteigenschaften generalisiert werden muß, bezieht sich auf abstrakte Klassen. Abstrakte Klassen sind solche Generalisierungen über Klassen, für die es selbst keine Instanzen gibt. So könnte über die Klassen "Rundfunkgerät", "Fernsehgerät" und dergleichen mit Hilfe der Klasse "Gerät der Unterhaltungselektro-

1. Vgl. dazu die davon abweichende Definition in Booch (1990, S. 93).

nik" abstrahiert werden, ohne daß für diese Klasse Instanzen erlaubt sind.

Eine Klasse kann ihre Eigenschaften an Unterklassen vererben. Sie selbst kann von einer oder mehreren Oberklassen erben. In dem hier vorgestellten Metamodell wird Einfachvererbung favorisiert. Es gibt eine Fülle von Argumenten für und gegen Mehrfachvererbung. So bietet Mehrfachvererbung ohne Zweifel reichhaltigere Möglichkeiten zur Generalisierung. Multiple Generalisierungen erhöhen allerdings die Komplexität von Modellen und führen zu Schwierigkeiten bei ihrer Interpretation - mit entsprechend negativen Konsequenzen für die Integrität. Falls mehrfache Generalisierungen möglich sind, kann die Beschränkung auf Einfachvererbung zu unangenehmer konzeptueller Redundanz führen. Diese Beschränkung kann allerdings dadurch relativiert werden, daß ein Objekt durch Beziehungen zu anderen Objekten gekennzeichnet wird. Um ein Beispiel zu geben: Ein Mitarbeiter, der gleichzeitig Führungskraft und Systementwickler ist, könnte bei Mehrfachvererbung durch die (artifiziell wirkende) Klasse "Führungskraft/Systementwickler" gekennzeichnet werden. Im Falle der Mehrfachvererbung könnte der Mitarbeiter als Instanz der Klasse "Mitarbeiter" abgebildet werden, die in Beziehung zu Rollenobjekten wie "Systementwickler" gesetzt wird.[1] Die Entscheidung für Einfachvererbung ist deshalb eher im Sinne einer vorläufigen Präferenz als einer strengen Apologetik zu verstehen.

Im Hinblick auf die Semantik der Vererbung stellt sich die Frage, ob geerbte Eigenschaften modifiziert werden können. Während das Überschreiben geerbter Eigenschaften grundsätzlich die Flexibilität der Spezialisierung erhöht ist es gleichzeitig mit konzeptionellen Schwächen verbunden: Wenn wesentliche Eigenschaften einer Klasse in einer Unterklasse überschrieben werden, gilt nicht mehr, daß jede Instanz der Unterklasse zugleich Instanz der Oberklasse ist.[2] Mitunter wird in diesem Zusammenhang auch zwischen Vererbung und "Subtyping"[3] unterschieden. Danach ist im Falle der Vererbung eine Modifikation von Eigenschaften erlaubt, während Subtyping eine solche Modifikation ausschließt, aber gleichzeitig garantiert, daß eine Instanz einer Klasse auch gleichzeitig Instanz ihrer Oberklasse ist. In dem hier vorgestellten Ansatz wird (vorläufig) ein

1. Dieser natürlicher erscheinende Ansatz ist allerdings auch nicht frei von Problemen. So werden die Vorteile der Typisierung teilweise preisgegeben: Wenn für einen bestimmten Verarbeitungskontext Objekte benötigt werden, die Führungskräfte repräsentieren, kann die damit verbundene Integritätsbedingung erst während der Laufzeit überprüft werden.
2. Dieser Umstand kommt bei nicht typisierten Programmiersprachen wie Smalltalk besonders drastisch zum Ausdruck. Hier kann durch Überschreiben sämtlicher Methoden (auch der Instanzierungsmethoden für die Attribute) eine Klasse eine völlig andere Semantik als ihre Oberklasse erhalten.
3. So in der von der OMG (1992) vorgelegten Methode für die Konzeptualisierung von Objekten.

Kompromiß favorisiert, dessen Qualität nicht zuletzt davon abhängt, wie die mit ihm verbundene Gestaltungsfreiheit beim Modellieren genutzt wird. Danach können von den geerbten Eigenschaften nur Dienste überschrieben werden.[1] Dabei sollte allerdings darauf geachtet werden, daß auf diese Weise keine völlig neue Semantik des Dienstes entsteht. Vielmehr sollte - der Name des Dienstes bleibt ja bestehen - eine entsprechende, lediglich durch den spezialisierten Kontext verfeinerte, Bedeutung erhalten bleiben.

Es gibt eine Reihe softwaretechnischer Gründe dafür, Klassen selbst als Objekte zu betrachten.[2] So können bestimmte Werte, die für alle Objekte einer Klasse gelten, in einem Klassenobjekt verwaltet werden. Für die Instanzierung eines Objekts können verschiedene Methoden angegeben werden, die von dem zugehörigen Klassenobjekt ausgeführt werden (sie müssen ja logisch zwingend vor der Lebenszeit einer Instanz gestartet werden). Daneben bietet ein solches Klassenobjekt die Möglichkeit, seine Veränderungen im Zeitverlauf selbst zu verwalten - ein eleganter Ansatz für die Verwaltung von Schema-Versionen. Die Betrachtung einer Klasse als Objekt führt zwingend zu einer Klasse dieses Objekts, auch Meta-Klasse genannt. Eine Meta-Klasse erlaubt eine erhöhte Flexibilität ihrer (einzigen) Unterklasse. So kann die - ansonsten fixe - Semantik der Unterklasse durch verschiedene Instanzierungsmethoden verändert werden. Ein Beispiel dafür sind Zeitzonen-abhängige Instanzierungen für die Klasse "Zeit".

Während etwa in Smalltalk eine Meta-Klasse selbst keine Meta-Klasse haben darf, gibt es Ansätze, die eine nahezu grenzenlose Flexibilität von Klassen durch die Einführung beliebig vieler Stufen von Meta-Klassen erlauben.[3] Dem möglichen Flexibilitätsverlust zum Trotz ist die Betrachtung von Klassen als Objekte für die konzeptuelle Modellierung m.E. wenig geeignet: Eine wesentliche Stärke des objektorientierten Ansatzes liegt gerade darin, daß Objekte in anschaulicher Weise mit realweltlichen Sachverhalten korrespondieren. Die Betrachtung von Klassen - die ja Generalisierungen darstellen - als Objekte ist geeignet, diese Anschaulichkeit zu gefährden - von Meta-Klassen ganz zu schweigen. Mögliche Eigenschaften von Klassen als Objekte bleiben deshalb im folgenden unberücksichtigt.[4]

---

1. Daneben können auch Eigenschaften überschrieben werden, die für die Semantik einer Klasse nicht relevant sind, also der Default View oder das Etikett.
2. Wie es in einigen objektorientierten Sprachen, z.B. Smalltalk, vorgesehen ist.
3. Vgl. beispielhaft Klas/Neuhold/Schrefl (1989).
4. Damit ist nicht ausgeschlossen, daß zum Zweck der Implementierung Klassen als Objekte betrachtet werden. Tatsächlich wurde bei der Implementierung der in V dargestellten Entwicklungsumgebung so verfahren.

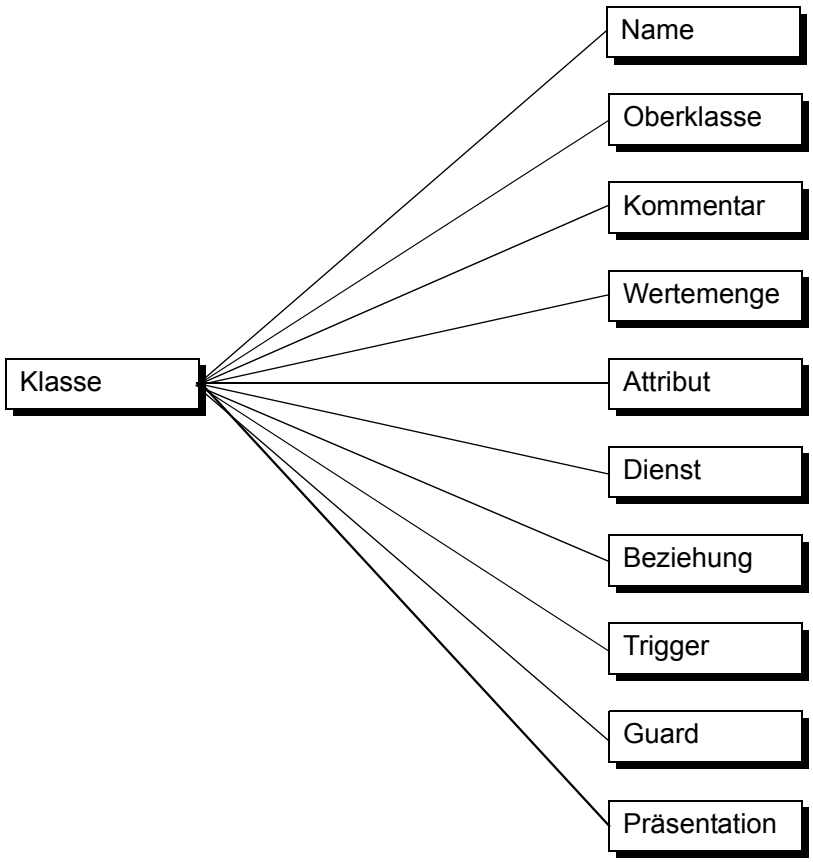


Abb. 25: Konzeptualisierung einer Klasse

Für die Beschreibung der Eigenschaften einer Klasse werden die in Abbildung 25 benannten Konzepte vorgeschlagen. Jenseits dieser Konzepte kann eine Klasse als abstrakte Klasse gekennzeichnet werden.

Während die Kardinalität von Name (Zeichenkette aus Literalen), Oberklasse (Name der Oberklasse) und Kommentar in min, max-Notation 1, 1 ist, gilt für alle anderen Konzepte die Kardinalität 0, *. Sie sind also optional, können aber in beliebiger Zahl verwendet werden. Die drei genannten Merkmale werden im Unterschied zu den im folgenden näher beschriebenen nicht an Unterklassen vererbt.

Wertemenge

Eine Klasse kann durch eine Wertemenge gekennzeichnet werden. In diesem Fall darf sie keine Attribute haben. Die Wertemenge kann entweder durch Aufzählung (wie etwa: rot, gelb, grün), durch die Festlegung eines oder mehrerer Intervalle oder durch dir mit Hilfe eines formalen Verfahrens beschriebene Teilmenge

einer anderen Menge (beispielsweise die Menge der geraden Zahlen als Teilmenge der natürlichen Zahlen) definiert werden.

2.1.1.1.1 Attribute

Die Attribute einer Klasse sind selbst wiederum durch eine Reihe von Eigenschaften gekennzeichnet (siehe Abb. 26). Im Unterschied zu Rumbaugh et al. wird die Semantik eines Attributs nicht durch einen Datentyp, sondern durch eine Klasse charakterisiert. Die konkreten Ausprägungen eines Attributs sind also selbst Objekte. Für diese schränken wir ein, daß sie keine Existenz unabhängig von dem Objekt, dem sie als Attribut zugeordnet sind, haben können.[1] Es kann zwar durchaus natürlich erscheinen, auch einen Verweis auf ein externes Objekt als Attribut zu betrachten (beispielsweise die Branche eines Unternehmens). Dadurch wird allerdings die wünschenswerte Korrespondenz der Klassen des konzeptuellen Modells mit denen der Implementierungsebene gefährdet: Für die Implementierung eines Objekts ist die Verkapselung seiner Attribute charakteristisch. Im Falle externer Objekte können allenfalls die Referenzen auf diese Objekte verkapselt werden.

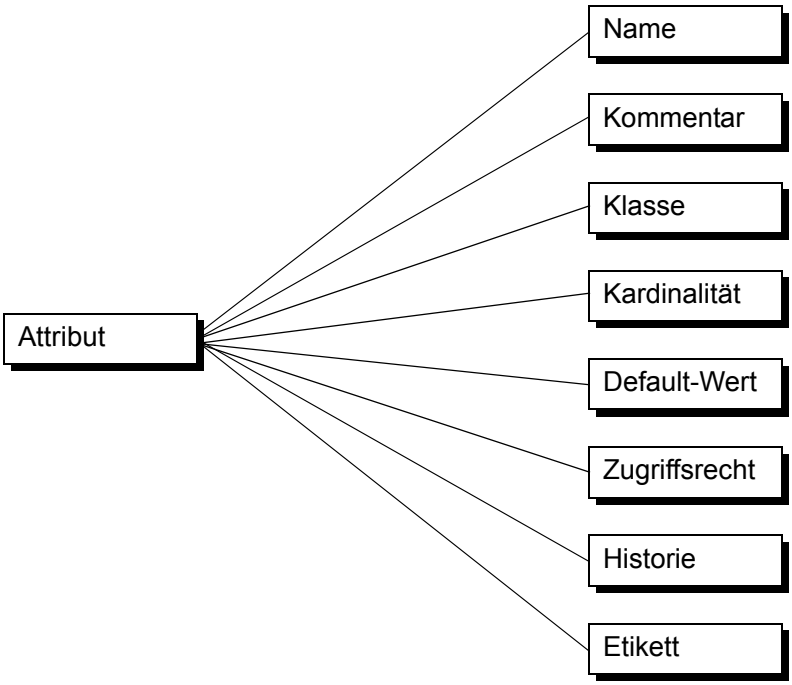


Abb. 26: Konzeptualisierung eines Attributs

1. Im Unterschied dazu schließen Velho/Carapuca (1992) solche Objekte nicht als Attribute aus.

Attribut-Klasse

Die Kennzeichnung eines Attributs durch eine Klasse anstatt durch einen Datentyp verspricht eine Reihe von Vorteilen. So wird eine realitätsnähere Modellierung unterstützt, wenn die Semantik eines Attributs durch eine anwendungsnahe Klasse charakterisiert werden kann. Wenn beispielsweise das Geburtsdatum einer Person mit Hilfe der Klasse "Datum" deklariert werden kann, werden auf diese Weise die Dienste dieser Klasse (wie etwa "alter in Jahren") mit übernommen - also auch ein Beitrag zur Wiederverwendbarkeit.

Die größere Ausdrucksmächtigkeit ist allerdings mit einer erheblich gestiegenene Komplexität verbunden. So können nicht alle Klassen eines Objektmodells für die Kennzeichnung des Attributs einer Klasse verwendet werden. Die Klasse selbst scheidet aus: Es wären sonst beim Instanzieren eines Objekts nicht auflösbare zyklische Verweise die Folge. Damit verbieten sich auch alle Unterklassen: Attribute werden an Unterklassen vererbt, dort wäre dann die Klasse eines Attributs identisch mit der Klasse des zugehörigen Objekts. Darüber hinaus wachsen die Anforderungen an die Wahrung der referentiellen Integrität des Modells: Wenn eine Klasse gelöscht oder geändert wird, kann davon eine Vielzahl von Attributen betroffen sein. Der damit verbundene Verwaltungsaufwand empfiehlt nachrücklich den Einsatz geeigneter Werkzeuge.

Attribut-Kardinalität

Durch die Kardinalität eines Attributs wird festgelegt, durch wieviele Objekte der Klasse des Attributs ein Attribut beschrieben werden darf. Dazu wird eine min, max-Notation verwendet. Die minimale Kardinalität gibt an, wieviele Objekte mindestens existieren müssen. Die maximale Kardinalität legt fest, wieviele maximal existieren dürfen. Der Wertebereich der minimalen Kardinalität ist festgelegt durch die natürlichen Zahlen, ergänzt um null. Wenn ein Attribut optional ist, hat es also eine minimale Kardinalität von 0. Der Wertebereich der maximalen Kardinalität ergibt sich aus den natürlichen Zahlen, wobei der Platzhalter "*" für "beliebig viele" verwendet werden kann.

Default-Wert

Einem Attribut kann ein Default-Wert (genauer: eine Instanz seiner Klasse) zugeordnet werden. Die Spezifikation eines solchen Wertes dient vor allem dem Anliegen, unzulässige Zustände bei der Instanzierung eines Objekts zu vermeiden. Die automatische Zuordnung eines Default-Werts beim Instanzieren eines Objekts kann durch das Generieren einer geeigneten Initialisierungsmethode erreicht werden.

Zugriffsrecht

Die Rechte, auf ein Attribut zuzugeifen, können in vielfältiger Weise kalibriert werden. In Abhängigkeit von der Semantik des Attributs können verschiedene

Zugriffsoperationen unterschieden werden, denen jeweils Rechte zugeordnet werden können - also beispielsweise: Überschreiben eines Werts, Verringern eines Werts, Erhöhen eines Werts etc. Daneben kann das Zugriffsrecht von der Autorisierung des zugreifenden Objekts abhängig gemacht werden. Da hier eine generalisierende Konzeptualisierung der Zugriffsrechte auf ein Attribut nötig ist, wird von Eigenheiten der Attribut-Klasse und des Nutzungskontextes abstrahiert. An anderer Stelle wird es Möglichkeiten zu einer weiteren Differenzierung geben.[1] Die einem Attribut zugeordneten Zugriffsrechte sollen nicht darüber hinwegtäuschen, daß der Verkapselung wegen grundsätzlich von außen nicht direkt auf ein Attribut zugegriffen werden kann. Es handelt sich hier um eine Auszeichnung, die durch die für den Entwurf gewählte Abstraktionsstufe nahegelegt wird. Im Hinblick auf die Implementierung werden die so zugeordneten Zugriffsrechte in Zugriffsrechte der entsprechenden Dienste zu transformieren sein. Es werden - in Anlehnung an Booch (1990) - drei Autorisierungsstufen eingeführt, die jeweils für lesenden und schreibenden Zugriff verwendet werden können: *public*, *protected* und *private*.

Public bedeuted, daß uneingeschränkter Zugriff möglich ist.[2] Im Falle einer Kennzeichnung als protected kann nur mittels einer geeigneten Autorisierung (von der hier abstrahiert wird) - also beispielsweise eines Codes - zugegriffen werden. Private schließlich heißt, daß von außen überhaupt nicht auf das Attribut zugegriffen werden kann. Dabei gilt, daß die Autorisierung für Schreibzugriffe nicht höher sein darf als die für Lesezugriffe. Auf diese Weise erhält man fünf unterschiedliche Autorisierungsstufen:

Lesen	Schreiben
public	public
public	protected
public	private
protected	protected
protected	private
private	private

Historie

Je nach Art eines Attributs kann es mehr oder weniger wichtig sein, die Veränderung seiner Zustände im Zeitverlauf aufzuzeichnen. Tendenziell nimmt diese Wichtigkeit mit der Häufigkeit der Veränderungen ab. So werden bei traditionell

1. Vgl. dazu die Spezifikation von Zugriffsrechten für Dienste.
2. Dabei wird unterstellt, daß die zur Gewährleistung der operationalen Integrität notwendige Synchronisation durch die Implementierung des Objekts bereitgestellt wird.

als "Bewegungsdaten" bezeichneten Attributen die verschiedenen Zustände im Zeitverlauf zumeist weniger wichtig sein als bei "Bestandsdaten". Wenn sich beispielsweise der Name eines Mitarbeiters ändert, gibt es gute Gründe dafür, den alten Namen weiterhin verfügbar zu halten.

Ein Attribut wird deshalb durch das boolesche Merkmal "Historie" gekennzeichnet. Wenn ihm der Wert "wahr" zugeordnet wird, können im Hinblick auf die Implementierung Verfahren generiert werden, die eine Aufzeichnung historischer Zustände ermöglichen[1] (sowie Verfahren, die den Zugriff auf diese Zustände erlauben).

Etikett

Ein besonderes Merkmal des hier vorgestellten Metamodells ist die Integration von Teilen der Benutzerschnittstelle. Sie erfolgt einerseits - wie noch näher zu erläutern sein wird - durch die Zuordnung sogenannter "default views" zu einer Klasse. Andererseits kann es wünschenswert sein, die Präsentation eines Attributs mit einem Hinweis auf seine Bedeutung, also einem Etikett, zu verbinden. Auch wenn der Name des Attributs möglichst selbstsprechend gewählt sein sollte, mag es sein, daß er für eine Erläuterung nicht geeignet ist. In solchen Fällen kann ein Etikett angegeben werden, das zum Zweck des Prototyping zusammen mit dem jeweiligen Zustand des Attributs präsentiert wird.

2.1.1.1.2 Dienste

Der wichtigste Unterschied zwischen Objekten und konventionellen Datenstrukturen ist darin zu sehen, daß Objekte Prozeduren bzw. Methoden beinhalten, die nach außen als Dienste angeboten werden. Zur Beschreibung von Methoden kann auf eine Vielzahl von Techniken des Programmentwurfs zurückgegriffen werden. Auf der Ebene der konzeptuellen Modellierung ist eine algorithmische Darstellung allerdings mit Nachteilen verbunden. So ist es sinnvoll, von der Spezifikation eines konkreten Algorithmus zu abstrahieren, da die Entscheidung über den jeweils besten Algorithmus an technische Randbedingungen gebunden sein kann, die sich im Zeitverlauf ändern mögen. Daneben ist zu berücksichtigen, daß auch die Präsentation eines Algorithmus mit Hilfe einer Darstellungstechnik häufig nicht die Forderung nach Anschaulichkeit erfüllt, die mit der Modellierung verbunden ist. Auch wenn die Möglichkeit algorithmischer Beschreibungen eines Dienstes nicht ausgeschlossen werden muß, sollten in jedem Fall andere Konstrukte zur Charakterisierung angeboten werden.

1. Ein in Smalltalk codiertes Beispiel findet sich in Frank/Hildebrand (1993).

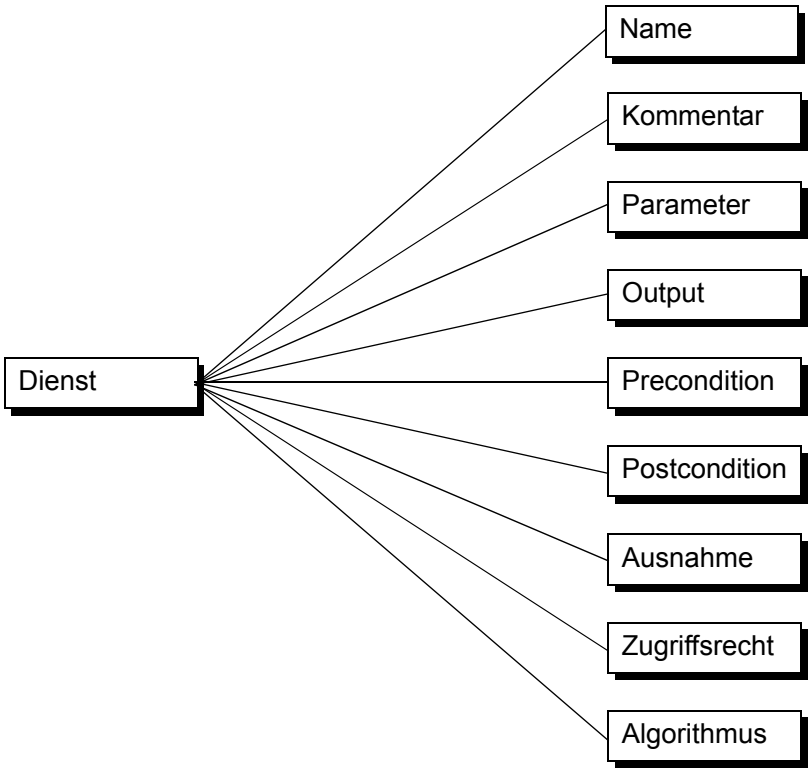


Abb. 27: Konzeptualisierung eines Dienstes

Abbildung 27 zeigt die hier vorgeschlagenen Konzepte. Da Polymorphie ermöglicht wird, muß der Name eines Dienstes nur innerhalb einer Klasse eindeutig sein.

Parameter

Nicht jeder Dienst erfordert einen Eingabeparameter. Falls eine Eingabe ermöglicht werden soll, kann eine Liste von Parametern angegeben werden. Jeder Parameter dieser Liste - es können beliebig viele Parameter genannt werden - wird durch eine Reihe von Eigenschaften gekennzeichnet. Neben der Klasse kann für einen Parameter ein Etikett angegeben werden. Es dient einerseits zur Anreicherung der Präsentation an der Benutzerschnittstelle, andererseits erlaubt es die Differenzierung von Parametern der gleichen Klasse (was für die Formulierung von Bedingungen wichtig sein kann). Außerdem ist für jeden Parameter anzugeben, ob er optional ist oder nicht. In einigen Programmiersprachen können für einen Dienst (oder genauer: für einen Dienst-Namen) alternative Parameter[1] verschiedener Klassen angegeben werden. In Abhängigkeit von der Klasse des übergebenen Parameters können dann unterschiedliche Methoden zur Ausführung

1. Booch spricht hier in Anlehnung an Lisp von "generic parameters". Vgl. dazu III.3.4.2.

gebracht werden. Ähnlich wie im Falle der (einfachen) Polymorphie erfolgt der dazu nötige Dispatch transparent. Ein solches "overloading" der Parameter ist nicht vorgesehen.

Output

Wenn ein Dienst einen Ergebnis zurückliefert, ist dies immer genau ein Objekt - ähnlich wie bei Funktionen in konventionellen Programmiersprachen. Damit ist nicht ausgeschlossen, daß es sich dabei um mehrere Werte handelt: Ein Objekt kann komplex sein - wie etwa eine Liste oder ein Vektor. Zur Charakterisierung dieses Objekts wird seine Klasse angegeben. Außerdem kann wiederum ein Etikett benannt werden.

Um die formale Voraussetzungen für die Korrektheit von Programmen zu spezifizieren, wird in der Theorie der Programmiersprachen unter anderem auf sogenannte Assertionen verwiesen. Assertionen sind Bedingungen, die bei der Ausführung eines Programmteils eingehalten werden müssen. Zur Kennzeichnung von Diensten lassen sie sich in Preconditions (Vorbedingungen) und Postconditions (Nachbedingungen) unterteilen.

Precondition

Vorbedingungen eines Dienstes beschreiben Voraussetzungen, unter denen der Dienst ausgeführt werden darf. Zu den impliziten Vorbedingungen gehört, daß die übergebenen Parameter von der jeweils erforderlichen Klasse sind. Explizite Preconditions drücken allgemein Bedingungen aus "under which a routine will function properly" (Meyer 1988, S. 114). Dabei ist vor allem an die Zustände der übergebenen Parameter zu denken. So mag es beispielsweise für einen bestimmten Dienst, der ein Objekt der Klasse "Person" erfordert, wichtig sein, daß dieses Objekt das Geschlecht "weiblich" aufweist.

Postcondition

Eine Postcondition beschreibt eine Bedingung, die nach der Ausführung eines Dienstes erfüllt sein muß. Hier sind viele Bedingungen, die den Zustand des gesamten Objekts oder auch assoziierter Objekte betreffen, denkbar. Besonders wichtig allerdings ist der Zustand des gegebenenfalls zurückgereichten Objekts. Diese Objekt muß - implizite Nachbedingung - von der geforderten Klasse sein. Daneben können in einer Postcondition Anforderungen an den Zustand des Objekts formuliert werden. Wenn ein Dienst beispielsweise ein Objekt der Klasse "Geschäftsbrief" zurückliefert, mag es wichtig sein sicherzustellen, daß der Brief ein Datum enthält.

Ausnahmen

Eine Ausnahme tritt immer dann ein, wenn ein Dienst die gewünschte Leistung nicht in der vorgesehenen Weise erbringen kann. Implizite Ausnahmen treten ein, wenn explizite Vor- oder Nachbedingungen nicht erfüllt sind. Explizite Ausnah-

men beschreiben andere mögliche Ursachen für das Scheitern eines Dienstes. Dazu gehört beispielsweise die Verfügbarkeit der zur Ausführung eines Dienstes benötigten Geräte. Grundsätzlich beschreiben Ausnahmen also die Verletzung solcher Voraussetzungen, deren Vorliegen sich erst zur Laufzeit erweisen kann.

Beim Auftreten einer Ausnahme sollte - wenn möglich - eine Ausnahmebehandlung veranlaßt werden. Sie kann in einer geeigneten Benachrichtigung des Benutzers am Bildschirm bestehen oder auch darin, einen Ersatz für ein nicht verfügbares Gerät zu finden. Im Unterschied zu anderen Eigenschaften einer Klasse oder eines Dienstes sollten Ausnahmen deshalb für das ganze Objektmodell einen eindeutigen Namen haben. Das schafft die Voraussetzung für eine systemweit einheitliche Ausnahmebehandlung.

Vor- und Nachbedingungen wie auch Ausnahmen dienen dazu, die Wahrscheinlichkeit für die korrekte Ausführung eines Dienstes zu verbessern. Für die an dieser Stelle nicht näher betrachtete Implementierungsebene ist deshalb zu fordern, daß äquivalente Konzepte zur Verfügung stehen - wie es beispielsweise in Eiffel (vgl. Meyer 1989 a) der Fall ist. Ferner ist im Hinblick auf Postconditions und Ausnahmen ein geeignetes Transaktionskonzept wünschenswert.

Grundsätzlich ist eine Vielzahl möglicher Ausnahmen, Vor- und Nachbedingungen denkbar. Für die konzeptuelle Modellierung ist zu überlegen, diese Vielzahl einzuschränken: Zum einen, um die mit der Vielfalt verbundene Komplexität zu reduzieren, zum anderen, um die Chancen für eine weitgehend automatische Implementierung zu verbessern. In V.2.1 werden entsprechende Vorschläge entwickelt.

Zugriffsrecht

Zur Beschreibung des Zugriffsrechts auf einen Dienst kann zwischen drei Autorisierungsgraden gewählt werden: public, protected und private - jeweils mit der oben erläuterten Bedeutung.

Algorithmus

Für den Fall, daß eine algorithmische Beschreibung der den Dienst bereitstellenden Methode vorliegt, kann sie als Ergänzung hinzugefügt werden. Im Hinblick auf die Implementierung lassen sich dabei zwei Formen unterscheiden: Solche, die vor allem der anschaulichen Darstellung des Algorithmus dienen (wie etwa Programmablaufpläne oder Struktogramme), und solche, die sämtliche Angaben, die für eine automatische Transformation in ausführbaren Code benötigt werden (wie Pseudo-Code oder Code einer geeigneten Programmiersprache). In jedem Fall sollten dabei die anderen Spezifikationen, die den Algorithmus betreffen, beachtet werden. Dazu gehören einerseits die oben erläuterten Kennzeichnungen des Dienstes wie die Festlegung der Klasse von eingehenden und zurückgelieferten Objekten, Pre-, Postconditions etc., andererseits sonstige Angaben zur

Klasse, die Auswirkungen auf die Ausführung des jeweiligen Dienstes haben. Dabei ist vor allem an die unten beschriebenen Trigger und Guards zu denken. In V.2.1 wird im Detail beschrieben, wie die Konsistenz dieser Zusammenhänge werkzeuggestützt gefördert werden kann.

2.1.1.1.3 Trigger und Guards

Im Lebenszyklus eines Objekts kann es Ereignisse geben, auf die in bestimmter Weise reagiert werden sollte. Daneben kann es - jenseits der für Attribute und Dienste beschriebenen Einschränkungen - Zustände des Objekts geben, die unbedingt vermieden werden sollten. Im ersten Fall sprechen wir in Übereinstimmung mit gängiger Terminologie von einem *Trigger*. Die Einschränkungen bzw. Constraints zulässiger Objekteigenschaften nennen wir im Kontrast dazu *Guard.*[1] Während es in traditionellen datenorientierten Systemen an den Anwendungen liegt, auf bestimmte Ereignisse in angemessener Weise zu reagieren oder bestimmte Operationen auf Daten zu verhindern, bietet ein objektorientierter Ansatz die Chance, diese Semantik unmittelbar an ein Objekt bzw. eine Klasse zu binden.

Ein Trigger kann allgemein beschrieben werden als ein Tupel aus einem Ereignis und einer Aktion. Darüber hinaus kann als weiteres Merkmal noch eine Bedingung genannt werden, unter der allein die Aktion ausgeführt werden darf. Ereignisse werden häufig als Zustandsänderungen, deren zeitliche Dauer vernachlässigt werden kann, definiert. Wir verwenden hier eine Begriffsfestlegung, die dieser weitgehend entspricht, sie allerdings ausweitet. Danach ist ein Ereignis, das für die Modellierung der Trigger einer Klasse relevant ist, definiert als das plötzliche Eintreten von Bedingungen, die von einem Objekt dieser Klasse überprüft werden können. Eine Aktion wird durch einen Dienst (evtl. mit der Vorgabe bestimmter Parameter) festgelegt. Da ein Ereignis als Bedingung beschrieben wird, ist es logisch nicht zwingend, davon noch eine weitere Bedingung zu unterscheiden: Beide Bedingungen werden durch eine Konjunktion zu der einen das Ereignis definierenden Bedingung. Im Hinblick auf die Implementierung ist es allerdings durchaus sinnvoll, die Bedingung, deren Eintreten das Ereignis bestimmt, von der Bedingung zu trennen, unter der das Ereignis relevant sein soll. Das folgende Beispiel wird diesen Umstand verdeutlichen. Zunächst können wir jedoch festhalten, wie ein Trigger allgemein konzeptualisiert werden kann:

1. Die Wahl zweier englischer Begriffe (von denen einer auf der Ebene der Fachterminologie bereits ins Deutsche überführt ist) neben den anderen deutschen Begriffen (wie Dienst, Attribut u.a.) ist nicht als Vorbild für die Schaffung einer einheitlichen Terminologie gedacht. Sie ist vielmehr ein Reflex auf die faktische Verwendung von (Fach-) Sprache. Mitunter wird zwischen Trigger und Guard nicht unterschieden. So spricht Graham (1991, S. 242) ohne weiter zu differenzieren von "business rules", die einem Objekt zugeordnet werden können.

Ereignis + Bedingung -> Aktion

Ereignisse können dabei in unterschiedlichen Grundformen und in beliebiger Komplexität auftreten. Behrends/Jasper (1993, S. 471) unterscheiden im wesentlichen vier Formen "atomarer" Ereignisse. "Ereignisse der Datenbank" werden ausgelöst durch Datenbankoperationen wie Einfügen, Löschen, Ändern und Lesen von Daten. "Zeitereignisse" werden über Bedingungen definiert, die unter Rückgriff auf Zeitpunkte oder Zeitintervalle spezifiziert werden. Daneben sind Ereignisse vorgesehen, die durch den menschlichen Benutzer über die Interaktionselemente der Benutzerschnittstelle ausgelöst werden. Als Besonderheit nennen Behrends und Jasper Ereignisse, die mit Hilfe statistischer Kennzahlen definiert werden. Solche Kennzahlen werden unter Rückgriff auf eine bestimmte Grundgesamtheit der Datenbank sowie statistische Verfahren spezifiziert. Ein Ereignis tritt dann ein, wenn eine Kennzahl im Zeitverlauf einen kritischen Wert erreicht hat.

Die hier vorgeschlagene Kategorisierung von Ereignissen unterscheidet sich von dem skizzierten Ansatz in einigen Punkten. So sind weder konventionelle Datenbankoperationen noch Eingriffe des Benutzers in einem aus Objekten bestehenden Informationssystem durch eine eigene Besonderheit gekennzeichnet. Letztlich kommt es in beiden Fällen zum Aufruf bestimmter Dienste in einzelnen Objekten. Um die grundsätzlich mögliche Vielfalt von Ereignissen, die der oben dargestellten Definition genügen, zu ordnen, betrachten wir im folgenden Grundformen von Ereignissen, die in betrieblichen Informationssystemen relevant sein können.

Im einfachsten Fall handelt es bei einem Ereignis um Änderungen von Attribut-Werten eines Objekts. Beispiel: Wenn der Bestellbestand eines Artikels erreicht bzw. unterschritten ist, soll ein Bestellvorgang ausgelöst werden. Das Beispiel macht deutlich, daß die explizite Einführung von Bedingungen logisch nicht notwendig ist: Wenn beispielsweise der beschriebene Trigger nur eingeschränkt für Artikel mit bestimmten Eigenschaften gelten soll, wird die Bedingung, die das auslösende Ereignis beschreibt, entsprechend erweitert. Gleichzeitig wird aber deutlich, daß die das Ereignis unmittelbar betreffende Bedingung (Bestand <= Bestellbestand) für die Implementierung in anderer Weise zu berücksichtigen ist als eine einschränkende Bedingung, die beispielsweise den Einkaufspreis eines Artikels berücksichtigt: Der Trigger kann *nur* dann feuern, wenn der Bestand verringert wird. Das Ereignis könnte also beispielsweise in Form einer Precondition in den Diensten, die eine Bestandsverringerung ermöglichen, überprüft werden.

Daneben gibt es Ereignisse, die durch Beziehungen des Objekts zu anderen Objekten beschrieben werden. Beispiel: Ein Kunde erhält einen neuen Haftpflichtvertrag. Auch hier ändert sich (in der Regel) der Zustand des Objekts, da

eine Referenz auf ein externes Objekt angelegt wird. In beiden Fällen treten die Ereignisse mit der Ausführung bestimmter Dienste des Objekts ein. Sie können also - denkt man an ihre Implementierung - durch geeignete Vor- oder Nachbedingungen in diesen Diensten beschrieben werden.

Eine andere Art von Ereignissen wird durch Bedingungen festgelegt, die auf die Frequenz, mit der einzelne Dienste ausgeführt werden, rekurrieren. Beispiel: Wenn der Bestand eines Artikels innerhalb eines Monats mehr als 1000 Mal verringert wurde, ist dem zuständigen Produktmanager eine Nachricht zu schicken. In der Regel tritt die Bedingung genau dann ein, wenn einer der betreffenden Zugriffsdienste ausgeführt wird. Durch die Implementierung eines entsprechenden Zählers kann das Objekt die Bedingung also leicht überprüfen. In einem Sonderfall, der hier nicht ausgeschlossen werden soll, gestaltet sich die Überprüfung schwieriger: Wenn die das Ereignis definierende Bedingung darin besteht, daß ein bestimmter Dienst innerhalb einer vorgegebenen Zeit überhaupt nicht ausgeführt wurde. Das Objekt kann die Bedingung nur evaluieren, wenn es permanent (in geeigneten Intervallen) die Zeit abfragt.

Ähnliche Schwierigkeiten ergeben sich, wenn man Attribute mit temporaler Semantik betrachtet: Deren Interpretation mag sich im Zeitverlauf ändern - ohne daß sich ihr Zustand ändert. Ein typisches Beispiel dafür ist das Ereignis "Geburtstag". Ein daran anknüpfender Trigger könnte lauten: "Wenn ein Mitarbeiter fünfzig Jahre alt wird, ist ihm eine einmalige Gratifikation auszuzahlen." Die Verwaltung solcher Trigger (die nicht durch den Aufruf bestimmter Dienste ausgelöst werden) ist innerhalb des Objekts unter heutigen Hardware-Restriktionen nicht immer möglich. Dazu müßten alle Instanzen einer Klasse (also beispielsweise alle Objekte, die Mitarbeiter repräsentieren) aktiv sein, also ständig die sich verändernden zeitlichen Randbedingungen überwachen.[1] Dennoch können sie auf der konzeptuellen Ebene beschrieben werden, da sie einerseits zur Semantik der Klasse gehören mögen und da andererseits von Implementierungsrestriktionen, die Änderungen unterworfen sind, abstrahiert werden sollte.[2]

Schließlich ist an solche Ereignisse zu denken, die insofern eine Sonderstellung einnehmen, als sie an den Grenzen oder gar außerhalb der Lebensdauer eines Objekts liegen. Mitunter ist es wichtig, zu erfahren, daß ein neues Objekt einer Klasse entstanden ist. Beispielsweise möchte man informiert werden, wenn in

1. Seit einiger Zeit werden Konzepte für sogenannte "aktive Datenbanken" diskutiert, die entweder in Erweiterung traditioneller Systeme oder als Bestandteil objektorientierter DBMS die Verwaltung "aktiver Objekte" erlauben. Vgl. dazu Behrends/Jasper (1993).

2. Eine andere Möglichkeit, die hier nicht ausgeschlossen werden soll, besteht darin, spezielle Trigger-Objekte zu entwerfen. Sie überprüfen eine Menge von Objekten auf bestimmte Bedingungen und sorgen gegebenenfalls für die Ausführung zugeordneter Aktionen..

einem bestimmten Nutzungskontext ein neues Dokument kreiert wurde. Auch der entgegengesetzte Fall, das Terminieren eines Objekts kann von Interesse sein. Wenn das Instanzieren oder Löschen nicht grundsätzlich für alle Objekte der Klasse von Bedeutung sein muß, ist die Menge der interessierenden Objekte einzuschränken. Dazu kann auf einen Dienst zurückgegriffen werden, der die relevanten Objekteigenschaften liefert. Die Semantik eines durch das Löschen ausgelösten Triggers kann ohne Probleme der jeweiligen Klasse zugeordnet werden, wenn grundsätzlich jedes Objekt für sein Ableben selbst verantwortlich ist. In diesem Fall gibt es einen entsprechenden Dienst wie etwa "delete", in den ein solcher Trigger als Postcondition eingefügt werden könnte.[1] Demgegenüber ist das Instanzieren eines Objekts grundsätzlich ein Ereignis, das von dem Objekt selbst nicht überwacht werden kann, da es ja vor seiner Lebenszeit liegt. Der Trigger müßte also auf Klassenebene - oder allgemeiner: dort, wo eine Instanz erzeugt wird - implementiert werden. Wenn auch der Zustand des erzeugten Objekts eine Rolle spielt, ist ein solcher Ansatz jedoch nicht angemessen. Eine mögliche Lösung ist darin zu sehen, daß beim Erzeugen neuer Instanzen grundsätzlich ein Initialisierungsdienst aufgerufen wird. Dort könnte dann - als Postcondition beschrieben - ein solcher Trigger eingebunden werden. Eine weitere Möglichkeit ist darin zu sehen, daß der Trigger, für den Modellierer transparent, auf Klassenebene - wenn die verwendete Implementierungssprache dies erlaubt - verankert wird.

Im Hinblick auf die Implementierung ist einem Umstand Rechnung zu tragen, der bei der Modellierung nicht explizit berücksichtigt werden muß: Nachdem ein Ereignis eine Aktion ausgelöst hat, kann die das Ereignis definierende Bedingung nach wie vor gültig bleiben. Es sind deshalb geeignete Verfahren vorzusehen, die verhindern, daß durch ein Ereignis eine endlose Folge von Aktionen ausgelöst wird.

Ein Guard bezeichnet solche Integritätsbedingungen, die über die Definition einzelner Attribute hinausgehen. Meyer (1988, S. 124) spricht in diesem Zusammenhang von einer "class invariant" als einer "list of assertions, expressing general consistency constraints that apply to every class instance as a whole". Die Bedingung, die eine solche Invariante beschreibt, kann sich auf die Verknüpfung von Eigenschaften des Objekts selbst beschränken. Beispiel: Der Verkaufspreis eines Artikels sollte niemals geringer sein als der Einkaufspreis. Mitunter ist die Instanzierung eines Attributs nur dann möglich, wenn ein anderes Attribut in bestimmter Weise instanziert wurde. Beispiel: Nur wenn für ein Objekt der

1. In Laufzeitsystemen, die einen "garbage collector" vorsehen, um den Programmierer also gerade von solchen Aspekten der Objektverwaltung zu entlasten, könnte das betreffende Objekt selbst den Trigger nicht verwalten - es wird ja nicht davon unterrichtet, wenn die auf es gerichteten Referenzen gelöscht werden.

Klasse "Drucker" das Attribut "grafikfähig" mit dem Wert "wahr" instanziert ist, soll das Attribut "Auflösung" instanziert werden dürfen. Solche Konstellationen sollte man zwar grundsätzlich durch geeignete Spezialisierungen zu vermeiden suchen. Falls dies aber im Einzelfall als nicht angemessen erscheint, könnte die skizzierte Integritätsbedingung mit Hilfe eines Guards artikuliert werden. Daneben können invariante Zustände eines Objekts der betrachteten Klasse in Abhängigkeit von Objekten anderer Klassen definiert werden. Beispiele: Das Gehalt eines Mitarbeiters sollte niemals höher sein als das Gehalt seines Vorgesetzten. Oder: Nur eine Person, die mindestens achtzehn Jahre alt ist, darf einen KFZ-Haftpflichtversicherungsvertrag halten. Eine Besonderheit stellen Container- oder Collection-Klassen wie Menge, Verzeichnis, Liste oder Vektor dar. Für sie kann ein Guard formuliert werden, der festlegt, von welcher Klasse die jeweils verwalteten Objekte sein müssen.[1]

Ein Guard kann allgemein deklarativ mit Hilfe einer logischen Proposition beschrieben werden, die immer erfüllt sein muß. Ähnlich wie bei Ereignissen sollen diese Propositionen auf solche Bedingungen eingeschränkt werden, die von den Objekten der modellierten Klasse evaluiert werden können. Neben Aussagen, die immer gelten müssen, können auch solche angeben werden, die unter bestimmten Bedingungen zu gelten haben. Dazu wäre dann eine logische Implikation zu verwenden. Beispiel: Wenn eine Person verheiratet ist, dann muß ihr Ehepartner ein anderes Geschlecht haben als sie selbst. Ähnlich wie bei Triggern stellt sich die Frage, in welcher Form die Spezifikation erfolgen sollte. Eine formalsprachliche Darstellung bietet große expressive Möglichkeiten. Sie ist allerdings mit Nachteilen verbunden: Einerseits konfrontiert sie den Betrachter des Modells mit einer erheblichen Komplexität, andererseits verringert sie die Chance einer weitgehend automatischen Implementierung. In V.2.2.1 wird eine Konzeptualisierung von Guards und Triggern entwickelt, die eine komfortable Spezifikation erlaubt und günstige Vorausetzungen für eine automatische Implementierung mit sich bringt, allerdings nur eine eingeschränkte Mächtigkeit aufweist.

Im Hinblick auf die Konsistenz eines Modells ist darauf zu achten, daß die Semantik eines Guards oder eines Triggers in den je betroffenen Diensten berücksichtigt wird. Beim Einsatz eines Entwicklungswerkzeugs könnte dies (zum Teil) erreicht werden, indem entsprechende Pre- oder Postconditions generiert werden. In diesem Zusammenhang stellt sich die Frage, warum die in einem Guard oder einem Trigger beschriebene Semantik nicht gleich in Pre- oder Postconditions spezifiziert werden sollte. Für die Definition von Triggern oder

1. Booch (1990, S. 116 f.) modelliert diese Art von Integritätsbedingung mit Hilfe einer sogenannten "instantiation relationship", durch die eine Container-Klasse mit der Klasse der Objekte, die ihre Instanzen aufnehmen dürfen, verbunden wird.

Guards spricht das höhere Abstraktionsniveau: Die dort beschriebene Semantik mag sich auf mehrere Dienste auswirken. Wenn wie in dem oben genannten Beispiel festgelegt wird, daß ein Verkaufspreis niemals kleiner als ein Einkaufspreis sein darf, ist dieser Umstand in allen Diensten zu berücksichtigen, die den Verkaufspreis verändern.

Wie bereits erwähnt können extern gesetzte Vergleichswerte (wie beispielsweise das Volljährigkeitsalter) durch Konstanten oder mit Hilfe von Variablen spezifiziert werden. Im zweiten Fall - der in aller Regel zu präferieren sein wird - kann die Trigger- bzw. Guard-Semantik während der Lebenszeit eines Objekts verändert werden. Es sind dann also entsprechende Dienste zur Konfiguration vorzusehen. Dabei kann zwischen einer klassenspezifischen und einer objektspezifischen Konfiguration unterschieden werden. Im ersten Fall werden die Variablen einer Bedingung oder einer Proposition auf Klassenebene instanziert. Eine so konfigurierte Aussage gilt dann in gleicher Weise für alle Objekte der Klasse. Im zweiten Fall kann eine Konfiguration für ein Objekt (also beispielsweise für einen bestimmten Mitarbeiter) vorgenommen werden.

2.1.1.1.4 Präsentation

In der konzeptuellen Modellierung wird die Benutzerschnittstelle traditionell nicht berücksichtigt. Zur Begründung wird häufig darauf verwiesen, daß die Ausgestaltung der Benutzerschnittstelle eine Eigenschaft des Systems darstellt (also nicht zu den zu modellierenden Eigenschaften der jeweiligen Domäne gehört), die zudem technologischem Wandel unterliegt. Es gibt dennoch gute Gründe dafür, bei der Modellierung von der Benutzerschnittstelle nicht zu abstrahieren. So kann die Präsentation eines Objekts durchaus als eine Eigenschaft des Objekts angesehen werden. Ein anschauliches Beispiel dafür ist das Photo einer Person: Seine Bedeutung ist - aus der Sicht der Anwender - gleichsam durch seine Darstellung konstituiert. Auch andere Objekte sind sehr eng mit einer bestimmten Darstellungsform assoziiert. Die konzeptuelle Modellierung sollte ein Bild entwerfen, das den Anwendern eine möglichst klare Vorstellung von dem zu implementierenden System vermittelt - nur so kann eine fruchtbare Kritik der Anwender erwartet werden. In diesem Zusammenhang sind drei Feststellungen unstrittig:

- Für die vom Anwender wahrgenommene Qualität eines Informationssystems ist dessen Benutzerschnittstelle von zentraler Bedeutung. Das empfiehlt die Beteiligung der Anwender bei der Gestaltung der Benutzerschnittstelle - auf einer anschaulichen Abstraktionsebene.
- Die Anschaulichkeit eines konzeptuellen Modells wird durch die Erstellung eines korrespondierenden Prototyps gefördert. Wenn ein konzeptuelles Modell Elemente der Benutzerschnittstelle bereits beinhaltet, ist es ceteris paribus mit

wesentlich weniger Aufwand verbunden, vom konzeptuellen Modell zu einem funktionsfähigen Prototyp zu gelangen.

- Der Entwurf und die Implementierung der Benutzerschnittstelle beansprucht bei vielen Anwendungssystemen den größten Teil der gesamten Entwicklungszeit. Es ist deshalb wünschenswert, daß die konzeptuelle Modellierung eine Beschreibung der Benutzerschnittstelle beinhaltet, die auf wiederverwendbare Konstrukte verweist.

Wenn vor dem Hintergrund dieser Argumente die Beschreibung von Aspekten der Benutzerschnittstelle mit in die konzeptuelle Modellierung übernommen werden soll, stellen sich vor allem zwei Fragen:

- Ist es möglich, Beschreibungen der Präsentation von Objekten zu erstellen, die einerseits von absehbarem technischen Wandel sowie den spezifischen Eigenarten existierender Formen der Bildschirmpräsentation abstrahieren und die andererseits dennoch mit vorhandenen und zukünftig entstehenden wiederverwendbaren Komponenten assoziierbar sind?
- Die bevorzugte Benutzer-Interaktion mit einem Objekt kann kontextabhängig variieren. Der Kontext ergibt sich durch den jeweiligen Verarbeitungszusammenhang und die Präferenzen des Betrachters. Im Objektmodell wird davon abstrahiert. Ist es dennoch in sinnvoller Weise möglich, die Präsentation eines Objekts im konzeptuellen Modell zu beschreiben?

Die erste Frage kann nur mit der Einschränkung beantwortet werden, daß die zukünftige Entwicklung der Präsentationstechnologie nicht mit letzter Bestimmtheit abzusehen ist. Gegenwärtig diskutierte Visionen wie die "Rooms"-Metapher oder noch ambitioniertere Formen der "virtual reality" liegen allerdings noch in weiter Ferne - und lassen darüber hinaus keinen revolutionären Wandel erkennen. Deshalb scheint es vertretbar, die Präsentation eines Objekts mit Hilfe eines Ansatzes zu modellieren, der sich schon heute in vielen Systemen findet und der darüber hinaus noch vielfältige Verfeinerungen zuläßt. Dazu werden auf der konzeptuellen Ebene Präsentations- oder besser: Interaktionsklassen eingeführt. Objekte dieser Klassen bieten bestimmte Darstellungs- und Interaktionsformen an. In der Terminologie heutiger Fenstersysteme werden solche Objekte "Widget" genannt. Gängige Bibliotheken zur Erstellung grafischer Benutzerschnittstellen enthalten eine Reihe verschiedener Widget-Klassen. In der Regel hat man Widgets für die Darstellung von Zeichenketten, von Listen von Zeichenketten, von Grafiken sowie diverser Druckknöpfe und Regler zur Verfügung. Häufig werden jeweils verschiedene Instanzierungsmöglichkeiten angeboten, so daß die Objekte einer Widget-Klasse sich in Aussehen und Verhalten unterscheiden können. So mag man ein Text-Widget in unterschiedlichen Größen und mit oder ohne Rollbalken erzeugen können. Die Verbindung eines Widgets mit einem Objekt erfolgt einerseits über Dienste des Widgets (wie etwa ein Dienst, der den

Inhalt des Widgets liefert - also beispielsweise eine Zeichenkette), andererseits über Dienste des Objekts, die Objektzustände - in geeigneter Form aufbereitet - an das Widget liefern. Die Kommunikation zwischen Objekt und Widget kann in unterschiedlicher Weise koordiniert sein. Fenstersysteme bieten in der Regel spezielle Objekte, die zwischen Objekt und Widget vermitteln.[1]

Die Beschreibung der Präsentation eines Objekts mit Hilfe von Widget-Klassen ist m.E. geeignet, in hinreichendem Maße von technischem Wandel zu abstrahieren. So bleiben auf der konzeptuellen Ebene Eigenschaften, die dem technischen Wandel in besonderem Maße unterliegen, weitgehend unberücksichtigt. Dabei ist an die grafische Gestaltung eines Widgets oder Einzelheiten seines Verhaltens zu denken. Die Modellierung ist zudem nicht an die Verwendung bereits existierender Widgets gebunden. Vielmehr können neue Widget-Klassen definiert werden (am komfortabelsten geschieht dies in der Regel durch Spezialisierung vorhandener Klassen).[2] Schon heute sind von einzelnen Anbietern spezielle Widget-Klassen für die Präsentation von Bildern und Videos am Markt verfügbar. Es ist deshalb davon auszugehen, daß das Widget-Konzept in Zukunft multimediale Benutzerschnittstellen zu gestalten erlauben wird.

Die Zuordnung einer Widget-Klasse zu einer Klasse ist in der Regel dann nicht hinreichend, wenn die Klasse Dienste hat, die unterschiedliche Objekte zurückliefern oder als Parameter benötigen. Dies ist unter anderem dann der Fall, wenn eine Klasse mehrere Attribute hat. Da aber in dem hier vorgestellten Metamodell die Klasse eines Attributs - oder allgemeiner: die Parameter oder das zurückgereichte Objekt eines Dienstes - selbst wieder mehrere Dienste aufweisen kann, kann auch auf dieser Ebene nicht unbedingt ein Widget zugeordnet werden. Um die skizzierte Problematik zu überwinden, wird ein Konstrukt eingeführt, das wir *Default View* nennen.[3] Um die Interaktion mit den Objekten einer Klasse zu modellieren, kann ihnen ein Default View zugeordnet werden. Für ein Default View werden folgende Annahmen gemacht:

1. Es gibt Klassen, deren Default View ein Widget ist. Hier ist an elementare Klassen wie solche zur Repräsentation von Zahlen und Zeichenketten zu denken.

---

1. Von den Details solcher Koordinationsmechanismen kann hier abstrahiert werden. In Goldberg/Robson (1989) wird mit dem Model/View/Controller-Konzept von Smalltalk ein herausragender Ansatz beschrieben.
2. Fortschrittliche Bibliotheken für grafische Benutzerschnittstellen bieten schon heute solche Adaptionsmöglichkeiten. Vgl. etwa Spenke et al. (1992).
3. Dadurch unterscheidet sich der hier entwickelte Ansatz von einigen Werkzeugen zur Gestaltung von Benutzerschnittstellen, die als Ergänzung zu objektorientierten Datenbankmanagement-Systemen angeboten werden. Sie erlauben lediglich die Zuordnung eines Widgets zur Klasse eines Attributs. Vgl. beispielhaft die Darstellungen in Butterworth/Otis/Stein (1991).

2. Klassen, deren Default View kein Widget ist, kann ein *Instance View* zugeordnet werden. Er dient der Darstellung einer gesamten Instanz. Beispielsweise mag der Default View einer Klasse "Datum" nicht ein Widget sein (sondern aus drei Widgets für "Tag", "Monat" und "Jahr" zusammengesetzt sein). Der Instance View wird definiert durch ein Widget und den zugehörigen Dienst der Klasse - also im Beispiel ein Dienst, der ein Datum als Zeichenkette liefert.
3. Der Default View anderer Klassen besteht aus einer Liste von *View-Beschreibungen* ihrer Dienste. Eine solche Beschreibung ist grundsätzlich optional (es gibt Dienste, deren Nutzung an der Benutzerschnittstelle nicht von Interesse ist). Sie ist nur möglich, wenn der Dienst nicht mit dem Zugriffsrecht "private" gekennzeichnet ist.
4. Die View-Beschreibung eines Dienstes rekurriert auf die Klassen seiner Parameter bzw. die Klasse des zurückgelieferten Objekts. Dabei können nur solche Klassen berücksichtigt werden, deren Default View ein Widget ist oder solche, denen ein Instance View zugeordnet ist.
5. Eine View-Beschreibung besteht aus folgenden optionalen Angaben: Für das gegebenenfalls zurückgelieferte Objekt ein Tupel mit dem Default View oder dem Instance View (also einem Widget) der Klasse des Objekts und einem Etikett. Falls der Dienst Parameter erfordert, kann für jeden Parameter ebenfalls ein Tupel mit dem Default View oder dem Instance View der Klasse des Parameters und einem Etikett angegeben werden.

Die in 4. gemachte Einschränkung hat folgenden Grund: Wenn ein Dienst ein Objekt zurückliefert, sind nur diejenigen Eigenschaften dieses Objekts für die Benutzerschnittstelle von Bedeutung, über die auf Dienste der zu beschreibenden Klasse zugegriffen werden kann.

Für Dienste, die Attribut-Werte setzen oder lesen[1], machen wir zwei vereinfachende Annahmen:

1. Es wird nur ein View für beide Zugriffs-Dienste verwendet, nämlich der Default View der Attribut-Klasse.
2. Es wird das Etikett des Attributs übernommen.

Daraus folgt für die Modellierung, daß die Möglichkeit geschaffen werden muß, die Identität zweier Widgets auszudrücken (beispielsweise durch die Vergabe von Namen, denen dann jeweils eine Widget-Klasse zugeordnet ist).

Das Beispiel in Abbildung 28 dient der Veranschaulichung der skizzierten Zusammenhänge. Es beschreibt die Zusammensetzung des Default Views einer Klasse "Mitarbeiter". Die Benennung der Dienste erfolgt in der in Smalltalk übli-

1. Andere Dienste, die auf ein Attribut zugreifen, beispielsweise solche, die einen Wert inkrementieren, sind davon ausgenommen.

chen Konvention: Der Dienst, der lesenden Zugriff auf ein Attribut erlaubt, heißt wie das Attribut selbst, wobei das erste Buchstabe klein zu schreiben ist. Die Bezeichnung des Dienstes, der den Wert eines Attributs setzt, wird zusätzlich mit einem Doppelpunkt abgeschlossen.

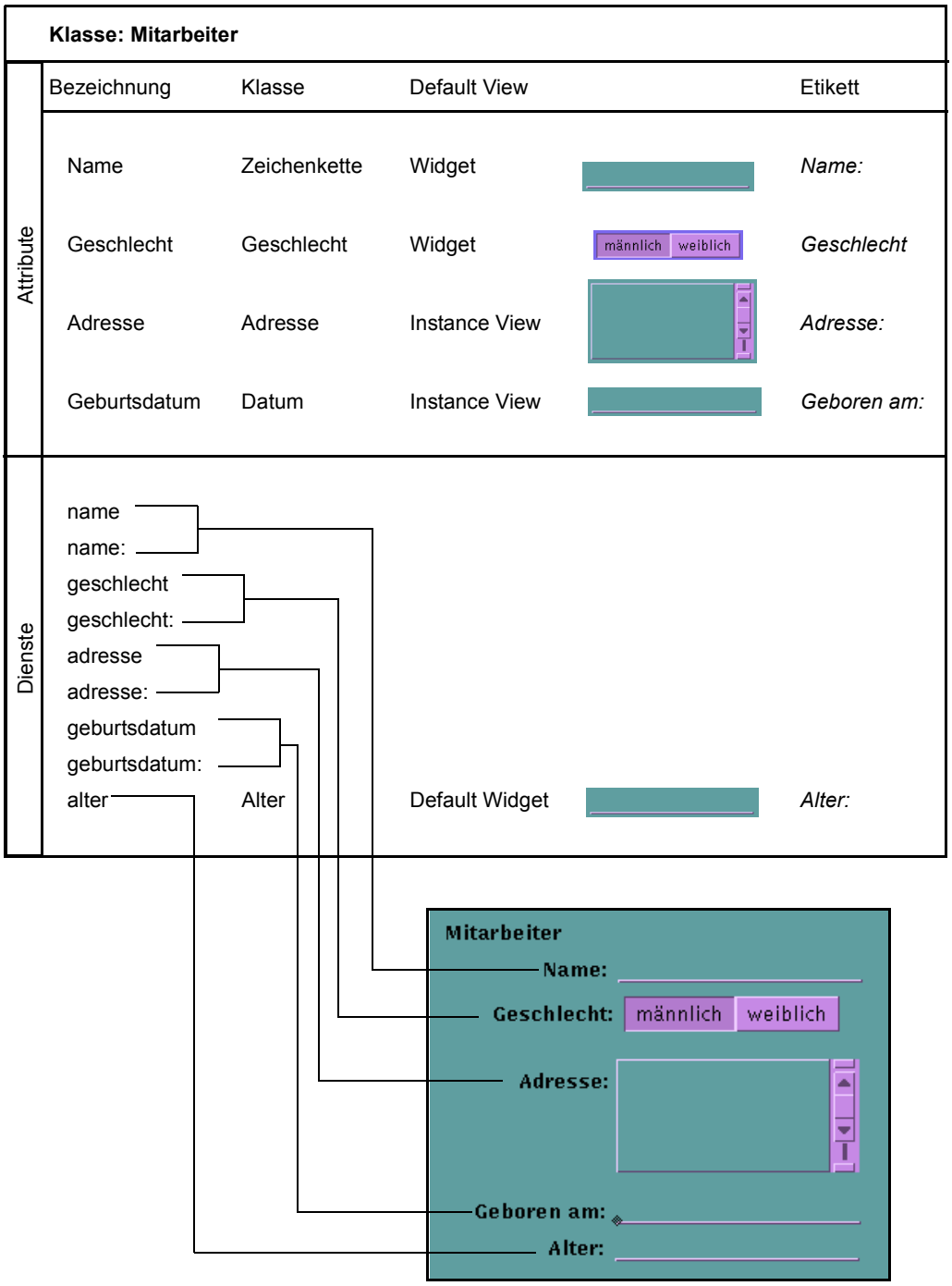


Abb. 28: Beispiel für die Zusammensetzung eines Default Views und die Verknüpfung der Widgets mit den Diensten einer Klasse.

Wenn die Bestandteile eines Attributs - wie etwa im Beispiel das Attributs "Adresse" - im Default View der Klasse getrennt dargestellt werden sollen, dann müssen in der Klasse entsprechende Zugriffsdienste (wie etwa "strasse") bereitgestellt werden. An diese Dienste werden dann die Default Views der jeweils behandelten Klassen (im Beispiel der der Klasse "Zeichenkette) gebunden.

Das dargestellte Beispiel ist nur begrenzt geeignet, den eingangs formulierten Zweifel daran, ob die Zuordnung von Präsentationsformen zu Objekten nicht in unzulässiger Weise gegen die Kontextabhängigkeit der Benutzerinteraktion verstößt, zu entkräften. Die in diesem Zweifel artikulierte Kritik kann allerdings dadurch entkräftet werden, daß von diversen Spezifika der Benutzerinteraktion abstrahiert wird. Es wird eben nur eine rudimentäre Beschreibung der Interaktionsmöglichkeiten an der Benutzerschnittstelle gegeben. So wird mit der Zuordnung einer Widget-Klasse weitgehend davon abstrahiert, wie die betroffenen Objekte die Darstellungsmöglichkeiten dieser Widget-Klasse nutzen. Das betrifft Marginalien wie die Wahl der Schriftart oder -größe, aber auch die Wahl zwischen unterschiedlichen grafischen Darstellungen eines Objekts (etwa einer Statistik). Vor allem aber wird von Details der Interaktion zwischen einem Objekt und seinem Default View abstrahiert: Wann soll der Inhalt eines Widgets aktualisiert werden, wie wird ein Objekt mit einem Widget assoziiert etc. Darüber hinaus wird davon abstrahiert, daß es Widgets gibt, die eine komplexere Funktionalität bieten. So kann eine Listbox beispielsweise auf einen Dienst zugreifen, der einen Vektor von Zeichenketten liefert - aber auch auf einen Dienst, der die Zeichenkette liefert, die gerade zu markieren ist.

Ein weiterer möglicher Einwand betrifft den Umstand, daß der Zugang eines Benutzers zu einem Informationssystem häufig nicht über genau ein Objekt erfolgt. Vielmehr gibt es Verarbeitungskontexte, in denen diverse Objekte eine Rolle spielen. Dabei muß die Interaktion eines Benutzers mit einem Objekt nicht immer all die Dienste betreffen, die im Default View berücksichtigt sind. In einem bestimmten Verarbeitungskontext mag eben neben dem Namen eines Mitarbeiters nur dessen Alter von Interesse sein. In V.3.2 wird ein Ansatz beschrieben, der es gestattet, solche kontextabhängigen Sichten auf Objekte zu modellieren.

2.1.1.2 Die Modellierung von Beziehungen

Die Objekte eines Informationssystems sind in vielfältiger Weise miteinander verknüpft: Sie bedienen sich gegenseitig ihrer Dienste, sie können aus anderen Objekten zusammengesetzt sein, ihre Existenz mag von der anderer Objekte abhängen und anderes mehr. Die Berücksichtigung solcher Beziehungen zwischen Objekten ist wesentlich dafür, die Integrität eines Informationssystem zu beschreiben. Sie werden deshalb gemeinhin als notwendiger Bestandteil von

Objektmodellen angesehen. Es gibt allerdings keinen allgemeinen Konsens darüber, wie Beziehungen konzeptualisiert werden sollten. Das betrifft vor allem die Frage, ob Beziehungen selbst als Objekte modelliert werden sollten.

Die Beziehungen, die in einem Objektmodell betrachtet werden, können solche zwischen Klassen sein und solche, die zwischen Instanzen bestehen. Die wichtigste Beziehung zwischen Klassen ist Generalisierung. Sie wird hier implizit durch die Angabe einer Oberklasse beschrieben. Daneben gibt es - in der Terminologie von Booch (1990) - Instanzierungsbeziehungen, deren Anwendung auf Container-Klassen beschränkt ist. Sie werden in unserem Modell ebenfalls nicht explizit, sondern implizit mit Hilfe von Guards beschrieben (vgl. IV.2.1.1.2). Aggregationsbeziehungen zwischen Klassen sind für das hier vorgeschlagene Metamodell von besonderer Bedeutung: Da Attribute grundsätzlich (von gewissen Ausnahmen abgesehen) durch beliebige Klassen gekennzeichnet werden können, geht eine Klasse eine Aggregationsbeziehung mit den Klassen ihrer Attribute ein. Die Beachtung solcher Beziehungen ist für die Wahrung der Konsistenz eines Modells von elementarer Wichtigkeit. Sie müssen allerdings nicht explizit beschrieben werden. Die implizite Festlegung von Beziehungen zwischen Klassen ist ein Reflex darauf, daß in dem hier vorgeschlagenen Metamodell keine unterschiedlichen Abstraktionen für Klassen und Objekte vorgesehen sind. Wir beschränken uns deshalb im folgenden auf Beziehungen zwischen Instanzen, über die mit Hilfe von Klassen generalisiert wird. Das heißt allerdings nicht, daß solche Beziehungen für abstrakte Klassen nicht spezifiziert werden dürfen: Eine abstrakte Klasse kann zwar selbst keine Instanzen haben, sie kann allerdings verwendet werden, um über alle Instanzen ihrer Unterklassen zu generalisieren.

2.1.1.2.1 Beziehungen als Objekte?

Es gibt eine Reihe von Eigenschaften, die gemeinhin verwendet werden, um die Beziehungen zwischen Objekten zu kennzeichnen. Dazu gehören Angaben über die zulässigen und tatsächlichen Kardinalitäten der assoziierten Objekte. Dieser Umstand allein scheint in einem konsequent objektorientierten Ansatz Anlaß genug, Beziehungen auch als Objekte zu modellieren: Ein Beziehungsobjekt kann dann beispielsweise Auskunft über die verknüpften Objekte und über zulässige Kardinalitäten geben, kann Objekte einfügen oder entfernen. Während Kardinalitäten und Referenzen auf andere Objekte grundsätzlich auch als Eigenschaften eines Objekts im weiteren Sinn betrachtet werden können, gibt es auch solche Merkmale, deren Bedeutung die Zuordnung zu einem der assozierten Objekte nicht sinnvoll erscheinen läßt. Wenn beispielsweise zwei Objekte der Klasse "Person" durch eine Beziehung "verheiratet mit" verbunden werden, liegt es nahe, Angaben wie Hochzeitsdatum und Hochzeitsort (wenn sie denn relevant sind) der Beziehung zuzuordnen. Die Modellierung einer Beziehung als Objekt

bietet darüber hinaus die Chance, die Vorteile der Vererbung zu nutzen und Beziehungen mit anwendungsspezifischer Semantik zu definieren. Es gibt denn auch in der Szene des objektorientierten eine Reihe von Proponenten der Modellierung von Beziehungen als Objekte.[1]

Auch wenn die dargestellten Argumente schwer wiegen, wird hier dennoch ein Ansatz favorisiert, in dem Beziehungen nicht als Objekte betrachtet werden. Er ist motiviert durch Erfahrungen mit der Modellierung von Beziehungen als Objekte.[2] Es hat sich gezeigt, daß jenseits möglicher softwaretechnischer Vorteile, die Aufhebung einer strikten Trennung zwischen Objekten und Beziehungen zu einer mitunter verwirrenden konzeptuellen Unschärfe führt. So ist eine Stärke der objektorientierten Modellierung gerade darin zu sehen, eine deutliche Korrespondenz zwischen Gegenständen der Realsphäre und Objekten zu ermöglichen. Anschließend sind die Beziehungen zwischen diesen Objekten zu betrachten. Wenn aber der Grundsatz gilt "alles ist ein Objekt", wird diese Differenzierung aufgehoben. Es können dann Beziehungen zwischen Beziehungen zwischen Beziehungen und so fort etabliert werden. Auch wenn dies zu mächtigen formalen Beschreibungen führen kann, ist damit die Erfüllung einer wesentlichen Anforderung an die konzeptuelle Modellierung gefährdet: Anschaulichkeit.

Die Betonung der Unterscheidung von Beziehungen und Objekten bewirkt in Einzelfällen eine entsprechend veränderte Modellierung. Immer dann, wenn einer möglichen Beziehung Attribute zugeordnet werden könnten, ist es angeraten, ein geeignetes Objekt dafür einzuführen. So wäre in dem oben beschriebenen Vermählungsbeispiel etwa eine Klasse "Hochzeit" vorzusehen, deren Instanzen dann über die genannten Sachverhalte Auskunft geben könnten.

2.1.1.2.2 Arten von Beziehungen

Auch wenn wir hier keine Beziehungsklassen bzw. -objekte vorsehen, sollte es doch möglich sein, verschiedene Formen oder Arten von Beziehungen zu unterscheiden: Aggregation hat eben eine andere Semantik als eine Beziehung, die lediglich ausdrückt, daß ein Objekt ein anderes benutzt. Damit stellt sich die Frage, welche Beziehungsarten zu berücksichtigen sind. Die Beantwortung der Frage hängt wesentlich von dem gewählten Abstraktionsniveau ab. Wenn man einen implementierungsnahen Standpunkt einnimmt, fällt es leicht, Booch (1990, S. 88) zuzustimmen: "We have found that two kinds of object hierarchies are of particular interest in object-oriented design, namely: using relationships, contai-

1. Vgl. dazu die Argumentation in Rumbaugh et al. (1991, S. 33 f.).
2. In Frank (1992 a) oder Frank/Klein (1992 b) sind Beziehungen noch als Objekte modelliert.

ning relationships."[1]

Eine Using-Beziehung besagt allgemein, daß die so verbundenen Objekte Nachrichten austauschen dürfen. Auch Ferstl/Sinz (1990) beschränken sich auf "interacts with" und "is part of"-Beziehungen. Aggregation und Interaktion sind sicherlich hinreichend, alle Beziehungen zwischen Objekten zu klassifizieren. Auch im Hinblick auf die Implementierung ist es sinnvoll, sich auf einige wohl verstandene Beziehungsarten zu beschränken. Deshalb beschränken wir uns hier ebenfalls auf diese beiden grundlegenden Arten von Beziehungen. Für sie können jeweils zwei Richtungen (Aggregation: "is part of", "contains"; Interaktion: "uses", "used by")[2] unterschieden werden. Aber: Auch wenn die Differenzierung in Aggregation und Interaktion softwaretechnisch hinreicht, ist sie für die Modellierung selbst wenig befriedigend. Die Modellierung soll schließlich in einer Diktion erfolgen, die an der Sprache in der jeweiligen Domäne und nicht an Implementierungsaspekten orientiert ist. Deshalb ist in dem hier vorgeschlagenen Ansatz vorgesehen, eine Beziehung mit Hilfe einer domänenorientierten Bezeichnung zu versehen. Da Beziehungen gerichtet sein können, kann zur Veranschaulichung zu jeder Bezeichnung eine inverse Bezeichnung angegeben werden.

Die Auszeichnung von Beziehungen mit Hilfe invertierbarer Bezeichnungen hat nicht nur den Vorteil, anschaulichere Modelle zu ermöglichen. Darüber hinaus liefert sie die Grundlage dafür, komplexe Objektmodelle durch die Definition geeigneter Filter übersichtlicher zu machen. Wenn man beispielsweise an einem Organisationsschema interessiert ist, könnte man aus dem Objektmodell alle Objekte ausblenden, die nicht in Beziehungen der Form “ist Vorgesetzter von”, “ist verantwortlich für” und dergleichen eingebunden sind.

Wir beschränken uns auf binäre Beziehungen. Das heißt, es können nur jeweils die Instanzen zweier Klassen[3] miteinander assoziiert werden. Diese Einschränkung scheint deshalb sinnvoll, weil binäre Beziehungen einfacher zu handhaben sind, und in den weitaus meisten Fällen hinreichen, um eine anschauliche Darstellung zu erhalten.

Die Kardinalität von Beziehungen wird in min, max-Notation ausgedrückt: Für jede der beiden beteiligten Klassen ist anzugeben, wie viele Instanzen für die Beziehung mindestens erforderlich sind und wie viele maximal zulässig sind. Die

1. Wobei Booch diese beiden Arten unter anderem durch die Unterscheidung von Beziehungen zwischen Klassen und solchen zwischen Objekten weiter differenziert.
2. Während in einer Aggregation immer beide Richtungsarten vertreten sein müssen, kann eine Interaktion zwei Objekte verbinden, die sich gegenseitig benutzen (und damit auch gegenseitig benutzt werden).
3. Darin ist der Spezialfall enthalten, daß Beziehungen zwischen Instanzen einer Klasse betrachtet werden.

Auszeichnung mit einer Rolle (also beispielsweise "Ehemann", "Ehefrau" in der Beziehung "ist verheiratet mit") ist nicht vorgesehen. Rollen dienen vor allem dazu, Integritätsbedingungen für Beziehungen zu formulieren. Eine solche an die Eigenschaften der assoziierten Objekte geknüpfte Bedingung kann in dem hier vorgestellten Metamodell mit Hilfe eines Guards beschrieben werden (vgl. IV.2.1.1.2). Ähnlich wie die Attribute eines Objekts können auch die zulässigen Beziehungen vom jeweiligen Zustand abhängen. Auch hier können Guards formuliert werden, um solche Integritätsbedingungen zu artikulieren. Beispiele: Ein Objekt der Klasse "Kunde" kann nur dann die Benutzt-Beziehung "besitzt" mit einem Objekt der Klasse "Kasko-Vertrag" eingehen, wenn das Alter des Kunden mindestens 18 Jahre beträgt. Oder: Einem Objekt der Klasse "Mitarbeiter" können nicht gleichzeitig die Rollen "Vorstandsmitglied" und "Systembetreuer" zugeordnet werden.

Die an realweltlicher Terminologie orientierte Bezeichnung von Beziehungen hat lediglich die Funktion, die Interpretation des Betrachters zu leiten und darauf basierende Auswertungen zu ermöglichen. Die für die Implementierung zu beachtende Semantik einer Beziehung ergibt sich allein aus der ihr zugeordneten Beziehungsart. Wir benötigen an dieser Stelle keine exakte Formalisierung der vorgeschlagenen Beziehungsarten. Vielmehr genügt eine Charakterisierung der Semantik, die sich im Überblick wie folgt darstellt:

- *uses*: Wenn ein Objekt eine uses-Beziehung zu einem anderen Objekt unterhält, bedeutet dies, daß es über eine Referenz auf dieses Objekt verfügen muß. In der Regel wird man damit die Integritätsbedingung verknüpfen, daß das benutzte Objekt während der Lebenszeit des nutzenden Objekts nicht gelöscht werden darf.
- *used by*: Ein benutztes Objekt muß das nutzende Objekt nicht kennen. Es hat allerdings zu beachten, daß es nicht terminieren darf solange die Beziehung besteht.
- *contains*: Hier handelt es sich um einen Spezialfall der uses-Beziehung. Wenn ein Objekt ein anderes beinhaltet, muß es eine Referenz auf dieses Objekt haben. Darüber hinaus sind Aggregationsbeziehungen durch Transitivität gekennzeichnet: Ein Objekt, das ein anderes Objekt beinhaltet, beinhaltet auch dessen Teile.
- *is part of*: Ein Spezialfall von used by, der ebenfalls durch Transitivität gekennzeicnet ist.

Eine Beziehung kann viele Paare von Klassen in unterschiedlicher Weise miteinander verknüpfen. Sie wird für je zwei Klassen konkretisiert, indem den Klassen Kardinalitäten zugeordnet werden. Außerdem ist im Fall der Interaktion zu spezifizieren, ob in beide Richtungen eine "uses"-Beziehung vorliegt oder nur in eine. Abbildung 29 zeigt ein Beispiel für die Beschreibung einer Beziehung zwischen

den Objekten zweier Klassen (die nicht unbedingt unterschiedlich ein müssen).

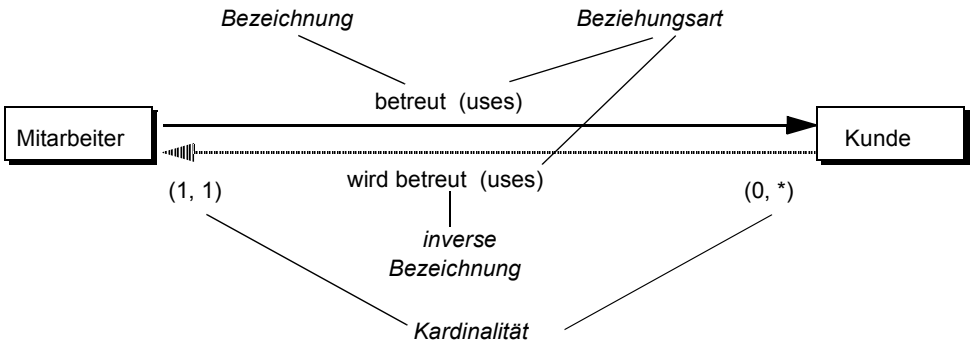


Abb. 29: Beispiel für die Darstellung einer Beziehung

2.1.2 Vorgangsmodelle

Die Beschreibung von Klassen in einem Objektmodell abstrahiert davon, unter welchen Bedingungen Objekte dieser Klassen in einzelnen Verarbeitungskontexten genutzt werden. Dabei ist vor allem an den Kontrollfluß zu denken, der logische und zeitliche Bedingungen dafür festlegt, in welcher Reihenfolge Dienste auszuführen sind. In den oben dargestellten Entwurfsmethoden von Booch und Rumbaugh et al. werden für die Beschreibung solcher dynamischer Sachverhalte State/Transition-Diagramme vorgeschlagen. Diese beschränken sich allerdings in beiden Methoden auf die abstrakte Darstellung des Lebenszyklus der Objekte *einer* Klasse. Für die Unternehmensmodellierung auf einem anwendungsnahen Niveau sind solche dynamischen Verarbeitungskontexte von besonderer Bedeutung, die gemeinhin als Vorgänge oder Bürovorgänge bezeichnet werden: Sie sind nicht nur für das Verständnis des Unternehmensgeschehens wichtig, sondern beschreiben Bedingungen für zulässige Zustandsänderungen einer Menge von Objekten, die für die Integrität eines Informationssystems von erheblicher Bedeutung sind. Da in solchen Vorgängen in aller Regel Objekte verschiedener Klassen verwendet werden, sind die oben genannten Methoden nicht hinreichend.

Dedizierte Ansätze zur Modellierung von Vorgängen[1] sind für eine anwendungsnahe, anschauliche Konzeptualisierung besser geeignet. Für unsere Zwecke sind sie allerdings mit dem Nachteil verbunden, daß sie vor allem auf eine (prototypi-

1. Wie etwa Ellis/Bernal (1982), Tsichritzis (1982), Croft (1987), Hasenkamp (1987), Kreifelts/Seuffert/Woetzel (1987). Dabei handelt es sich implizit um Bürovorgänge, eine Einschränkung, die auch hier übernommen wird.

schen) Implementierung zielen und entweder gar nicht objektorientiert[1] sind oder aber mit dem hier vorgestellten Objektmodell nicht kompatibel sind. Im folgenden wird deshalb ein Ansatz zur Modellierung von Vorgängen[2] entwickelt, der einerseits auf eine Integration mit dem Objektmodell, andererseits auf die Unterstützung eines schnellen Prototyping ausgerichtet ist.

2.1.2.1 Anforderungen

Nach gängiger Begriffsauffassung ist ein Vorgang ein wohlstrukturierter arbeitsteiliger Ablauf in Organisationen. Für den an dieser Stelle gewählten Betrachtungswinkel ist der Begriff auf solche Prozesse einzuschränken, die rechnergestützt ablaufen - womit eine Benutzerinteraktion allerdings nicht ausgeschlossen ist. Wohlstrukturiert heißt, daß es Vorgaben darüber gibt, wodurch ein Vorgang gestartet wird, und unter welchen Bedingungen welche möglichen Resultate erreicht werden. Diese Vorgaben müssen allerdings nicht vollständig in dem Sinne sein, daß ein deterministischer Ablauf garantiert ist. Die Modellierung eines Vorgangs einer bestimmten Art abstrahiert von konkreten Ausprägungen, beschreibt vielmehr alle zulässigen Abläufe.

Die Modellierung eines Vorgangs sollte die Einhaltung von Integritätsbedingungen unterstützen. Dazu gehören einerseits implizite Bedingungen, die die generelle Semantik eines Vorgangs kennzeichnen und schon beim Entwurf überprüft werden können:[3]

- Ein Vorgang darf nicht so entworfen sein, daß er niemals enden kann. Es dürfen also keine Zyklen eingebaut sein.
- Es darf in einem Vorgangsmodell keine Teile geben, die unter keinen Umständen ausgeführt werden können. Solche Teile würden zwar nicht notwendig die Konsistenz gefährden, wohl aber unnötigerweise die Übersichtlichkeit.
- Entwurfsbedingte Verklemmungen sind zu vermeiden.
- Wenn zwei Teilvorgänge als nebenläufig gekennzeichnet sind, müssen sie unabhängig voneinander ablaufen können. Es darf also beispielsweise nicht passieren, daß ein Teilvorgang ein Resultat produziert, das den anderen obsolet werden läßt.

Darüber hinaus gibt es eine Reihe von Anforderungen, die die Laufzeit eines Vorgangs betreffen. Dazu gehören:

---

1. Ein bekannter explizit objektorientierter Ansatz ist in Hogg (1987) dargestellt.
2. Vorgänge in der hier verwendeten Bedeutung schließen Transaktionen mit ein. Tendenziell können sie durch die Dauer unterschieden werden, was aber am Konzept nichts ändern muß.
3. Vgl. dazu Kreifelts/Seuffert/Woetzel (1987, S. 685).

- *Dispatching*: Bei der Ausführung des Vorgangs sollte nicht nur sichergestellt werden, daß die im zugehörigen Modell spezifizierten Bedingungen über die zeitliche Folge der Bearbeitungsschritte eingehalten werden. Darüber hinaus kann es wünschenswert sein, den Durchsatz zu fördern - was beispielsweise die Berücksichtigung der Kapazität von Sachbearbeitern nötig machen kann.
- *Ausnahmebehandlung*: Wenn in einem Vorgang eine Ausnahme[1] eintritt, sollte dies nicht zum Abbruch führen. Vielmehr sollte eine Ausnahmebehandlung dafür sorgen, den Vorgang durch geeignete Maßnahmen in konsistenter Weise weiterzuführen.
- *Transaktionskonzept*: Die im Rahmen eines Vorgangs vorgenommen Änderungen an den jeweils bearbeiteten Informationsbeständen sind in der Regel im Zusammenhang zu sehen. Wenn ein Vorgang oder Teile eines Vorgangs ex post ungültig werden, sollte dafür gesorgt werden, daß davon betroffene Änderungen wieder zurückgesetzt werden können.
- *Typkonsistenz*: Bei der Ausführung eines Vorgangs sollte sichergestellt werden, daß die eingegebenen und produzierten Informationen den Typen bzw. Klassen entsprechen, die ihnen bei der Modellierung zugewiesen wurden.

Neben diesen generellen Forderungen, die in unterschiedlicher Differenzierung in diversen Vorgangssystemen Berücksichtigung finden, sind für unsere Betrachtung zwei weitere Kriterien von besonderer Bedeutung:

- *Integration mit dem Objektmodell*: Hier ist einerseits an die Klassen des Objektmodells zu denken, die im Rahmen eines Vorgangs genutzt werden. Darüber hinaus kann ein Vorgangssystem selbst objektorientiert konzipiert und damit Bestandteil des Objektmodells werden.
- *Unterstützung des Prototyping*: Ein Vorgangsmodell sollte ein Objektmodell so ergänzen, daß eine komfortable Erstellung von Prototypen möglich wird.

2.1.2.2 Die Konzeptualisierung von Vorgängen

Die einschlägigen Ansätze zur Modellierung von Vorgängen sehen in der Regel die Aufspaltung eines Vorgangs in einzelne Teilvorgänge vor. Diese Unterteilung dient einerseits der Reduktion von Komplexität, andererseits spiegelt sie häufig die räumliche Verteilung von Teilvorgängen wieder. Ein Teilvorgang, mitunter auch Aktivität oder Aufgabe genannt, wird dabei zumeist durch ein bestimmtes Ereignis ausgelöst und produziert ein oder mehrere neue Ereignisse, die wiederum - wenn sie nicht den Vorgang terminieren - jeweils einen weiteren Teilvorgang auslösen. Auf diese Weise ergibt sich die Darstellung eines Vorgangs als

1. Um einige Beispiele für Ausnahmen zu nennen: Ein Sachbearbeiter erkrankt. Ein Benutzer macht eine falsche Eingabe, die erst später entdeckt wird. Ein Teilvorgang wird vom zuständigen Bearbeiter vergessen.

Netz. In Vorgängen werden zielgerichtet Informationen verarbeitet. Ein Vorgangssystem sollte dazu sicherstellen, daß die einzelnen Aktivitäten mit den benötigten Informationen versorgt werden und die jeweils zuständigen Sachbearbeiter informiert werden.

In dedizierten Ansätzen werden die Aktivitäten denn auch meistens durch die benötigte und die produzierte Information beschrieben. Die Konzeptualisierung dieser Informationen ist allerdings unterschiedlich. Mitunter wird zu deren Darstellung auf Datenmodelle von Datenbankmanagementsystemen zurückgegriffen oder es werden spezielle Strukturen eingeführt. In DOMINO (Woetzel/Kreifelts 1985) wird dazu in einer an PROLOG orientierten Sprache die Struktur eines Formulars festgelegt. Ein instanziertes Formular wird dann als Datei abgelegt und mittels elektronischer Post versendet. Die in DOMINO beschriebenen Daten sind nicht typisiert. Neben der Struktur eines Formulars können den einzelnen Aktivitäten Rollen der jeweils zuständigen Bearbeiter zugeordnet werden.

Während in DOMINO das Schwergewicht auf der automatischen Vorgangssteuerung lag, wurde in TLA[1] (Tsichritzis 1982) der Verarbeitung der Informationen in den einzelnen Teilvorgängen besondere Beachtung gewidmet. Dazu wurde ein rudimentär objektorientierter Ansatz gewählt: Ein Formular wird danach als ein abstrakter Datentyp definiert, dem Operationen und spezifische Integritätsbedingungen zugeordnet werden können. Ähnlich wie in DOMINO wird elektronische Post verwendet, um die Instanzen eines Formulartyps (die dazu auf ein sogenanntes "form template" abgebildet werden) zwischen den physisch veteilten Verarbeitungsorten zu übermitteln.

Die hier gewählte Konzeptualisierung von Vorgängen knüpft an die genannten Vorläufer an. Im Unterschied zu diesen Systemen ist unser Fokus allerdings auf die Modellierung, nicht auf die Implementierung gerichtet. Daraus folgt, daß davon abstrahiert werden kann, wie Nachrichten zwischen physisch verteilten Einheiten ausgetauscht werden. Außerdem ist der Ansatz darauf gerichtet, die mit Vorgängen verbundene Transaktionssemantik zu beschreiben, nicht darauf, ein konkretes Transaktionskonzept vorzuschlagen. In diesem Sinne unterscheidet er sich auch von aktuellen Weiterentwicklungen der erwähnten Vorgangssysteme, wie sie seit einiger Zeit unter dem Etikett "work flow management" diskutiert werden. Solche Systeme dienen der Steuerung oder Untersützung arbeitsteiliger Prozesse, die auch schwach strukturiert sein können und deren Integritätsbedingungen von den Benutzern in einem komfortablen Dialog in gewissem Umfang individuell spezifiziert werden können.[2]

1. TLA steht für "Toronto's Latest Acronym".

2. Ein Überblick über solche Systeme findet sich in Hasenkamp/Syring (1993). Kreifelts (1991) macht die wesentlichen Unterschiede zu traditionellen Vorgangssystemen deutlich.

Ein Vorgang wird als semantisch angereichertes Petri-Netz beschrieben. Er kann in Teilvorgänge unterteilt werden, die selbst wieder als Vorgänge beschrieben werden können. Einen Teilvorgang, der nicht weiter unterteilt werden soll, nennen wir *Aktivität*. Zwischen den Teilvorgängen erfolgt der Fluß der zu bearbeitenden Informationen. Die Informationen sind in unserem Fall Objekte, die für den Vorgang benötigt werden. Da sie in einem logischen Zusammenhang stehen (wenn beispielsweise ein Vorgang nachträglich gelöscht wird, sind alle benutzten Objekte davon betroffen), ist es angeraten, sie zusammenzufassen. Dies entspricht auch der traditionellen Vorgehensweise, bei der Vorgangsmappen - mitunter auch einfach "Vorgang" genannt - angelegt werden. In dem hier vorgeschlagenen Ansatz wird dazu ein Objekt eingeführt, das wir "Vorgangsdokument" nennen. Ähnlich wie eine Vorgangsmappe beinhaltet es die in einem Vorgang benutzten und produzierten Informationen. Anders formuliert: Ein Teilvorgang benötigt einen bestimmten Zustand des Vorgangsdokuments (das Eintreten dieses Zustands ist gleichzeitig das auslösende Ereignis) und er produziert einen oder mehrere Zustände des Vorgangsdokuments.

Ein Vorgangsdokument geht wesentlich über die Funktionalität einer Vorgangsmappe hinaus. So beinhaltet es nicht einfach nur bestimmte Präsentationen von Informationen, sondern Objekte, die bestimmte Dienste ausführen können. Wenn beispielsweise einem Vorgangsdokument ein Objekt der Klasse "Kunde" zugeordnet ist, dann sind damit sämtliche Dienste dieses Objekts verfügbar.

Darüber hinaus ist ein Vorgangsdokument nicht an die physischen Restriktionen einer Mappe gebunden: Es kann zur gleichen Zeit an unterschiedlichen Orten bearbeitet werden. Auch wenn hier von der technischen Realisation eines solchen Konzepts abstrahiert werden kann, läßt sich ein solches Objekt am einfachsten denken als eine Kollektion von Referenzen auf jeweils einmal vorhandene Objekte, auf die von allen Verarbeitungsorten zugegriffen werden kann.

Abbildung 30 zeigt einen Ausschnitt aus dem Modell eines Vorgangs. Auch wenn ein solches Netz beschrifteter Teilvorgänge und Vorgangsdokumentzustände durchaus eine anschauliche Darstellung eines Vorgangs liefern mag, so sind die damit verbundenen Angaben natürlich nicht hinreichend. Dazu sind Teilvorgänge sowie die Zustände des Vorgangsdokuments näher zu beschreiben. Im folgenden beschränken wir uns auf Aktivitäten, da andere Teilvorgänge durch Verweis auf sie beschrieben werden können. Ein spezielles Vorgangsdokument (etwa für die Bearbeitung eines Versicherungsschadens) ist ein Objekt einer Klasse, die von einer generellen Klasse "Vorgangsdokument" erbt. In der Definition dieser Klasse wird festgelegt, Objekte welcher Klassen in welcher Zahl in die Kollektion aufgenommen werden dürfen.

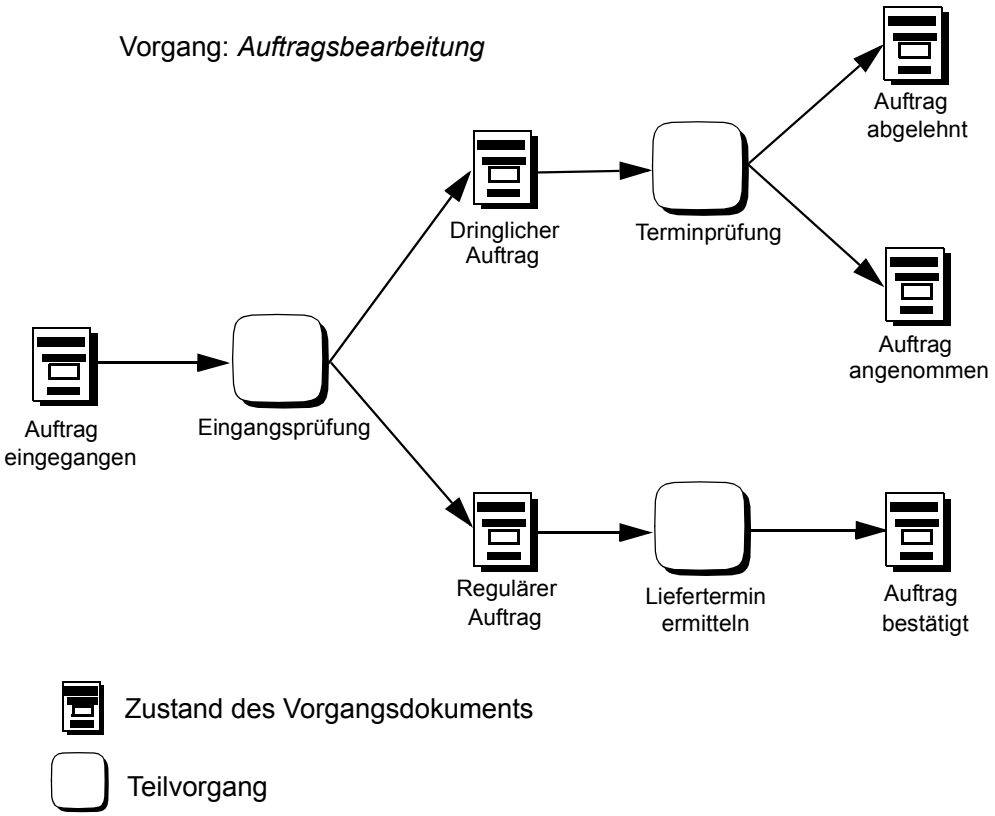


Abb. 30: Ausschnitt aus der Netzdarstellung eines Vorgangs mit Hilfe von Teilvorgängen und Zuständen eines Vorgangsdokuments

Darüber hinaus sind Dienste festgelegt, die über den Zustand des Dokuments Auskunft geben. Er ergibt sich aus den aufgenommenen Objekten, deren Zuständen sowie bestimmten Auszeichnungen des Dokuments selbst. Wenn beispielsweise ein Dokumentzustand von der Beurteilung eines Sachbearbeiters abhängt (wie etwa "genehmigt"), kann diese Statusinformation in einem entsprechenden Attribut des Vorgangsdokuments aufgenommen werden. In der Spezifikation einer Vorgangsdokument-Klasse können auch Trigger und Guards definiert werden. Auf diese Weise können kritische Zustände ausgezeichnet werden, die bestimmte Aktionen auslösen sollen. Dabei ist vor allem an die Zustände eines Vorgangsdokuments zu denken, die das Ende der Bearbeitung durch eine Aktivität anzeigen und damit (wenn es sich nicht um einen terminalen Zustand handelt) das Instanzieren der nachfolgenden Aktivität veranlassen.

Eine spezielle Aktivität sei ein Objekt einer Klasse, die Unterklasse einer generellen Klasse "Aktivität" ist.[1] Eine Aktivität wird nach dem Instanzieren mit dem Vorgangsdokument initialisiert (das heißt, sie erhält eine Referenz auf das Doku-

ment). Eine Aktivität kann neue Objekte in das Vorgangsdokument fügen - sofern sie gegen die dort festgelegten Anforderungen an Klasse und Kardinalität nicht verstoßen - oder auch Objekte entfernen. Von der generellen Klasse "Aktivität" werden diverse Dienste (und die zu deren Realisation nötigen Attribute) geerbt, wie etwa solche, die die geschätzte Bearbeitungszeit, die bisherige Bearbeitungszeit oder das bearbeitete Vorgangsdokument zurückliefern. Daneben ist eine spezielle Aktivität durch Dienste gekennzeichnet, die die für sie typischen Bearbeitungsschritte beschreiben - also beispielsweise "ermittle Liefertermin". Im Rahmen dieser Dienste werden wiederum bestimmte Dienste der Objekte des Vorgangsdokuments benutzt (wie etwa ein Dienst "ermittle Lagerbestand" eines Objekts der Klasse "Artikel"). In dem Dienst, der der Initialisierung einer Aktivität durch ein Vorgangsdokument dient, wird in einer Precondition geprüft, ob das Dokument in dem benötigten Zustand ist. Der Dienst oder die Dienste, die die Aktivität beenden, enthalten entsprechende Postconditions über die zu produzierenden Zustände des Vorgangsdokuments.

Neben den Objekten, die einer Aktivität über das Vorgangsdokument zugeordnet werden, kann ihr ein Bearbeiter in Form einer Rolle (beispielsweise als Objekt der Klasse "Sachbearbeiter") zugewiesen werden. Außerdem können Ausnahmen und Maßnahmen zu deren Behandlung festgelegt werden. Dazu gehören beispielsweise Zeitüberschreitungen oder die vom Benutzer festgestellte (und dem Objekt mitgeteilte) Inkonsistenz des bearbeiteten Vorgangsdokuments.

Eine Besonderheit bei der Modellierung von Vorgängen ergibt sich daraus, daß zwischen den Objekten, die dem Vorgangsdokument und den einzelnen Aktivitäten zugeordnet werden, Beziehungen bestehen, die für die Semantik eines Vorgangs wesentlich sind. Daraus folgt, daß die Verwendung von Klassennamen allein nicht hinreichend ist. Wenn beispielsweise mehreren Aktivitäten eines Vorgangs die Rolle "Sachbearbeiter" zugewiesen ist, wird daraus nicht deutlich, ob für jede Aktivität derselbe Sachbearbeiter zuständig sein soll. Ebenso ist durch die Verwendung von Klassenbezeichnern allein nicht festgelegt, ob ein Objekt der Klasse "Versicherungsvertrag" in einer Aktivität mit einem gleichartigen Objekt in einer anderen Aktivität übereinstimmt. Es ist also letztlich notwendig, die jeweils zugeordneten Objekte nicht allein durch ihre Klasse, sondern auch durch ihre Identität oder ihre Beziehung zu anderen Objekten zu kennzeichnen. Gängige Ansätze zur Vorgangsmodellierung abstrahieren in der Regel von diesem Problem: Die Beziehungen zwischen Instanzen werden entweder gar nicht

1. Peters/Schultz (1993) skizzieren eine objektorientierte Rekonstruktion von Petri-Netzen, die davon abweicht. Danach wird eine Transition nicht durch das Verhalten eines Objekts (der Klasse "Aktivität") repräsentiert, sondern lediglich durch eine "operation" - wobei nicht darauf eingegangen wird, von welcher Klasse das zugehörige Objekt ist.

berücksichtigt oder ihre Spezifikation wird der Anwendungsprogrammierung überlassen.

In dem hier vorgeschlagenen Ansatz sind zwei Maßnahmen vorgesehen, die Identität von Objekten zu beschreiben. Wenn mehrere Instanzen einer Klasse, die untereinander nicht in Beziehung stehen, in einem Vorgang vorkommen, werden sie durch den Klassennamen und einen Zusatz gekennzeichnet. Dieser Zusatz könnte gegebenenfalls der Name einer Rolle sein, die die Verwendung der Instanz in dem jeweiligen Kontext kennzeichnet. Falls es solche diskriminierenden Rollen nicht gibt (weil beispielsweise die verschiedenen Instanzen einer Klasse die gleiche Rolle spielen), können systematische Identifikatoren (beispielsweise durch Nummerierung) zugeordnet werden.

Ein Vorgang muß nicht völlig deterministisch in dem Sinn ablaufen, daß feststeht, welche Aktivitätsinstanz (die sich von anderen beispielsweise durch den konkret zugeordneten Sachbearbeiter unterscheidet) nach Abschluß einer anderen Aktivitätsinstanz instanziert oder initialisiert werden soll. Vielmehr empfiehlt eine dynamische Optimierung, Aktivitätsinstanzen im Hinblick auf bestimmte Ziele auszuwählen. Ein solches Ziel könnte die Maximierung des Durchsatzes sein. Daneben ist sicherzustellen, daß beim Auftreten von Ausnahmen der Vorgang wieder in einen konsistenten Zustand zurückversetzt wird. Im Hinblick auf die konzeptuelle Beschreibung eines Vorgangs ist es wesentlich, festzulegen, wo solches Verwaltungswissen abzulegen ist. Vor dem Hintergrund des Ziels Wiederverwendbarkeit ist es sinnvoll, diese vorgangsspezifischen Angaben nicht den Aktivitäten zuzuordnen: Sonst wäre es kaum möglich, sie auch in anderen Vorgängen einzusetzen.

Wir führen deshalb ein dediziertes Verwaltungsobjekt ein, dessen Klasse von einer generellen Klasse "Vorgangsmanager" erbt. Ein solcher spezieller Vorgangsmanager könnte über das skizzierte Wissen verfügen. Das Vorgangsdokument und alle Aktivitätsinstanzen eines Vorgangs registrieren sich bei ihm, so daß er jederzeit auf diese Objekte zugreifen kann. Wenn eine Aktivität terminiert (also die Postcondition eines Dienstes erreicht wird, der einen - für diese Aktivität - terminalen Zustand des Vorgangsdokuments produziert), dann wird der Vorgangsmanager genauso davon unterrichtet wie beim Auftreten einer Ausnahme. Im Anschluß daran veranlaßt er das weitere Vorgehen. Ein Beispiel dafür ist die Auswahl eines Sachbearbeiters. Dazu könnte der Vorgangsmanager die Liste der verfügbaren Sachbearbeiter durchgehen, um festzustellen, welcher - im Hinblick auf bestimmte Ablaufziele - am besten geeignet ist. Auch die generelle sowie spezielle Vorgangsmanager-Klassen werden nach der Vorgabe des Objekt-Metamodells beschrieben. Dabei wird offengelassen, ob ein Vorgangsmanager jeweils nur einen Vorgang einer bestimmten Art verwalten kann oder mehrere.[1]

Abbildung 31 zeigt mögliche Beschreibungen eines Vorgangsdokuments, einer Aktivität und eines Vorgangsmanagers - jeweils für die generellen Klassen. In Abbildung 32 finden sich Beispiele von daraus spezialisierten Klassen. Dabei wird unterstellt (was nicht notwendigerweise so sein muß), daß ein Aktivitätsobjekt weiterlebt, nachdem es einen terminalen Dokumentzustand produziert hat. Da es sich nicht um eine Spezifikation handelt, sondern lediglich um eine Veranschaulichung der Funktionalität der Objekte, beschränkt sich die Darstellung auf eine Beschreibung charakteristischer Dienste. Es wird also kein Anspruch auf Vollständigkeit erhoben.[1] Das gilt nicht zuletzt für die Objekte, die im Vorgangsdokument gehalten werden: Sie liefern einen wesentlichen Teil der Dienste, die in einem Vorgang genutzt werden. In Abbildung 33 ist ein Beispiel für ein partielles Objektmodell von Vorgangsobjekten und deren Integration mit anderen Objekten des Objektmodells dargestellt.

Das skizzierte Vorgangsmodell determiniert die Beschreibung von Vorgängen nicht. Ähnlich wie der zuvor dargestellte Ansatz zur Konzeptualisierung von Objekten handelt es sich vielmehr um ein Metamodell, in dem Konzepte zur Beschreibung von Vorgängen bereitgestellt werden. Dabei geht es vor allem darum, zu zeigen, wie ein Vorgangsmodell und ein Objektmodell integriert werden können: Indem die Abwicklung eines Vorgangs mit Hilfe geeigneter Objekte nach Maßgabe des Objekt-Metamodells beschrieben wird und dabei gleichzeitig auf Objekte referiert wird, die im Objektmodell definiert sind.

1. Im Hinblick auf eine unternehmensweite Vorgangsoptimierung könnte es wünschenswert sein, einen Vorgangsmanager einzusetzen, der mehrere gleichartige Vorgänge überwacht. Da auch verschiedenartige Vorgänge gleiche Ressourcen beanspruchen können, könnte es darüber hinaus sinnvoll sein, einen Manager der Vorgangsmanager einzusetzen.

1. So gibt es beispielsweise neben den beschriebenen Diensten, die bestimmte Zeiten liefern, auch solche, die es erlauben, einige dieser Zeiten zu setzen.

Klasse: **Vorgangsdokument**		
Dienste	vorgangsManager:	erlaubt die Zuweisung des zuständigen Vorgangsmanagers.
	vorgangsManager	liefert den zuständigen Vorgangsmanager.
	existiertSeit	liefert den Zeitpunkt der Instanzierung.
	bearbeitungsDauer	liefert die bisherige Bearbeitungsdauer.
	objekte	liefert die Liste der verwalteten Objekte.

Klasse: **Aktivität**		
Dienste	vorgangsManager:	erlaubt die Zuweisung des zuständigen Vorgangsmanagers
	vorgangsManager	liefert den zuständigen Vorgangsmanager.
	gestartet	liefert den Zeitpunkt der Instanzierung bzw. Initialisierung
	aktiv	liefert einen booleschen Wert, der anzeigt, ob die Aktivität gerade ausgeführt wird.
	beendet	liefert den Zeitpunkt, an dem die Aktivität beendet wurde (falls sie noch nicht beendet ist, ist dies durch einen speziellen Wert - etwa null- anzuzeigen.
	bisherigeZeit	liefert die bisherige Bearbeitungsdauer.
	durchschnittlicheZeit	liefert die durchschnittliche Bearbeitungszeit.
	voraussichtlichBeendet	liefert den Zeitpunkt des voraussichtlichen Bearbeitungsendes - berechnet aus dem Startzeitpunkt und der durchschnittlichen Bearbeitungszeit. Falls die Aktivität schon beendet ist, ist ein spezieller Wert zurückzuliefen.
	maximaleZeit	liefert die Bearbeitungszeit, nach deren Eintreten dem Vorgangsmanager eine Zeitdauer-Ausnahme gemeldet werden muß.
	betreutVon:	erlaubt die Zuweisung des zuständigen Bearbeiters.
	betreutVon	liefert den zuständigen Bearbeiter.

Klasse: **Vorgangsmanager**		
Dienste	vorgangsDokument:	erlaubt die Zuweisung des Vorgangsdokuments.
	vorgangsDokument	liefert das zugehörige Vorgangsdokument.
	aktivitäten	liefert die Liste der aktiven oder beendeten Aktivitäten.
	aktuelleAktivitäten	liefert die Liste der aktuell ausgeführten Aktivitäten.
	verantwortlich:	erlaubt die Zuweisung des für den Vorgang verantwortlichen Mitarbeiters.
	verantwortlich	liefert den für den Vorgang verantwortlichen Mitarbeiter.
	gestartet	liefert den Zeitpunkt, an dem der Vorgang gestartet wurde.
	durchschnittliche Zeit	liefert die durchschnittliche Bearbeitungszeit für Vorgänge der verwalteten Art.
	voraussichtlichBeendet	liefert den Zeitpunkt des voraussichtlichen Endes des Vorgangs - berechnet aus dem Startzeitpunkt und der durchschnittlichen Bearbeitungszeit.
	bearbeitetVon	liefert die Liste der Mitarbeiter, die Aktivitäten betreuen.
	ausnahmen	liefert die Liste der bisher aufgetretenen Ausnahmen.

Abb. 31: Funktionalität genereller Vorgangsobjekte.

Klasse:	**SchadenregulierungsDokument**	
Oberklasse:	Vorgangsdokument	
Dienste	zustände	liefert die Liste der möglichen Zustände des Dokuments - jeweils in Form eines eindeutigen Identifikators, beispielsweise als Zeichenkette.
	aktuellerZustand	liefert in gleicher Weise den aktuellen Zustand
	zustandsbeschreibungen	liefert eine Liste mit den für den jeweiligen Zustand nötigen Objekten deren Zuständen.

Klasse:	**MaterielllePrüfung**	
Oberklasse:	Aktivität	
Dienste	start:	erwartet einen bestimmten Zustand des Vorgangsdokuments. Ruft nach erfolgreicher Überprüfung des Zustands den ersten Bearbeitungsdienst auf.
	prüfeMateriell	kontrolliert den Ablauf der materiellen Prüfung. In Abhängigkeit von speziellen Gegebenheiten werden geeignete Bearbeitungsdienste aufgerufen.
	prüfeUrsache	prüft, ob die angegebene Schadenursache durch den Versicherungsvertrag abgedeckt ist.
	prüfeSchadensumme	prüft, ob die angegebene Schadensumme durch den Versicherungsvertrag gedeckt ist.
	prüfeVorschäden	prüft, ob bisherige Schadenfälle des Versicherungsnehmers eine Regulierung ausschließen
	lehneAb	versieht das Vorgangsdokument mit einem geeigneten Vermerk, terminiert die Aktivität und meldet dies dem zuständigen Vorgangsmanager
	akzeptiere	versieht das Vorgangsdokument mit einem geeigneten Vermerk, terminiert die Aktivität und meldet dies dem zuständigen Vorgangsmanager

Klasse:	**SchadenregulierungsManager**	
Oberklasse:	Vorgangsmanager	
Dienste	starteVorgang:	erwartet Objekt, das den Vorgang auslöst (beispielsweise eine Schadenmeldung), instanziert und initialisiert damit das Vorgangsdokument und die erste Aktivität.
	wähleNachfolger:	erwartet einenZustand des Vorgangsdokuments. Wählt nach gegebenen Dispatch-Regeln eine Nachfolgeaktivität aus und startet sie.
	setzeZurück:	erwartet einen Zustand des Vorgangsdokuments. Veranlaßt das Zurücksetzen des Vorgangs zu diesem Zustand - das heißt alle beteiligten Objekte sind wieder in den Zustand zu bringen, der mit diesem Dokument-Zustand korrepondiert.
	behandleAusnahme:	erwartet den Identifikator einer Ausnahme. Veranlaßt nach Maßgabe von Regeln zu Ausnahmebehandlung weitere Aktionen.

Abb. 32: Funktionalität spezieller Vorgangsobjekte.

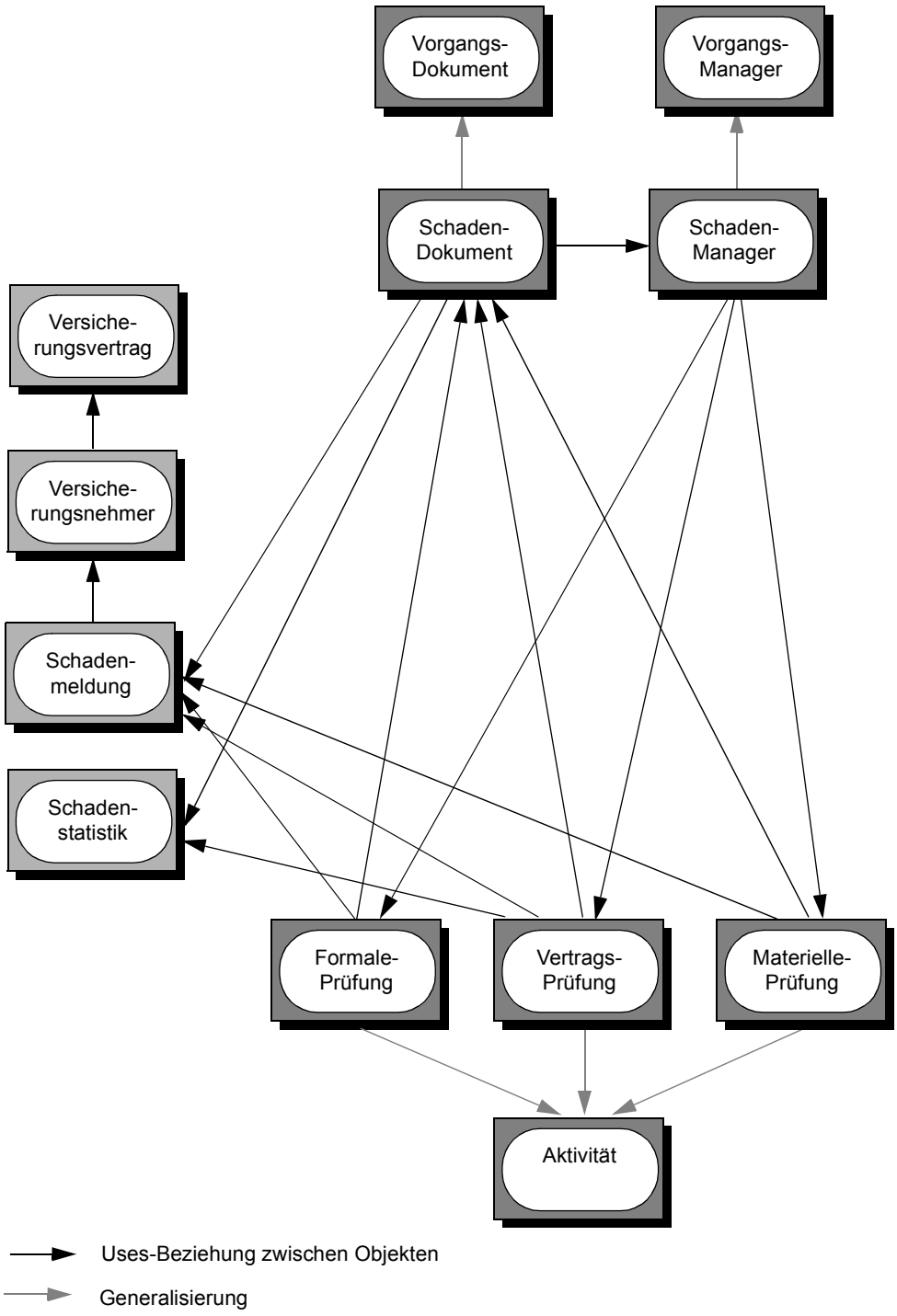


Abb. 33: Beziehungen zwischen Vorgangsobjekten untereinander und zu anderen Objekten des Objektmodells. Um die Übersichtlichkeit nicht noch weiter zu gefährden, sind keine Kardinalitäten eingetragen.

2.1.2.3 Ansatzpunkte für ein schnelles Prototyping

Da ein dynamisches Modell Kontrollstrukturen beinhaltet und - im Fall der Vorgangsmodellierung - an konkreten Anwendungen orientiert ist, bietet sich grundsätzlich die Chance, aus einem solchen Modell einen Prototypen abzuleiten. Dazu ist vor allem auf die Wiederverwendung bereits implementierter Klassen zu setzen. Der dargestellte Ansatz war ursprünglich durch die Hoffnung motiviert, Vorgangsmodelle durch eine Menge vorgegebener Aktivitäten zu erstellen.[1] Wenn diese Aktivitäten als implementierte Klassen vorlägen, könnte auf diese Weise ein Vorgangssystem gleichsam durch das anschauliche Zusammenfügen von Komponenten erzeugt werden. Es hat sich allerdings gezeigt, daß die Komplexität der Zusammenhänge einen bescheideneren Anspruch empfiehlt.[2] Aktivitäten können eben nicht autonom für die Modellierung verwendet werden. So hängt ihre Semantik einerseits von dem je bearbeiteten Vorgangsdokument (oder allgemeiner: von den bearbeiteten Objekten) ab. Andererseits ist zu berücksichtigen, daß Aktivitäten nicht unabhängig vom jeweiligen Vorgangsmanager eingesetzt werden können - auch wenn die Verankerung der Vorgangssteuerung außerhalb der Aktivitäten tendenziell deren Autonomie fördert.

Während die dargestellten Interdependenzen die Kompositionsvision zunichte machen, bleibt dennoch Spielraum für Wiederverwendung. Dabei ist die Hoffnung darauf gerichtet, vorhandene Klassen durch geringfügige Spezialisierung an die Anforderungen eines neuen Vorgangs anzupassen. Wenn es beispielsweise Vorgangsmanager, Vorgangsdokument und die Aktivitäten eines Vorgangs "Schadenbearbeitung-KFZ" vorliegen, können sie auf die Ansprüche eines bestimmten Versicherungsunternehmens spezialisiert werden. Zudem ist es denkbar, eine ähnliche Vorgangsart (wie etwa "Schadenbearbeitung-Unfall") durch Spezialisierung abzudecken. Dabei gilt, daß die Chancen für Wiederverwendung einerseits von der Flexibilität der vorhandenen Klassen, andererseits von deren Zahl abhängen.

Ein Prototyp sollte in anschaulicher Weise die Funktionalität eines zu entwikkelnden Systems abbilden. Dazu ist die Benutzerschnittstelle von wesentlicher Bedeutung. Der hier vorgestellte Ansatz zur Vorgangsmodellierung bietet günstige Voraussetzungen für die weitgehend automatisierte Implementierung einer prototypischen Benutzerschnittstelle. Wenn man davon ausgeht, daß die Sicht eines Benutzers auf einen Vorgang durch die Anforderungen einer Aktivität geprägt ist, kann ein Vorschlag für die Benutzerschnittstelle einer Aktivität gene-

1. Vgl. dazu Frank (1992 a).
2. In anderen Ansätzen, in denen ähnliche Visionen ("glueing of components") propagiert werden, wird der Anspruch entweder implizit reduziert (Nierstrasz et al. 1990) oder aber es wird von Lösungsmöglichkeiten völlig abgesehen (Wiederhold/Wegner/Ceri 1990).

riert werden: In einer Aktivität ist beschrieben, welche Dienste von welchen Objekten genutzt werden. In der Beschreibung der zugehörigen Objektklasse im Objektmodell ist den Objekten, die durch diese Dienste geliefert oder gefordert werden, ein Default View zugeordnet. Auf diese Weise kann also unter Rekurs auf entsprechende Angaben im Objektmodell eine kontextspezifische Benutzerschnittstelle erzeugt werden. In IV.3.2 wird eine Implementierung der prototypischen Erzeugung von Benutzerschnittstellen nach diesem Ansatz dargestellt.

2.2 Die Perspektive der Systemverwalter

Ein konzeptuelles Modell - in dem hier vorgetragenen Ansatz als Konglomerat von Objektmodell und einer Menge von Vorgangsmodellen - dient der anschaulichen Beschreibung der Unternehmensrealität. Es bildet die wesentliche Grundlage für die Verwaltung der in einem Informationssystem abgelegten Informationen. Die Verwaltung eines Informationssystems beschränkt sich allerdings nicht allein auf die abgelegten Informationen. Vielmehr spielen auch solche Sachverhalte eine wesentliche Rolle, von denen im konzeptuellen Modell zunächst bewußt abstrahiert wird: Hardware, Basissoftware, Portierbarkeit und so fort. Während eine solche organisatorische Aufgabe grundsätzlich auch der organisatorischen Perspektive zugeordnet werden könnte, wird die Verwaltung des Informationssystems ihrer besonderen Eigenarten wegen hier als Teil der Informationssystem-Perspektive behandelt.[1] Es geht also letztlich darum, die Teile des Informationssystems, deren Eigenarten mit gutem Grund bei der konzeptuellen Modellierung so weit wie möglich vernachlässigt werden, auf einer anderen Abstraktionsebene wieder in das Modell einzuführen: Nicht als Restriktion der Beschreibung von Realität, sondern als Bestandteil der Realität. Dabei ist zu berücksichtigen, daß eine konsequente Objektorientierung mit einer erheblich gewandelten Vorstellung vom Aufbau und der Nutzung eines Informationssystems einhergeht. Wir werden deshalb zunächst mögliche Auswirkungen auf die Architektur betrachten. Damit zusammenhängend wird skizziert, wie sich der Anwendungsbegriff in objektorientierten Informationssystemen ändern könnte. Schließlich betrachten wir die besonderen Herausforderungen, die sich aus dem Bemühen um Integration und Wiederverwendbarkeit für die Systemverwaltung ergeben - und wie sie in der Modellierung der Informationssystem-Perspektive berücksichtigt werden können.

1. Das schließt Überschneidungen mit Konzepten der organisatorischen Ebene allerdings nicht aus.

2.2.1 Architektur

Traditionell sind Informationssysteme durch eine Schichtenarchitektur gekennzeichnet, die sich in vereinfachender Form so darstellt: Die Hardware wird vom Betriebssystem verfügbar gemacht, darauf setzt ein Datenbankmanagementsystem auf, das wiederum von Anwendungen genutzt wird. Daneben gibt es in der Regel eine Reihe von Anwendungen, die nicht das Datenbankmanagementsystem, sondern unmittelbar die Dateiverwaltung des Betriebssystems nutzen. Wenn es sich um verteilte Systeme handelt, können die Dienste des Datenbankmanagementsystems beispielsweise durch eine Server/Client-Architektur nutzbar gemacht werden: Die Anwendungen laufen lokal auf verteilter Hardware und wenden sich nach Maßgabe eines bestimmten Kommunikationsprotokolls an einen dedizierten Server, der den Zugriff auf ein Datenbankmanagementsystem erlaubt.

Die objektorientierte Modellierung ist darauf gerichtet, durch die Spezifikation anwendungsnaher Klassen die Integration und Integrität von Informationssystemen zu fördern. Um diese Vorteile im Zuge der Implementierung nicht zu gefährden, sollte die Verwaltung der im Modell beschriebenen Objekte nicht verschiedenen Anwendungen überlassen werden. Vielmehr empfiehlt sich der Einsatz eines Objektverwaltungssystems.[1] In Analogie zu Datenbankmanagementsystemen, aber mit einer erheblich erweiterten Funktionalität, ist es die Aufgabe eines Objektverwaltungssystems, die im Klassenschema (also der Implementierung von Objektmodell und Vorgangsmodellen) definierten Integritätsbedingungen zu überwachen. Um an einige Beispiele zu erinnern: Die Kardinalität von Attributen oder Beziehungen gehört ebenso dazu wie die Wahrung referentieller Integrität oder die Ausführung bestimmter Aktionen, wenn vordefinierte Zustände erreicht sind.

Auch für ein objektorientiertes Informationssystem sind die Vorteile einer Schichtenarchitektur nicht von der Hand zu weisen: Sie fördert tendenziell Wartbarkeit und Portierbarkeit. Wenn wir zunächst einen einzelnen Rechner betrachten, sind die beiden untersten Schichten ähnlich wie in traditionellen Architekturen durch das Betriebssystem und ein Objektverwaltungssystem bestimmt. Dabei ist allenfalls zu berücksichtigen, daß in zukünftigen Betriebssystemen eine solche Trennung hinfällig sein könnte: An die Stelle von Datei- und Ressourcenverwaltung tritt aus der Sicht des Betrachters eine Verwaltung von Objekten. Die

1. Dieser Begriff wird dem heute üblichen, aber letztlich wenig konsistenten Begriff "objektorientiertes Datenbankmanagementsystem" (werden nun Objekte verwaltet oder Daten?) im folgenden vorgezogen. In neueren Veröffentlichungen zeichnet sich ein ähnlicher Begriffswandel ab. So sprechen Bode/Cremers/Freitag (1992) und Kelter (1992) von "Objektmanagentsystem", Güntzer/Bayer/Sarre/Werner/Myka (1992) von "Objektbank".

beiden unteren Schichten bieten neben einer generellen Objektverwaltung (Überwachung von Integritätsbedinungen, Garantie von Objektidentität, Ereignismanagement, Versionsverwaltung etc.) eine Reihe genereller Klassen an, die für die Anwendungsentwicklung genutzt werden können. Dazu gehören beispielsweise Klassen, die Kommunikationsdienste anbieten, diverse Zahlen- und Zeichenketten-Klassen, Container-Klassen oder Multimedia-Klassen.

Im Falle der hier unterstellten konsequent objektorientierten Architektur auch der Anwendungen sind die Klassen, die eine Anwendung (beispielsweise ein Desktop Publishing System) konstituieren, im Klassenschema des Objektverwaltungssystems beschrieben. Sie sind damit grundsätzlich auch für andere Anwendungen zugänglich. Auf diese Weise wird eine Integration der Anwendungen auf einem hohen semantischen Niveau möglich: Wenn man in einem Textverarbeitungssystem eine Tabelle, wie sie in Tabellenkalkulationssystemen üblich ist, darstellen möchte, muß dies nicht mehr auf der Ebene des grafischen Austauschformats geschehen, vielmehr kann die Tabelle als Instanz einer auch dem Textverarbeitungssystem bekannten Klasse referenziert werden. Zur Darstellung im Dokument kann die Textverarbeitungssoftware auf Präsentationsdienste der Tabelle zurückgreifen. Auf der anderen Seite kann beispielsweise ein Fakturierungsprogramm auf geeignete Klassen des Textverarbeitungssystems zurückgreifen, um auf diese Weise Editier- und Formatierungsfunktionen zu übernehmen. Darüber hinaus können auch die Dienste domänenspezifischer Klassen (wie Rechnungen, Kunden, Aufträge etc.) über verschiedene Anwendungen hinweg genutzt werden. Auf diese Weise ergibt sich gleichsam ein hierarchisches Geflecht von Klassen: Einige Klassen sind anwendungsnäher als andere, generellere Klassen. Während anwendungsnähere und generellere Klassen sich jeweils untereinander benutzen können, gibt es daneben eine einseitige Nutzung generellerer Klassen durch anwendungsnähere.

Wenn ein Informationssystem weitgehend in einem Klassenschema beschrieben ist und die zugehörigen Instanzen von einem Objektverwaltungssystem verwaltet werden, ergeben sich sehr günstige Voraussetzungen dafür, ein hohes Maß an Integration (damit einhergehend: Wiederverwendung) und Integrität zu erreichen. Gleichzeitig ist allerdings zu berücksichtigen, daß mit einer solchen Architektur auch besondere Schwierigkeiten verbunden sind. Im Unterschied zu Daten können Objekte auch Dienste anbieten. Dieser Umstand ist bei verteilten Systemen mit besonderen Herausforderungen verbunden: Bei der Kommunikation durch das Versenden von Objekten muß sichergestellt werden, daß der Empfänger die Dienste des Objekts nutzen kann. Dabei ist zweierlei zu berücksichtigen: Die Dienste werden durch Prozeduren realisiert, die nicht im Objekt selbst vorliegen, sondern vielmehr in der Klasse des Objekts sowie deren Oberklassen. Es ist also zu klären, wie die Referenzen auf diese Prozeduren, die in einem Objekt

angelegt sind, beim Empfänger befriedigt werden können. Darüber hinaus ist an die Restriktion zu denken, daß in heterogenen Systemen ausführbarer Code einer Maschine auf einer anderen nicht brauchbar ist.

Es wäre grundsätzlich möglich, ein Objekt immer dann, wenn der Empfänger nicht über eine identische Implementierung der Klasse des Objekts verfügt, mit vollständigem maschinenunabhängigen Quellcode zu versehen. Dieser Ansatz ist allerdings wenig sinnvoll. Einerseits kann er bei komplexen Klassen mit einem kaum zumutbaren Zeitaufwand verbunden sein (zudem drohen Namenskonflikte, wenn beim Empfänger einige der referenzierten Klassen ebenfalls definiert sind), andererseits wird auf diese Weise der hier vertretene Integrationbegriff nicht unterstützt: Dienste des empfangenen Objekts sind zwar nutzbar, der Empfänger kennt die Klasse des Objekts aber nicht, es liegt also keine gemeinsame semantische Referenz vor. Dieser Nachteil trifft ebenfalls für einen Ansatz zu, der mitunter im Hinblick auf die Integration gegenwärtig genutzter Software diskutiert wird: Für Objekte einer bestimmten Art (bei heutigen Systemen zutreffender: Dateien) wird die zugehörige Anwendungssoftware - beispielsweise ein Textverarbeitungssystem - in ausführbarem Code für mehrere Prozessor-Architekturen bereitgehalten. Falls der Empfänger nicht über ein entsprechendes Programm verfügt, wird ihm der passende Code mitgeschickt.

Vor dem Hintergrund dieser Überlegungen bleiben vor allem zwei prototypische Architekturvarianten, die in der Realität durchaus in Mischformen auftreten können. Der sinnvollste Ansatz bei verteilter Verarbeitung in heterogenen Systemen ist m.E. zur Zeit darin zu sehen, auf jedem Rechner im Netz eine Kopie des zentral geführten Klassenschemas zu halten. Die damit verbundene Redundanz läßt sich in akzeptabler Weise kontrollieren, indem z.B. allein das Objektverwaltungssystem für die konsistente Aktualisierung der lokalen Kopien zuständig ist. Wenn von einem Arbeitsplatzrechner ein Objekt angefordert wird, wird tatsächlich nur sein Zustand übertragen (wenn sich der aktuelle Zustand nicht bereits in einem lokalen Cache-Speicher befindet), die Ausführung von Objektprozeduren wird dann durch einen Verweis auf die lokal implementierte Klasse realisiert. Daneben ist ein anderer Ansatz zu berücksichtigen, der auch von einigen Anbietern objektorientierter Datenbankmanagement-Systeme explizit empfohlen wird: Sämtliche Prozeduren von Objekten laufen innerhalb des Objektverwaltungssystems ab. Dabei könnte es sich durchaus um ein physisch verteiltes Objektverwaltungssystem handeln. Die Kommunikation mit den verteilten Arbeitsplätzen könnte dann beispielsweise über Window-Server erfolgen. Eine m.E. durchaus reizvolle Vorstellung, die allerdings unter anderem im Hinblick auf das Verhalten unter hoher Last sorgfältig geprüft werden sollte. Die Abbildungen 34 und 35 veranschaulichen die beiden Basis-Architekturen für verteilte objektorientierte Informationssysteme.

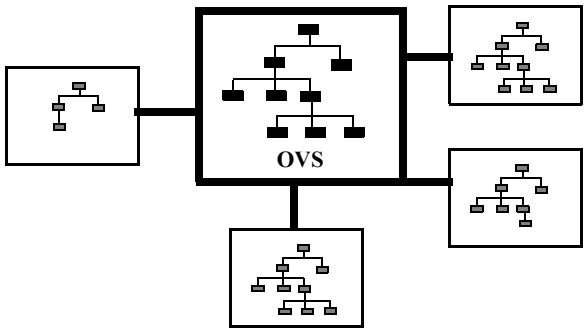


Abb. 34: Die Klassenhierarchie wird logisch zentral im Objektverwaltungssystem (OVS) gepflegt. Die Arbeitsplatzrechner erhalten auf Veranlassung des OVS eine Kopie des gesamten oder von Teilen des Schemas. Die Referenzen auf Methoden werden so innerhalb der Arbeitsplatzrechner befriedigt.

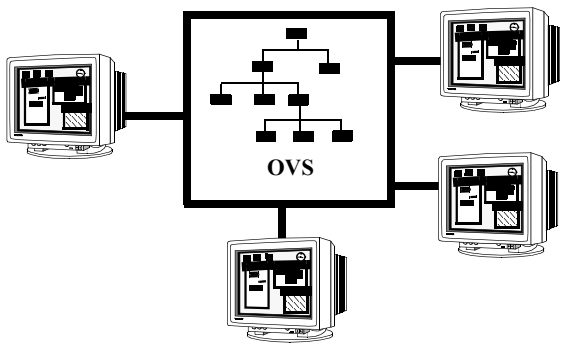


Abb. 35: Die Klassenhierarchie wird auch hier logisch zentral im OVS gepflegt. Sämtliche Objektdienste werden von Prozeduren realisiert, die auf dieser Maschine laufen. Die Rechner an den Arbeitsplätzen sind allein für die Realisation der Benutzerschnittstelle zuständig (z.B. Window-Server).

Je mehr anwendungsnahe Klassen und daraus instanzierte Objekte über verschiedene Anwendungen hinweg benutzt werden können, desto näher rückt eine Überwindung des traditionellen Anwendungsbegriffs - schließlich wird die Semantik einer Anwendung wesentlich durch die sie konstituierenden Klassen bestimmt. Aus der Sicht der Systemverwaltung führt dies dazu, daß Anwendungen nicht mehr nur als (mehr oder weniger) isolierte Programmpakete betrachtet werden können. Es wird sicherlich weiterhin Anwendungsobjekte geben, die sich dem Benutzer wie heutige Anwendungen präsentieren. Daneben aber sind flexiblere, man könnte auch sagen: konfigurierbare Anwendersichten auf das Informationssystem denkbar.

Exkurs: Szenario objektorientierter Anwendersichten

Die Rede von konfigurierbaren Anwendersichten bleibt vage, solange nicht veranschaulicht wird, wie sich eine solche Sicht präsentieren könnte. Im folgenden wird deshalb mit Hilfe eines Szenarios skizziert, in welcher Weise die Nutzung eines Informationssystems durch eine Neuorientierung des Anwendungsbegriffs beeinflußt werden könnte. Dazu ist keine völlig visionäre Betrachtung nötig, vielmehr kann man an bereits existierende Tendenzen anknüpfen. Es sind vor allem zwei Fragen, die aus der Sicht des Anwenders von Bedeutung sind:

- *Navigation/Retrieval*: Wie wird die Suche nach Objekten (und auch nach Klassen) unterstützt? In heutigen Systemen sind die zwischen Anwendungen ausgetauschten "Objekte" zumeist Dateien. Dementsprechend beschränkt sich Navigation auf das Durchsuchen von Dateiverzeichnissen. Für die Suche nach anwendungsnahen Objekten ist allerdings deren Semantik entscheidend. Eine Suche, die vor allem an einer oft willkürlichen Benennung orientiert ist, also von der Bedeutung eines Objekts abstrahiert, ist dazu kaum hinreichend.
- *Interaktion/Präsentation*: Anwendungen haben spezifische Benutzerschnittstellen, deren Bandbreite sich nach dem jeweiligen Einsatzbereich richtet. Wenn dem Anwender die Funktionalität anwendungsübergreifender Objekte verfügbar gemacht werden soll, ist eine solche Einschränkung nicht hinzunehmen.

Die Suche nach Klassen ist ein bekanntes Problem bei objektorientierten Entwicklungssystemen. Das wesentliche Instrument zur Unterstützung dieser Suche sind Browser, die entweder textuell oder grafisch aufgebaut sind. Wenn bei der Suche Beziehungen zwischen Klassen (dabei ist vor allem an Generalisierung zu denken) berücksichtigt werden sollen, dann bieten grafische Browser eine größere Übersichtlichkeit. Darüber hinaus ist es denkbar, daß Klassen mit Deskriptoren versehen werden, um so ein darauf ausgerichtetes Retrieval zu ermöglichen.[1] In ähnlicher Weise wie bei heutigen objektorientierten Entwicklungsumgebungen könnte die Suche zukünftiger Anwender nach Klassen ihres Bedarfs unterstützt werden. Die Suche auf der Instanzenebene (nachdem beispielsweise die Klasse "Sachbearbeiter" gefunden wurde) erfolgt in Analogie zur traditionellen Datenbanksuche, wobei eben nicht nur nach Zuständen gesucht werden kann, sondern grundsätzlich nach den Merkmalen, die von den Diensten eines Objekts geliefert werden.

Es ist sinnvoll, die Interaktion mit einem Objekt und - damit zusammnenhängend - seine Präsentation in dem Kontext stattfinden zu lassen, in dem das Objekt tatsächlich benötigt wird. In rudimentärer Form geschieht das in fortschrittlichen Systemumgebungen dadurch, daß Objekte, die in einer Anwendung erstellt wur-

1. Eine eingehendere Darstellung von Suchmöglichkeiten findet sich in V.2.1.2.

den, in gleicher Darstellung in eine andere übernommen werden können und dort in eingeschränkter Weise weiterbearbeitet werden können (beispielseweise kann eine in einem Grafik-Editor erstellte Grafik in ein Textverarbeitungssystem übernommen werden; dort kann dann ihre Größe verändert werden). Sehr viel weiter reicht die Vorstellung eines "aktiven Dokuments"[1], in dem Objekte verschiedener Art nicht nur präsentiert, sondern auch genutzt werden können. Ähnlich wie in heutigen Grafik-Editoren können Objekte, die mit Hilfe von ihnen zugeordnenten Widgets im Dokument präsentiert werden, selektiert werden. Dann können, über eine objektspezifische Benutzerschnittstelle, Dienste des Objekts ausgewählt werden, so daß das Objekt unter Umständen innerhalb des Dokuments seine Darstellung verändert.

Das folgende Beispiel verdeutlicht die dargestellte Skizze. Der Benutzer öffnet zunächst einen Ordner mit der Aufschrift "Dokument-Arten". Daraufhin wird ihm in Baumdarstellung die Hierarchie der im Unternehmen verwendeten Dokument-Klassen präsentiert. Nach dem Anklicken der Klasse mit der Bezeichnung "Vorstandsvorlage" wird ein leeres Dokument dieser Art geöffnet. Der Benutzer wählt nun im Menu "Transfer" die Funktion "Thesaurus". Daraufhin wird ein weiteres Fenster geöffnet, in dem - wiederum als Baum - die darstellbaren Objekte des Informationssystems präsentiert werden. Nach Anklicken des Objekts "Monatsumsätze" kann der Benutzer seine Anforderungen spezifizieren (Zeitraum, Darstellungsart ...). Nach Bestätigung wird dann im Dokument die entsprechende Grafik erzeugt. In gleicher Weise werden die restlichen Grafiken erstellt. Dies gilt auch für das Objekt "Debitorenkonto", da es eine Methode zur grafischen Darstellung eines Kontos beinhaltet. Die Integration verschiedener Anwendungen (und damit letztlich: ihre Überwindung) erfolgt also durch die Verfügbarkeit von Objekten, die in einer einheitlichen Umgebung benutzt und präsentiert werden können.

1. Es handelt sich dabei nicht um einen etablierten Begriff, vielmehr um die hier gewählte Bezeichnung. Die Vorstellung deutet sich in einigen populärwissenschaftlichen Veröffentlichungen, aber auch in fortschrittlichen objektorientierten Anwendungssystemen (wie beispielsweise in "The Analyst®" von Parc Place Systems) an.

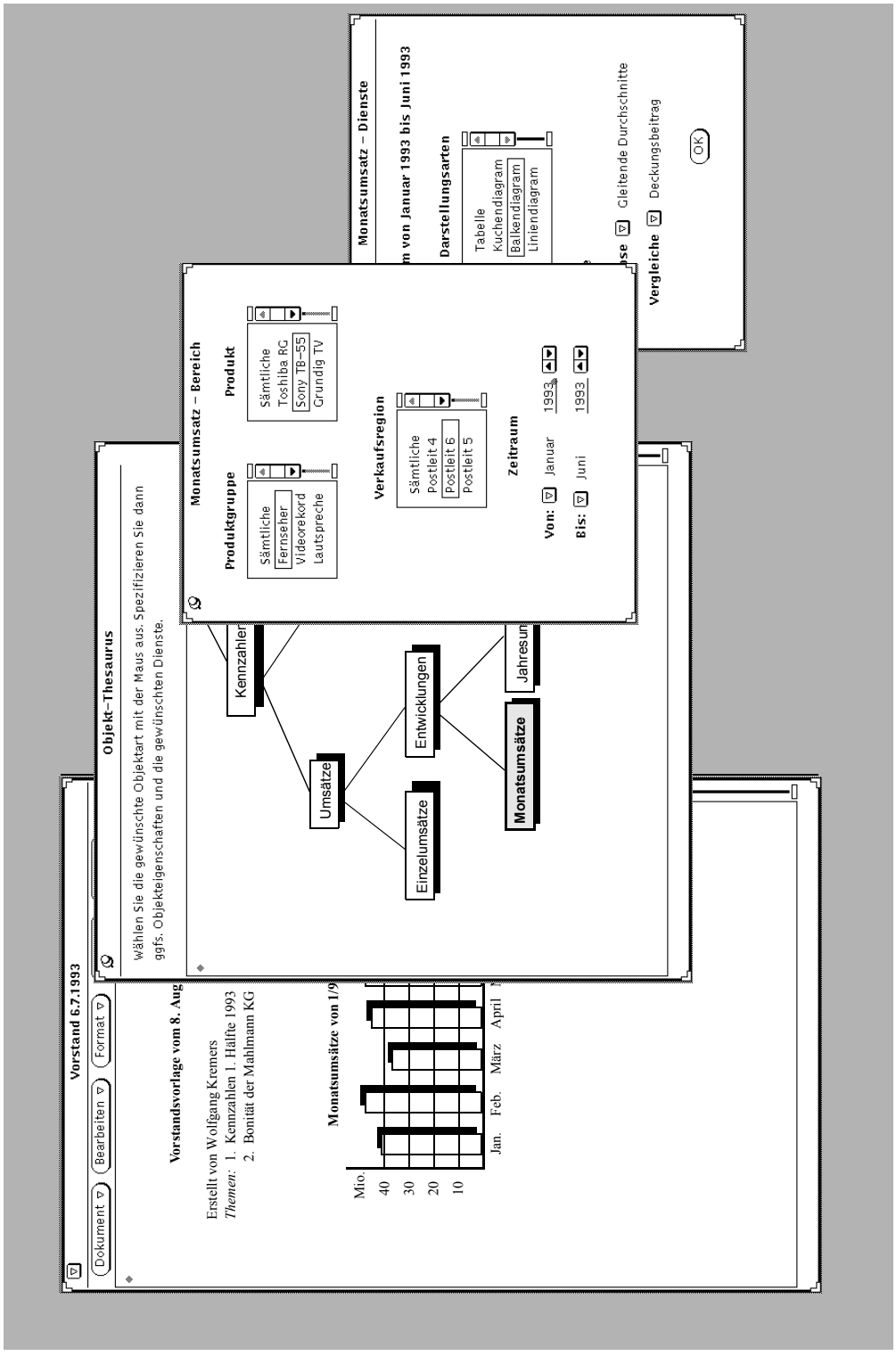


Abb. 36: Beispiel für den Zugriff auf Objekte in einem "aktiven Dokument".

2.2.2 Implikationen für die Systemverwaltung

Die skizzierte Architektur objektorientierter Informationssysteme bietet günstige Chancen und gleichzeitig neue Herausforderungen für eine ökomische Systemverwaltung. Wir betrachten dazu im folgenden vier Aspekte, die anschließend als Grundlage für ein prototypisches Modell dienen, in dem sie untereinander verknüpft werden.

Software-Wartung

In einem Objektmodell sind wesentliche Angaben für die Wartung enthalten. So ist für jede Klasse beschrieben, von welchen anderen Klassen sie in welcher Weise benutzt wird. Es ist allerdings kaum zu erwarten, daß Objektmodelle und die sie implementierenden Klassenschemata vollständig von einem Anbieter - oder von der Anwenderorganisation selbst - erstellt werden. Es scheint realistischer, davon auszugehen, daß Klassen in einem Verbund erstellt werden, der bestimmte Anwendungsbereiche oder -aspekte abdeckt - wie beispielsweise eine Menge von Klassen zur Manipulation grafischer Objekte in einem Editor oder Klassen, die die Durchführung diverser statistischer Analysen erlauben. Coad/ Yourdon (1990, S. 93 f.) nennen einen solchen Klassenverbund "Subject". Rumbaugh et al. (1991, S. 43) sprechen von einem "Module": "Modules enable you to partition an object model into manageable pieces. Modules provide an intermediate unit of packaging between an entire object model and the basic building blocks of class and association."

Im Hinblick auf die Verwaltung ist es wünschenswert, für jede Klasse festzuhalten, zu welchem Verbund sie gehört. Im Grenzfall entspricht ein Verbund heutigen Anwendungen: Ein monolithischer Block (also keine Ansammlung von Klassen), der nach außen eine spärliche Zahl von Diensten anbietet - die Funktionen werden eben vor allem über die Benutzerschnittstelle verfügbar gemacht. Für einen Verbund selbst sind dann Angaben darüber zu erfassen, wann und von wem er erstellt wurde, welche anderen Verbünde er in welcher Version benötigt. Zudem sollte vermerkt sein, in welcher Programmiersprache er erstellt wurde und für welche anderen Sprachen Schnittstellen existieren. Auch wenn es mit Blick auf die Integration wünschenswert ist, daß ein Verbund offen ist in dem Sinn, daß er möglichst alle seine Klassen auch nach außen verfügbar macht, so kann Offenheit nicht so weit führen, daß jeder berechtigt ist, Klassen zu verändern. Eine wirksame Systemverwaltung impliziert - ähnlich wie bei der Datenbankadministration - eine genaue Festlegung der Autorisierung für Änderungen am Schema.

Falls die Nutzung einzelner Klassen (oder eines Klassenverbunds) durch Lizenzvereinbarungen eingeschränkt ist, sollten entsprechende Angaben (Anzahl der Berechtigten, eventuell die Personen) verfügbar gemacht werden. Schließlich ist

es wünschenswert, daß für einen Verbund und/oder eine Klasse die jeweils benötigten oder nutzbaren Ressourcen angegeben werden - soweit dies sinnvoll und möglich ist.[1]

Das folgende Beispiel zeigt die rudimentäre Beschreibung eines Verbunds zu Verwaltungszwecken. Dabei ist zu berücksichtigen, daß differenziertere Angaben über die Kosten und deren Berechnung sinnvoll sein können (Lizenzgebühren, Abschreibungsmodalitäten, Entwicklungskosten, Wartungskosten, Mengenrabatte, Einstands- oder Wiederbeschaffungspreise etc.). Daneben mag es im Einzelfall nötig sein, für jede im Zeitverlauf hinzugefügte Lizenz das Beschaffungsdatum zu vermerken. Zudem könnten die hier vorgeschlagenen Kardinalitäten auch restriktiver formuliert werden. Da eine Klassifizierung der Attribute eine Beschreibung der verwendeten Klassen voraussetzen würde, beschränken wir uns hier darauf, Hinweise auf mögliche Klassen zu geben. "Multimedia" soll dabei verdeutlichen, daß das Merkmal multimedial dargestellt werden kann - eine Darstellung nur als Text ist darin eingeschlossen.

Klasse	**Merkmal**	**Kardinalität**	**Formalisierungshinweise**
Verbund	Bezeichnung	1,1	Zeichenkette
	Beschreibung	0,1	Multmedia
	Versionsnr.	0,1	Zeichenkette
	Versionsdatum	0,1	Datum
	Beschaffungsdatum	0,1	Datum
	Zahl der Lizenzen	0,1	Positive ganze Zahl
	Kosten pro Jahr und Lizenz	0,1	Geldbetrag

Neben diesen Merkmalen (die im Sinne des oben dargestellten Objektmodells Attribute sind) sind diverse Beziehungen zu beachten, die weiter unten dargestellt werden.

Für die Verwaltung von Software ist die Betrachtung von Systemsoftware wie etwa Betriebssysteme oder Objektverwaltungssysteme von großer Bedeutung. Da - wie bereits erwähnt - die konkrete Differenzierung der Komponenten dieser Ebene in unterschiedlicher Weise erfolgen kann, beschränken wir uns hier darauf, Systemsoftware ebenfalls als Verbund zu betrachten. Ihre besondere Bedeutung für andere Verbünde ist dann über geeignete Beziehungen auszudrücken.

1. Es ist zumeist wenig sinnvoll, für jede Klasse beispielsweise den Speicherbedarf eines seiner Objekte anzugeben (wie es in der Methode von Booch optional möglich ist): Der Speicherbedarf kann unter Umständen während der Lebenszeit eines Objekts erheblich schwanken, außerdem ist er ohnehin in der Regel von geringer Bedeutung. Wichtiger sind die peripheren Geräte, die von den Objekten einer Klasse benötigt werden oder deren Nutzung durch die Klasse ermöglicht wird.

Verwaltung von Hardware

Die Hardware eines Informationssystems setzt sich aus einer Vielzahl von Komponenten zusammen. Dabei ist die Unterscheidung von Hardware und Software nicht immer trivial. Das ist beispielsweise dann der Fall, wenn umfangreiche Software (wie etwa Teile eines Betriebssystems) auf einem nicht flüchtigen Speicherchip residiert. Zudem sind die Grenzen zwischen Hardware und Software im Zeitverlauf nicht immer stabil: Funktionen, die heute von Software erfüllt werden, mögen morgen von Hardware erbracht werden - et vice versa. Für die Verwaltung eines Informationssystems ist ein pragmatischer Ansatz gefragt. Dazu unterscheiden wir im folgenden sechs Kategorien von Hardware:

- Rechner
- Interne Komponenten
- Periphere Geräte
- Netzinfrastruktur
- Arbeitsplätze
- Betriebsmittel[1]

Die Konzeptualisierung dieser Kategorien als Klasse oder als Menge von Klassen dient dem Anliegen, eine mögliche Modellierung der Hardware-Verwaltung zu veranschaulichen. In Abbildung 37 ist die Konzeptualisierung statischer Aspekte einiger Klassen veranschaulicht. Die abstrakte Klasse "Hardware" dient dabei der Spezifikation genereller Merkmale (für die Klassen der vier zuerst genannten Kategorien). Die Oberklasse einer Klasse ist unter dem Klassennamen in Klammern angegeben. Abstrakte Klassen sind kursiv gesetzt. Wenn die minimale Kardinalität eines Merkmals als null angegeben ist, so kann dies daran liegen, daß der Erhebungsaufwand im Einzelfall nicht gerechtfertigt sein mag oder auch daran, daß die konkrete Ausprägung nicht verfügbar ist (beispielsweise der Preis eines Prozessors, der in der Rechnung für einen Rechner nicht ausgewiesen ist). In den Fällen, in denen eine Typisierung als Zeichenkette genannt ist, könnte stattdessen eine restriktivere (also semantisch angereicherte) Klasse gewählt werden - also beispielsweise eine Klasse "Prozessorfamilie", deren mögliche Instanzen durch eine Wertemenge beschrieben werden. Es hängt letztlich vom Aufwand ab, ob solche Klassen eingeführt werden sollten.

1. Betriebsmittel sind dabei in einem engeren Sinn zu verstehen als in der Industriebetriebslehre, nach der die gesamte Hardware zu den Betriebsmitteln zählen würde.

	Klasse	Merkmal	Kardinalität	Formalisierungshinweise
	Hardware	Bezeichnung	1,1	Zeichenkette
		Inventarnummer	1,1	Zeichenkette
		Beschafft am	1,1	Datum
		Wartungsintervall	0,1	Zeitintervall
		letzte Wartung	0,1	Datum
		Preis	0,1	Geldbetrag
		Kosten pro Jahr	0,1	Geldbetrag
Rechner	Rechner (Hardware)	Rechnerfamilie	0,1	Zeichenkette
		Hauptspeicherkapazität	1,1	positive ganze Zahl
		Transportfähigkeit	1,1	< hoch, gering, ausgeschlossen >
		Anzahl Steckplätze	1,1	positive ganze Zahl
	Prozessor (Hardware)	Prozessorfamilie	0,1	Zeichenkette
		MIPS	1,1	positive ganze Zahl
	Co-Prozessor (Hardware)	Prozessorfamilie	0,1	Zeichenkette
		Funktion	1,1	Zeichenkette
Interne Komponenten	*Erweiterungskarte* (Hardware)	Steckplatz	1,1	Zeichenkette
		Schnittstelle	0,*	Zeichenkette
	Bildschirmkarte (Erweiterungskarte)	Auflösung	1,*	Tupel von zwei positiven ganzen Zahlen
		Anzahl Farben	1,*	positive ganze Zahl
		Anzahl Graustufen	0,*	positive ganze Zahl
		Videoeingang	1,1	boolescher Wert
	Netzwerkkarte (Erweiterungskarte)	Protokoll	1,*	Zeichenkette
	Modem-Karte (Erweiterungskarte)	Übertragungsrate	1,*	positive ganze Zahl
		Fax-Senden	1,1	boolescher Wert
		Fax-Empfangen	1,1	boolescher Wert

Abb. 37: Beschreibung ausgewählter Klassen der Systemverwaltung

Die Betrachtung der dargestellten Klassen macht zweierlei deutlich:

- Wenn die Hardware eines Informationssystems formal beschrieben ist, wird auf diese Weise nicht nur ein unmittelbarer Auskunftsdienst für die Systemverwalter etabliert. Darüber hinaus ergibt sich die Chance, wissensbasierte Analysen oder Konfigurationen durchführen zu lassen.[1]
- Der Aufwand zur Erfassung der Daten wird vielen Anwenderorganisationen als zu hoch erscheinen. Dieser Einwand wäre jedoch weitgehend obsolet, wenn die Anbieter von Hardware standardisierte Produktbeschreibungen als Teil der Dokumentation mitliefern würden.

Datensicherung

Die Datensicherung kann auf unterschiedlichen Abstraktionsniveaus beschrieben werden. Im einfachsten Fall wird ein Sicherungsverfahren dadurch gekennzeichnet, daß der Inhalt eines Speichermediums komplett zu bestimmten Zeiten auf ein Sicherungsmedium kopiert wird. Daneben sind Verfahren denkbar, die nur die Sicherung von Objekten vorsehen, deren Zustand sich seit der letzten Sicherung verändert hat. Es ist auch möglich, die Objekte bestimmter Klassen von der Sicherung auszuschließen. Wenn ein besonders hohes Sicherheitsniveau erreicht werden soll, können Verfahren zur Echtzeit-Sicherung eingeführt werden, die bei jeder Änderung von Objekten bestimmter Klassen eine Kopie des Objektzustands vorsehen - auf diese Weise werden permanent zwei identische Objektmengen verwaltet. Sicherungsverfahren sollten so weit wie möglich automatisiert und dokumentiert sein. Das schließt die Maßnahmen ein, die zu ergreifen sind, wenn die gesicherten Objekte zur Wiederherstellung eines Informationssystems benötigt werden.

Ein objektorientierter Ansatz legt es nahe, die Sicherungsverfahren mit Hilfe dedizierter Objekte zu beschreiben.[2] Das folgende Beispiel zeigt, wie eine entsprechenden Klasse konzeptualisiert werden könnte. Ein wesentlicher Teil der Semantik ergibt sich durch die Beziehungen zu den zu sichernden Speichern oder Objekten sowie zu den Sicherungsmedien (vgl. dazu Abb. 38).

---

1. Entsprechende wissensbasierte Konfigurationssysteme (ein prominentes Beispiel ist in McDermott 1982 dargestellt) werden mittlerweile von einer Reihe von Herstellern eingesetzt.
2. Daneben ist es denkbar, die Datensicherung in die Verantwortung eines jeden Objekts selbst zu legen. Dies gilt vor allem für solche Verfahren, die bestimmte Zustandsänderungen eines Objekts zum Anlaß für eine Sicherung nehmen.

Klasse	Merkmal	Kardinalität	Formalisierungshinweise
Sicherungs-Agent	Bezeichnung	1,1	Zeichenkette
	Betriebsunterbrechung	1,1	boolescher Wert
	Sicherungstage	1,*	< täglich | Wochentag >
	Sicherungszeiten	1,*	Tageszeit
	Delta-Sicherung	1,1	boolescher Wert
	Operator zur Sicherung	1,1	boolescher Wert
	Operator für Recovery	1,1	boolescher Wert

Kommunikation

Während die Kommunikationsmöglichkeiten zwischen Objekten durch die im Objektmodell festgehaltenen Beziehungen hinreichend beschrieben sind, wird daraus nicht deutlich, in welcher Weise mit externen Partnern kommuniziert werden kann. Dabei ist an dieser Stelle nicht an das Übertragungsmedium, den Kommunikationsdienst oder die Adressierung zu denken, sondern an das semantische Niveau, das jeweils genutzt werden kann. Für die Systemverwaltung bietet es sich dazu an, zunächst diejenigen Klassen zu kennzeichnen, deren Objekte Gegenstand einer Übertragung sein könnten. Anschließend kann für jede dieser Klassen festgehalten werden, ob und wie ihre Objekte übertragen werden können.

Im einfachsten Fall wird ein Objekt (beispielsweise ein Text) zusammen mit einer Adresse einem Übertragungsdienst übergeben. In anderen Fällen ist der Zustand des Objekts zunächst in eine übertragbare Form zu transformieren. Dabei ist zu unterscheiden, ob die Semantik des Objekts erhalten bleibt oder reduziert wird (Beispiel für den zweiten Fall: Ein Objekt der Klasse "Formatierter Text" wird in eine Zeichenkette transformiert). Da nicht zu erwarten ist, daß alle externen Partner in gleicher Weise ausgestattet sind, sind diese Angaben für jeden Partner (oder für homogene Gruppen) zu erstellen. Besondere Bedeutung erhalten dabei Standards oder Konventionen der Daten- oder Objektübertragung. Für jeden der nutzbaren Standards sind dazu die Klassen zu erfassen, deren Objekte dem Standard genügen oder eine Transformation in den Standard erlauben.

Neben den genannten Aspekten ist es für die Systemverwaltung von großer Bedeutung, die Beziehungen von Anwendern, Systementwicklern und -betreuern zum Informationssystem zu modellieren. Dazu gehört die Klassifikation von Zugriffsrechten, die Beschreibung von Verantwortungs- und Zuständigkeitsbereichen. Schließlich kann es sinnvoll sein, Sachverhalte zu modellieren, die ein Controlling des Informationssystems unterstützen. Dazu gehört neben der Zuordnung von Kosten zu Hardware- und Softwarekomponenten die Einrichtung von

Kostenstellen, die Feststellung von Nutzungshäufigkeiten etc.

2.2.3 Das Teilmodell der Systemverwaltung

Im folgenden werden die Beziehungen zwischen den zur Systemverwaltung benötigten Konzepte im Sinne der objektorientierten Modellierung skizziert. Angesichts des Umfangs des Modells werden an dieser Stelle neben der Generalisierungshierarchie allerdings nur Teile der Beziehungen auf Instanzenebene dargestellt.

Die Skizze des Teilmodells der Systemverwaltung macht zweierlei deutlich:

- Die Modellierung ist zwangsläufig mit einer gewissen Willkür behaftet. Das gilt für die grundsätzliche Entscheidung zwischen Einfach- und Mehrfachvererbung ebenso wie für die im Einzelfall gewählte Generalisierung. So könnte man beispielsweise im Unterschied (oder auch in Ergänzung) zu dem hier vorgeschlagenen Ansatz für Drucker eine Generalisierung in "Impact-Drucker" und "Non-Impact-Drucker" vorsehen.
- Die jeweils angemessene Differenzierung hängt wesentlich vom Verwendungszweck ab. Während es für viele Formen der Verwaltung beispielsweise hinreichend ist, eine Klasse "Bildschirm" vorzusehen und die Unterscheidung zwischen den Instanzen durch entsprechende Attribute durchzuführen, kann es dann, wenn eine rigide Überwachung von Konfigurierungsbedingungen erwünscht ist, sinnvoll sein, für jeden Bildschirmtyp eine eigene Klasse zu bilden. Nur dann wäre es beispielsweise möglich, durch Beziehungen und Kardinalitäten auszudrücken, daß ein bestimmter Rechner (der dann auch durch eine von mehreren Rechner-Klassen zu kennzeichnen wäre) einen ganz bestimmten Bildschirm erfordert. Angesichts der möglichen Komplexität von Konfigurierungswissen und den zu seiner Verwendung nötigen Auswertungsverfahren scheint es allerdings angeraten, auf geeignete Ansätze aus dem Bereich wissensbasierter Systeme zurückzugreifen. Dabei ist eine enge Integration mit dem Objektmodell erstrebenswert. Sie kann beispielsweise dadurch erfolgen, daß auf die im Objektmodell spezifizierten Klassen rekurriert wird und die für die Auswertungen benötigten Prädikate durch die Verwendung von Diensten überprüft werden.

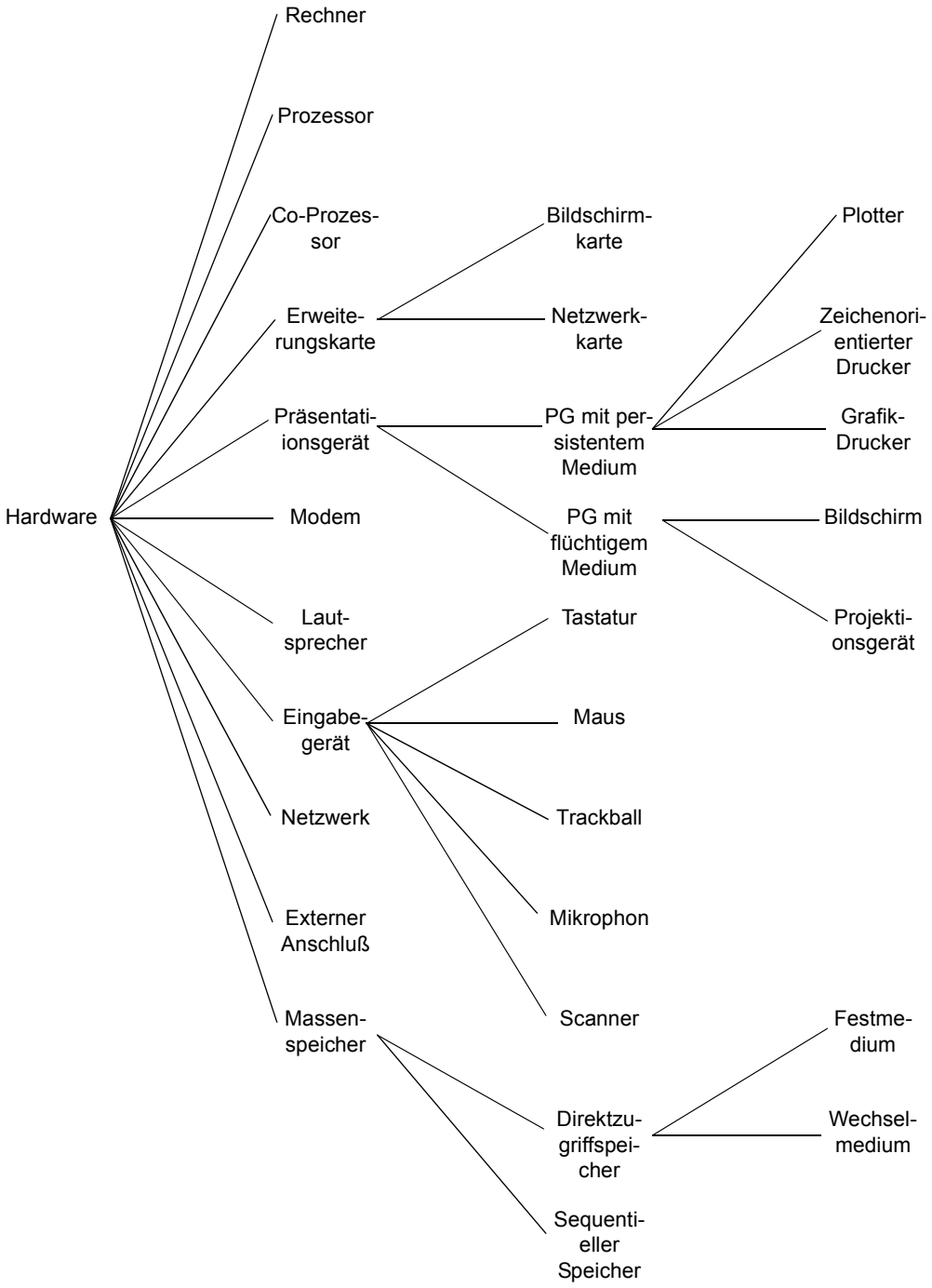
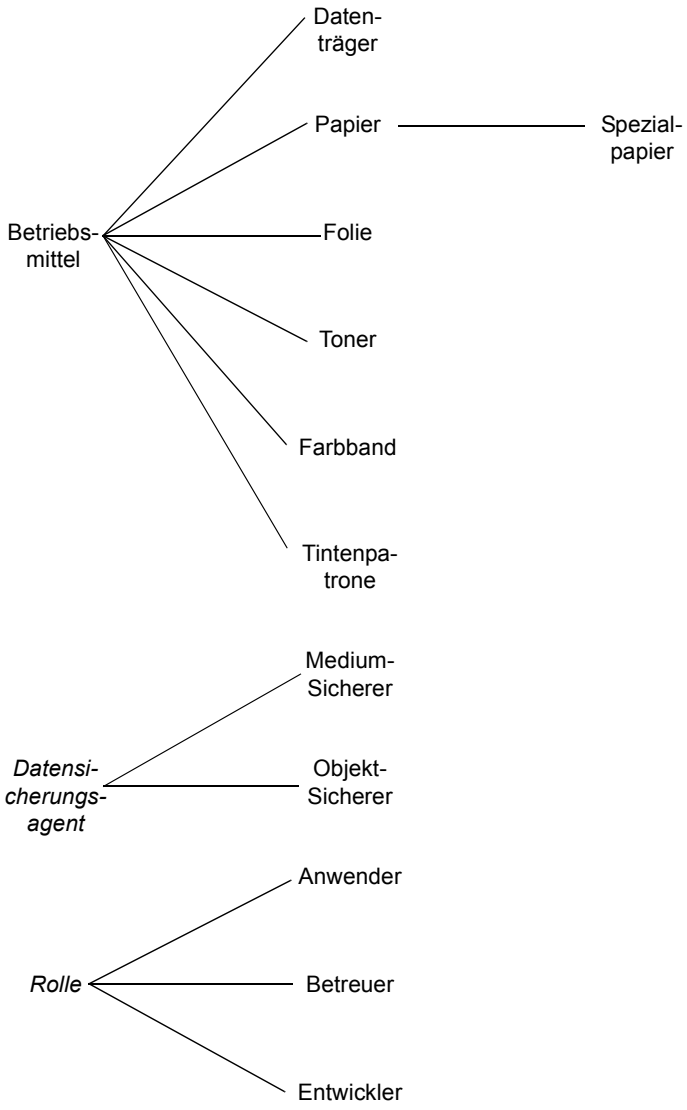


Abb. 38: Generalisierungshierarchie von Klassen für die Systemverwaltung, Fortsetzung folgende Seite


Daten-
träger
Papier
Spezial-
papier
Betriebs-
mittel
Folie
Toner
Farbband
Tintenpa-
trone
Medium-
Sicherer
*Datensi-
cherungs-
agent*
Objekt-
Sicherer
Anwender
Rolle
Betreuer
Entwickler

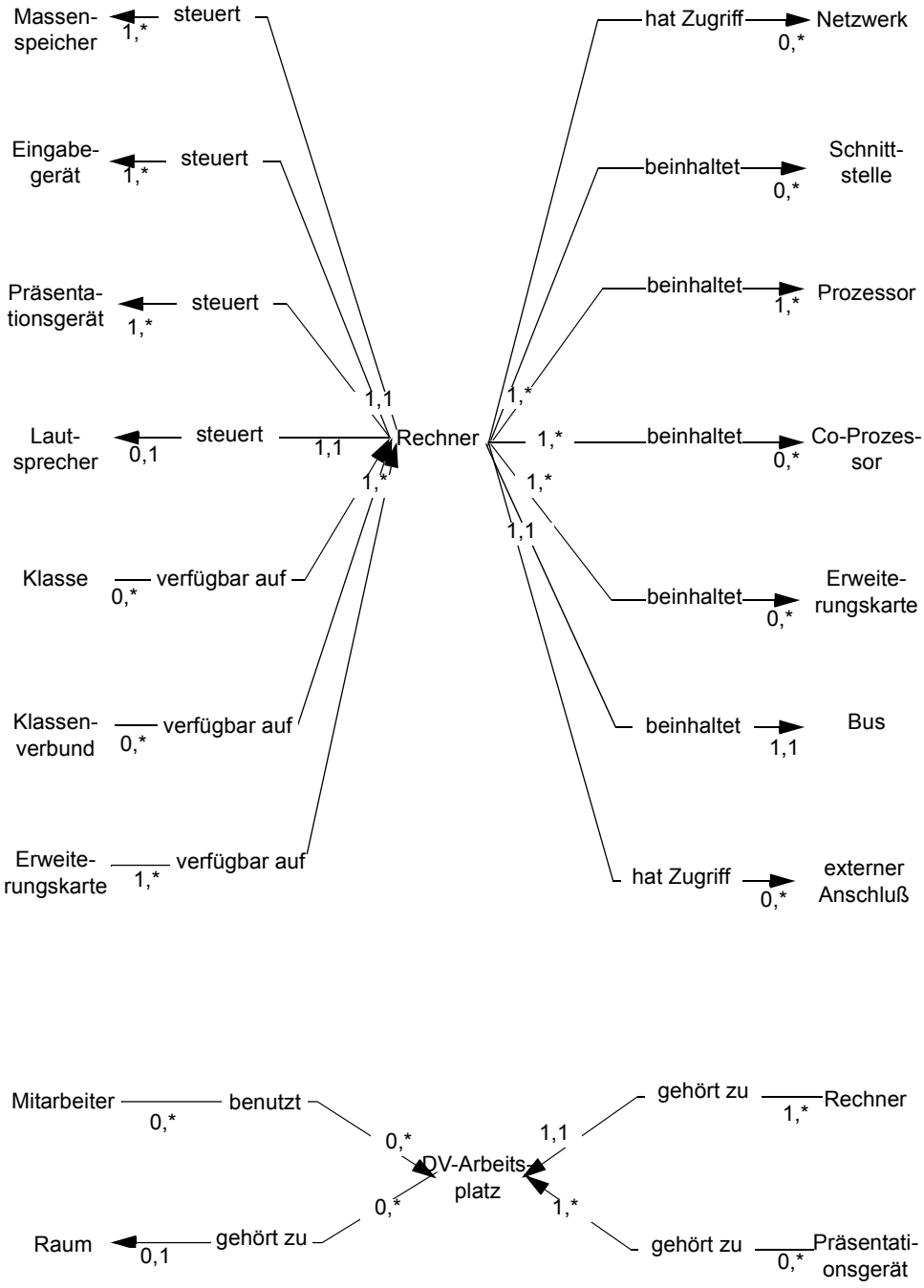
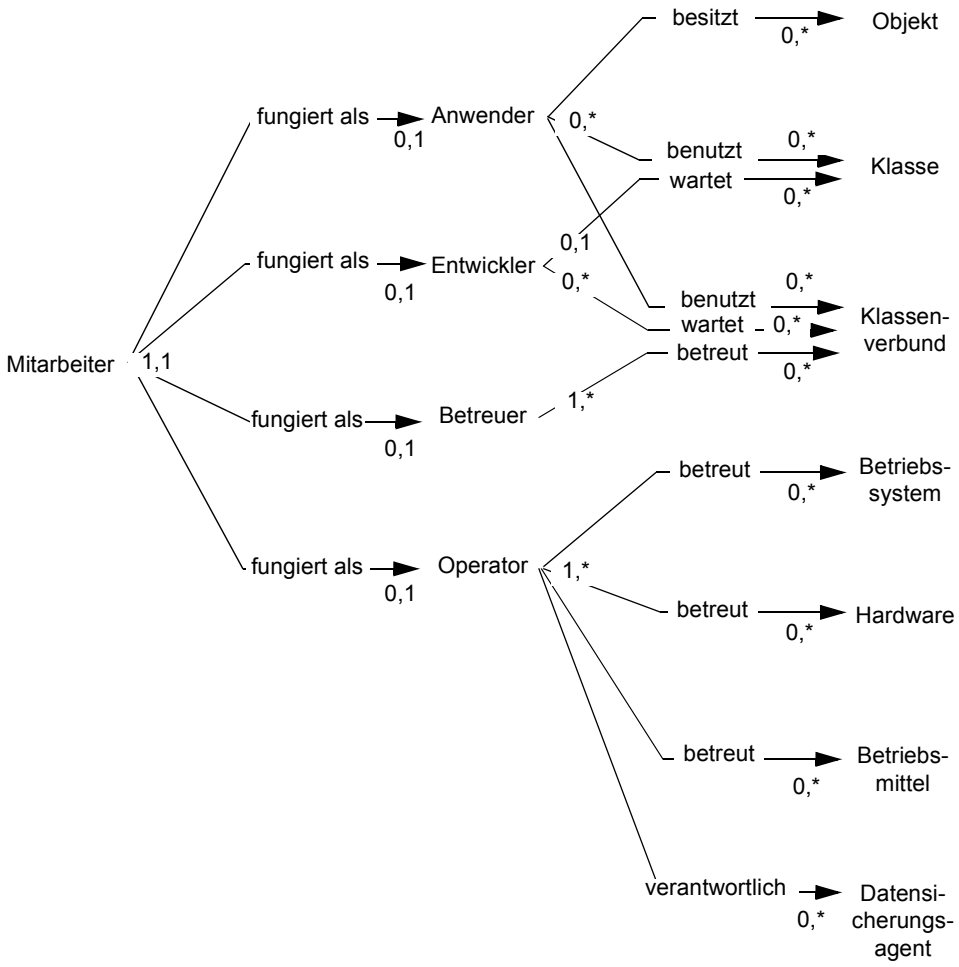


Abb. 39 Beispiele für Beziehungen zwischen Instanzen des Systemverwaltungsmodells, Fortsetzung nächste Seite.



3. Die organisatorische Perspektive

Auch wenn Unternehmen immer weiter von Informationstechnik durchdrungen werden, ist doch die Wahrnehmung der betrieblichen Realität für die Vielzahl der Mitarbeiter von anderen Sachverhalten bestimmt. In dem hier vorgeschlagenen Bezugsrahmen artikuliert sich dieser Umstand in der Einführung zweier weiterer Perspektiven. Die organisatorische Perspektive, so ist es in IV.1.3 vorgesehen, umfaßt die Gestaltung und Durchführung der kooperativen Handlungen im Unternehmen. Dabei ist unstrittig, daß dazu Informationstechnik nicht völlig vernachlässigt werden kann. Es geht hier allerdings - im Sinne des in IV.1.1 entwikkelten Perspektivenbegriffs - um die Darstellung einer anderen Betrachtungsweise des Unternehmens als sie für die Informationssystem-Perspektive sinnvoll

ist. Zur vorläufigen Eingrenzung kann auf drei Aspekte verwiesen werden:

- *Zielgruppe:* Hier ist zunächst vor allem an die Mitarbeiter zu denken, die unmittelbar mit der Organisation betrieblicher Handlungsprozesse befaßt sind. Darüber hinaus sollten aber auch alle anderen Mitarbeiter berücksichtigt werden, denn auch für sie kann ein entsprechendes Modell hilfreich sein, um den Kontext der eigenen Arbeit besser zu verstehen. Das gilt natürlich auch für diejenigen, die mit dem Entwurf und der Implementierung von Informationssystemen befaßt sind: Die Modellierung der organisatorische Ebene sollte ihnen helfen, eine angemessene Vorstellung von den Geschäftsabläufen zu erhalten.
- *Gegenstand und Fokus*: Grundsätzlich umfaßt die organisatorische Perspektive sämtliche Bereiche eines Unternehmens. Um einige zu nennen: die Beschaffung, die Produktion, das Lager oder die Distribution. Der Rahmen unserer Untersuchung läßt jedoch eine so weite Abgrenzung nicht zu. Wir schränken deshalb den Fokus auf generische Elemente der Kooperation im Büro- und Verwaltungsbereich ein. Im Unterschied zur Informationssystem-Perspektive sind dabei auch solche Sachverhalte zu berücksichtigen, die nicht Gegenstand von Automatisierungsprozessen sind.
- *Repräsentation:* Der Zielgruppe und dem Gegenstand entsprechend ist es für die Darstellung dieses Teilmodells wichtig, eine hohe Anschaulichkeit zu realisieren. Das heißt konkreter, daß die Semantik der beschriebenen Sachverhalte nicht immer formalisiert werden muß (mitunter ist dies ohnehin nicht möglich). Stattdessen sollten dem Betrachter Annotationen geboten werden, die die Korrespondenz zur betrieblichen Realität besonders deutlich werden lassen. Im Hinblick auf eine werkzeuggestützte Abbildung ist dabei einerseits an multimediale Darstellungen und die Verwendung von Beispielen zu denken. Andererseits sollten den Betrachtern - ihren unterschiedlichen Interessen entsprechend - verschiedene Detaillierungsstufen zur Wahl angeboten werden.

Die Modellierung organisatorischer Zusammenhänge dient drei Anliegen:

- Im Hinblick auf den Entwurf von Informationssystemen kommt dem Organisationsmodell die Funktion zu, die realweltlichen Zusammenhänge weitgehend unabhängig von Anforderungen formaler Modellierungskonzepte abzubilden - eine ähnliche Funktion also, wie sie mitunter mit der Dokumentation von Domänen-Analysen (Arango 1989, Iscoe et al. 1991) verbunden wird. Im Informationssystem selbst bieten die formalisierbaren Teile des Organisationsmodells die Grundlage für die Verwaltung von Objekten, die von zahlreichen anderen (Anwendungs-) Objekten genutzt werden können.[1]
- Der effiziente Einsatz von Informationstechnik setzt mitunter organisatorische Veränderungen voraus. Das Organisationsmodell sollte die Möglichkeit bieten, organisatorische Alternativen deutlich zu machen. In diesem Zusammen-

hang sollte die Modellierung auch die spezifischen Randbedingungen computergestützter Organisationsanalyse und -gestaltung[1] berücksichtigen.

- Das Organisationsmodell bietet den Mitarbeitern eine Orientierung (ein semantisches Referenzsystem) für gemeinsames Arbeiten. Es unterstützt damit die Koordination der Aktitivitäten im Unternehmen. Das gilt sowohl für die im Modell beschriebenen Konzepte, als auch für die mit Hilfe dieser Konzepte verwalteten Instanzen. Es bildet damit die Grundlage für ein Reservoir organisatorischen Wissens.[2]

Die damit angestrebte Abbildung organisatorischer Zusammenhänge weist deutliche Parallelen zu sogenannten Organisationshandbüchern auf. Müller-Pleuß (1992, Sp. 1507) nennt unter anderem folgende Aufgaben, an denen die Gestaltung eines Organisationshandbuchs auszurichten sei:

- "Einarbeitung und Schulung von Mitarbeitern"
- "Kodierte Regelung und Abgrenzung von Verantwortung und Kompetenz in der Aufbauorganisation und Ablauforganisation des Unternehmens"
- "Anweisungen für Instanzen, Stellen, Arbeitsplätze und Verrichtungen"

Um die organisatorische Perspektive differenziert zu entfalten, betrachten wir im folgenden fünf Teilbereiche, die - wie sich zeigen wird - nicht völlig disjunkt sind: *Organisationsstruktur, Kommunikationsverzeichnis, Informationsmanagement*, *Ressourcenverwaltung* und *Ablauforganisation*. Dabei werden zunächst die wesentlichen Anforderungen an die Modellierung skizziert. Im Anschluß daran wird ein Modell entwickelt, das die Beziehungen zwischen den Teilbereichen verdeutlicht.

3.1 Organisationsstruktur

Die Organisationsstruktur bildet gleichsam das Gerüst einer Organsation. Sie reflektiert vordergründig Regelungen und Koordinationsmechanismen, also formale Organisation. Es ist allerdings unstrittig, daß bei der Gestaltung und Begründung der Organisationsstruktur auch die Bedeutung informaler Organisa-

1. Organisationsmodelle werden deshalb mehr und mehr als Integrationsbasis für kooperationsunterstützende Systeme (CSCW) gefordert. Vgl. dazu Teufel (1992) oder Prinz (1993). Darüber hinaus sind rudimentäre Organsationsmodelle zur Unterstützung der Koordination in verteilten Systemen bereits Gegenstand von Standards, wie etwa X.500 der ISO (ISO 1991) oder - zur Zeit noch in den Anfängen - das "Information Resource Directory System" (ISO 1990).

1. In diesem Bereich gibt es mittlerweile eine Fülle von Ansätzen, die zu Beginn der achtziger Jahre mit der Popularisierung von Expertensystemen neuen Auftrieb erhielten. Ein aktueller Überblick computergestützter Gestaltungstechniken findet sich in Kortzfleisch (1992). Lehner (1991) gibt einen Überblick über Software zur Unterstützung des Organisators. In Schönecker/Nippa (1990), Reichwald (1990) und Frank/Kronen (1991) werden dedizierte Verfahren für den Bürobereich dargestellt.

tion berücksichtigt werden sollte. Der Entwurf und die Pflege der Organisationsstruktur ist die klassische Domäne der organisatorischen Gestaltung. Dazu ist seit geraumer Zeit eine Reihe von Konzepten und Instrumenten bekannt, die in zahlreichen Lehrbüchern[1] dokumentiert sind. Sie liefern eine terminologische Grundlage und geben Hinweise auf Wirkungszusammenhänge. Die unternehmensspezifische Gestaltung der Organisationsstruktur bleibt dessen ungeachtet eine erhebliche Herausforderung, die Luhmann (1977, S. 186) schon vor geraumer Zeit in treffender Weise beschrieben hat:

> "Diese Struktur muß bestimmt und unbestimmt zugleich sein - bestimmt, um als Entscheidungsprämisse die Auswahl der Informationen und der auf sie hin zu veranlassenden Kommunikationen anleiten zu können; unbestimmt, um möglichst viel Umweltkomplexität und -veränderlichkeit ohne Strukturänderung absorbieren zu können."

Es geht hier allerdings nicht vorrangig darum, einen Beitrag zur Optimierung von Organisationsstrukturen zu leisten. Wir beschränken uns vielmehr darauf, einen Ansatz für eine im Gesamtkontext der Unternehmensmodellierung angemessene Abbildung der Organisationsstruktur zu entwerfen. Die auf diese Weise erstellten Modelle mögen dann für eine weitere analytische Durchdringung und daran anknüpfende Reorganisationsmaßnahmen hilfreich sein.

Zur Beschreibung der Organisationsstruktur greifen wir auf eine Differenzierung zurück, die in dieser oder ähnlicher Form in der einschlägigen Literatur weit verbreitet ist. Kieser/Kubicek (1983, S. 79 ff.) sprechen dabei von "Hauptdimensionen"[2] der Organisationsstruktur. Grochla (1982, S. 96 ff.) betont stärker den Gestaltungsprozeß und spricht deshalb von "Aktionsparametern", nämlich: Arbeitsteilung, Koordination und Konfiguration. Die folgende Darstellung ist an den genannten Lehrbüchern orientiert.

---

2. Dieser Aspekt der Wiederverwendung wird in der Organisationsforschung seit einiger Zeit unter Schlagwörtern wie "Organizational Knowledge", "Organizational Memory" und "Organizational Learning" diskutiert. Vgl. etwa Carley (1992), Lyles/Schwenk (1992), Daft/Weick (1984) oder Sims/Gioia (1986). Einige Verfasser widmen dabei dem Einsatz von Informationstechnik zum Zweck der Aufbewahrung und Vermittlung des kollektiven Erfahrungsschatzes besondere Aufmerksamkeit - so etwa Walsh/Ungson (1991) oder Huber (1990). Daneben gibt es eine Reihe von Arbeiten über den Entwurf von "Organisationsdatenbanken" oder "Organisationswissensbasen". Dazu gehören: Chrapary et al. (1991), Kurpicz (1990), Heilmann (1989), Frank (1989), Martial/Victor (1988), Schöllmann (1986).

1. Um einige Beispiele zu nennen: Daft (1992), Schmidt (1991), Remer (1989), Kieser/Kubicek (1983), Grochla (1982).

2. Kieser und Kubicek nennen außer den unten genannten drei Dimensionen noch "Formalisierung" und "Kompetenzverteilung" als weitere Dimensionen. Formalisierung im Sinne von schriftlicher Fixierung bezeichnet allerdings weniger Eigenschaften der Struktur als vielmehr die Formen ihrer Dokumentation und Vermittlung. Kompetenzverteilung wird hier als Bestandteil der Koordination betrachtet.

Arbeitsteilung

Die Differenzierung der in einem Unternehmen zu erfüllenden Aufgaben spiegelt sich in der Abgrenzung organisatorischer Einheiten. Dazu kann auf verschiedene Strukturierungsprinzipien zurückgegriffen werden. Die wichtigsten Prinzipien sind die funktionsorientierte Differenzierung (wie etwa Beschaffung, Produktion, Vertrieb etc.), die objektorientierte Differenzierung (beispielsweise in Form einer Divisionalisierung nach Produktgruppen) sowie als Mischform die mehrdimensionale Differenzierung (in Form von Matrix- oder Tensor-Strukturen). Das jeweils gewählte Strukturierungsprinzip kann gegebenenfalls unter Hinweis auf Unternehmensziele und -strategien sowie die unterstellten Wirkungszusammenhänge begründet werden.

Die kleinste organisatorische Einheit ist die Stelle. Sie kann entweder nach der Art der Verrichtung oder aber nach der Menge der zu verrichtenden Arbeit abgegrenzt werden. Stellen sind neben einer mehr oder weniger restriktiven Aufgabenbeschreibung durch eine Darstellung der erforderlichen Qualifikation zu charakterisieren. Für Stellen mit einer schwach strukturierten Aufgabenstellung kann eine solche Charakterisierung nicht immer mit formaler Exaktheit durchgeführt werden. Stattdessen müssen generelle Prinzipien oder Richtlinien verwendet werden, deren Interpretation durch Erläuterungen und Beispiele angeleitet werden kann. Hier ist etwa an das Auftreten gegenüber Kunden zu denken. Mitunter werden Stelle und organisatorische Rolle synonym verwendet. Es ist allerdings sinnvoll, hier eine sprachliche Regelung zu wählen, die eine präzisere Modellierung zuläßt - und dabei gängige Sichtweisen in der Praxis berücksichtigt: Danach kann ein Mitarbeiter nur eine Stelle bekleiden, deren Bedeutung unter anderem in der Konfiguration (s. unten) der Organisationsstruktur beschrieben ist, während er mehrere organsitorische Rollen wahrnehmen kann.

Koordination

Die Koordination dient dem Anliegen, die Aktivitäten der organisatorischen Einheiten auf die Unternehmensziele auszurichten. Für jede organisatorische Einheit können Ziele[1] oder, konkreter, Aufgaben vorgegeben werden. Die Dokumentation der Organisationsstruktur kann zusätzlich mit Semantik angereichert werden, wenn Beziehungen zwischen diesen Zielen dargestellt werden, also Ziel-Mittel-Gefüge transparent gemacht werden. Generelle Richtlinien für die Koordination können in Form von Führungsgrundsätzen dargestellt werden. Sie lassen sich unter anderem in Management-By-Konzepten und anderen Formen der explizit gemachten Unternehmenskultur artikulieren.

Eine konkretere Form der Koordination spiegelt sich in der Kompetenzvertei-

1. Ziele können dabei, wie in Grochla (1982) vorgeschlagen, in Sach- und Formalziele differenziert werden.

lung. Durch sie wird allgemein der Zuständigkeitsbereich von organisatorischen Einheiten festgelegt. Auf einzelne Stellen heruntergebrochen kann die Kompetenz differenziert werden in Entscheidungsbefugnisse und Weisungsbefugnisse. Entscheidungsbefugnisse werden beschrieben, indem für bestimmte Aufgabenbereiche oder organisatorische Einheiten Entscheidungskriterien und Entscheidungsspielräume vorgegeben werden (durch konkrete Richtlinien oder auch weniger eingeschränkt - beispielsweise durch die Zuteilung von Budgets). Grundsätzlich kann dabei differenziert werden zwischen Entscheidungskompetenzen, die einer Person und solchen, die Gruppen zugeordnet sind. Im Hinblick auf die Autorisierung und Dokumentation von Entscheidungen sind dabei gegebenenfalls Zeichnungsbefugnisse zu definieren sowie Verweise auf Dokumente anzulegen.

Zur Ausgestaltung der Weisungsbeziehungen kann zwischen Einlinien- und Mehrliniensystem unterschieden werden - je nachdem, ob eine Stelle nur einer oder mehreren anderen Instanzen unterstellt sein darf. Das Weisungsrecht kann auf einen Gegenstand (Produkt, Maschine) oder auf eine bestimmte Aktivität bezogen werden. Stellen, denen Weisungsbefugnis zukommt, werden auch als Instanzen bezeichnet. Bei der Verteilung der Kompetenzen sollte das "Kongruenzprinzip" (Grochla 1982, S. 102) beachtet werden, wonach Aufgabe, Kompetenz und Verantwortung kompatibel sein sollten. Eine besondere Stellung in der Hierarchie der Weisungsbeziehungen nehmen sogenannte Stabstellen sein. Sie werden gemeinhin dadurch definiert, daß ihnen innerhalb der Linie keine Weisungsbefugnis zukommt oder, inhaltlich, daß sie Dienstleistungs- und Beratungsfunktionen für Instanzen wahrnehmen.[1]

Die Koordination kann zudem mit Hilfe von Anreizsystemen gefördert werden. In der Regel werden dazu die in einer Stelle zu erbringenden Leistungen operationalisiert - unter Umständen nach verschiedenen Kriterien.[2] Andere Anreize, die an bestimmte Stellen geknüpft sind, stehen nicht in unmittelbarem Zusammenhang mit der Aufgabenerfüllung. Hier ist an Privilegien wie die Nutzung von Dienstwagen einer bestimmten Klasse, das Büromobiliar etc. zu denken.

Konfiguration

Die Bildung von Organisationseinheiten und die Definition der zwischen ihnen bestehenden Weisungsbeziehungen führt zur Frage nach der angemessenen Zahl hierarchischer Ebenen und - damit zusammenhängend - nach der sinnvollen Leitungsspanne[3] der vorgesetzten Stellen. Die Festlegung von Hierarchie und Lei-

1. Zur Problematik einer solchen terminologischen Abgrenzung vgl. Kieser/Kubicek (1982, S. 145 f.).

2. Für Außendienstmitarbeiter könnten solche Kriterien beispielsweise sein: Umsatz, Deckungsbeitrag, Anzahl neuer Kunden, Umsatz eines bestimmten Produkts (jeweils für eine bestimmte Periode).

tungsspanne - gewöhnlich in Form von Organigrammen dargestellt - wird als Konfiguration der Organisationsstruktur bezeichnet. Die Konfiguration setzt wesentliche Randbedingungen für die Kommunikationsbeziehungen in Unternehmen - worauf noch einzugehen sein wird. Im Hinblick auf die Modellierung der Organisationsstruktur muß die Konfiguration nicht explizit behandelt werden, da sie sich indirekt durch die Organisationseinheiten und deren Beziehungen ergibt.

Einen Sonderfall bilden temporäre Organisationseinheiten, wie sie bei der Zusammenstellung von Projektteams entstehen. Bei der Festlegung der Aufgaben und Kompetenzen der Projektmitglieder ist darauf zu achten, Konflikte mit den Definitionen der jeweiligen Stellen zu vermeiden. Dies gilt insbesondere für die Weisungsbefugnisse von Projektmanagern.[1]

Durch die Modellierung der Organisationsstruktur ergibt sich die Chance, formale Verfahren für die Analyse einer gegebenen Struktur und das Generieren von Gestaltungsalternativen anzuwenden. In Kortzfleisch (1992, S. 118 ff.) werden entsprechende wissensbasierte Ansätze kritisch rezipiert. Sie basieren auf Hypothesen über Zusammenhängen zwischen (vage formulierten) Umwelteigenschaften (wie etwa "complexity is complex") und der je erstrebenswerten Struktur (beispielsweise "mixed decentralized"). Auch wenn es angeraten ist, solchen Systemen mit kritischer Distanz zu begegnen, kann es doch für die Bewertung der Organisationsstruktur eines Unternehmens hilfreich sein, wenn mögliche Beziehungen zu Einflußgrößen (wie etwa zu den noch zu entwickelnden Konzepten der strategischen Ebene) abgebildet werden.

3.2 Ressourcenverwaltung

Die für die Arbeit im Büro- und Verwaltungsbereich benötigten Ressourcen werden mitunter in Sachressourcen und Personal unterteilt (Teufel 1992). Da Personal in IV.3.1 und in IV.3.3 behandelt wird, beschränken uns an dieser Stelle auf Sachressourcen - genauer: einen Teil derselben, da die Informationstechnik bereits berücksichtigt wurde. Informationen stellen besonders wichtige Ressourcen da. Deshalb und wegen ihrer spezifischen Eigenheiten werden sie im Zusammenhang mit dem Informationsmanagement getrennt behandelt.

Die verbleibenden Sachressourcen können danach unterschieden werden, ob sie mobil sind oder nicht. Zu der ersten Gruppe gehören Gebäude und Räume. In die

3. Die Leitungsspanne mißt sich an der Zahl der jeweils untergeordneten Stellen.

1. Hier wird in der Literatur zwischen einem "Einfluß-Management" (der Projektmanager hat keine Weisungsbefugnisse) und dem "reinen Projektmanagement" (der Projektleiter hat bezüglich des Projektgegenstands volle Entscheidungs- und Weisungskompetenz) unterschieden. Vgl. Grochla (1982, S. 276 ff.) und Kieser/Kubicek (1982, S. 146 ff.).

zweite Gruppe fallen nicht in das DV-System integrierte kommunikationsunterstützende Geräte wie Telefon und Fax sowie diverse Büromaschinen: Kopiergeräte, Frankiermaschinen, Aktenvernichter, Schreibmaschinen etc. Bei der Modellierung dieser Geräte ist es erstrebenswert, ihre Funktionalität zu konzeptualisieren. Auf diese Weise wird es möglich, durch eine automatisierte Analyse mögliche Substituierungen durch funktional äquivalente Geräte aufzudecken. Neben den Büromaschinen kann das Mobiliar abgebildet werden. Dabei können Beziehungen zu Räumen und gegebenenfalls zu Geräten (wie etwa: "Schreibmaschine benötigt Schreibmaschinentisch") deutlich gemacht werden. Daneben ist es denkbar, daß Mobiliar nicht einem bestimmten Raum, sondern einer Stelle oder einem Arbeitsplatz zugeordnet ist. Ähnliches gilt für Büromaschinen. Dabei ist der Begriff Arbeitsplatz weiter zu fassen als im Kontext der Systemverwaltung: Er setzt sich zusammen aus einer Reihe von Sachmitteln. Dazu kann auch ein DV-Arbeitsplatz gehören, der allerdings nicht innerhalb des Organisationsmodells beschrieben wird. Stattdessen ist ein Verweis auf das entsprechende Objekt in der Informationssystem-Ebene vorzusehen.

Die Beschreibung von Räumen sollte Aufschluß über ihren Standort geben (um auf diese Weise die zwischen einzelnen Räumen zurückzulegenden Wege abschätzen zu können). Außerdem sollte die Ausstattung mit Kommunikationsanschlüssen (Telefon, Fax, LAN) verzeichnet sein. Für Gebäude können dementsprechend Verkabelungspläne abgelegt werden.

Sachmittel werden organisatorischen Einheiten oder einzelnen Aufgaben zugeordnet. Im Hinblick auf eine betriebswirtschaftliche Analyse der Sachmittelverwendung ist es sinnvoll, Informationen über die Kostenrechnung zuzuordnen. Dazu gehören einerseits Angaben über Fixkosten und variable Kosten, andererseits Angaben darüber, welchen Kostenstellen diese Kosten nach welchen Kriterien zugewiesen werden.

3.3 Kommunikationsverzeichnis

Kooperation impliziert Kommunikation - zwischen den Mitarbeitern eines Unternehmens und mit externen Partnern. Um eine zweckgerichtete Kommunikation zu unterstützen, sind zunächst drei Anforderungen zu erfüllen:

- Identifikation des Kommunikationspartners
- Auswahl des jeweils geeigneten Mediums
- Adressierung des Kommunikationspartners

Der in X.500 festgeschriebene Directory-Service der ISO fokussiert vor allem die Identifikation und die Adressierung. Dazu werden Objektklassen eingeführt, die einerseits die Suche nach bestimmten Personen unterstützen, andererseits Bestandteile der Adresse markieren. Sie basieren auf einem rudimentären Modell

von Organisationsstrukturen (vgl. Abb. 40).

Die Klassen werden durch Attribute spezifiziert. Für die Klasse "Organization" sind dies unter anderem der Name, die postalische und die Telekommunikationsadresse sowie die Branche. Die Suche nach Kommunikationspartnern kann durch ein Retrieval über die Eigenschaften der Objekte im Verzeichnis unterstützt werden. Die Adressierung erfolgt dann ähnlich wie in X.400 hierarchisch, indem zunächst eine Organisation, dann eine Abteilung, eine Rolle und schließlich ein Mitarbeiter spezifiziert werden.

Realweltliches Objekt	X.500-Klasse
Unternehmen	Organization
Standort	Locality
Abteilung	Organizational Unit
Projekt	Project
Rolle	Organizational Role
Mitarbeiter	Organizational Person

Abb. 40: X.500-Klassen zur Identifikation und Adressierung von Kommunikationspartnern- in Anlehnung an Prinz (1990, S. 16 ff.)

Auch wenn die Berücksichtigung von Standards empfehlenswert ist, ist es dennoch unbefriedigend, sich mit X.500 zu begnügen.[1] So erfolgt die Modellierung allenfalls in rudimentärer Objektorientierung: Klassen können lediglich durch Attribute[2], nicht durch Dienste definiert werden. Zur Kennzeichnung der Attribut-Semantik steht lediglich eine kleine Menge von Typen bereit.[3] Die Kardinalität der Attribute kann nicht in min, max-Notation ausgedrückt werden. Es kann lediglich zwischen "single value" und "multi value" sowie zwischen "optional" und "mandatory" unterschieden werden. Die Nachteile, die mit der fehlenden Verfügbarkeit von Diensten verbunden sind, werden an einigen Stellen offenkundig: So kann zwar durch ein bestimmtes Attribut angegeben werden, welches der einem Mitarbeiter zugeordneten Kommunikationsmedien dieser bevorzugt. Es ist

1. Diese Feststellung gilt ungeachtet des Umstands, daß X.500 nicht nur die Idenfikation von Organisationseinheiten und Personen unterstützt. Das Verzeichnis beinhaltet darüber hinaus auch Klassen zur Modellierung (und damit Identifikation) von Anwendungen und peripheren Geräten.
2. Genauer handelt es sich dabei um sogenannte "attribute sets" wie etwa dem "postalAttributeSet", das die Attribute zur Beschreibung der postalischen Adresse enthält. Vgl. ISO (1991).
3. Vgl. ISO (1991, S. 50 ff.). Dabei handelt es sich im wesentlichen um Typen für Zeit und Zeichenkette. Für die Typen können zudem Regeln für den Vergleich und das Sortieren der Attribut-Werte angegeben werden.

allerdings auf diese Weise kaum möglich, diese Präferenz in Abhängigkeit von bestimmten Bedingungen, wie etwa Wochentag und Tageszeit, auszudrücken. Auch können Beziehungen zwischen Objekten nicht eigenständig modelliert werden, sondern nur indirekt über Attribute. Es bleibt damit als einziges (gewiß nicht hinreichendes) Merkmal der Objektorientierung der Umstand, daß zwischen Klassen Generalisierungsbeziehungen bestehen können.

Jenseits dieser softwaretechnischen Unzulänglichkeiten ist die in X.500 vorgesehene Beschreibung möglicher Kommunikationspartner dürftig. So gibt es beispielsweise weder für die Klasse "Organizational Person" noch für die Klasse "Organizational Role" Attribute, die Auskunft über den Beruf oder sonstige Qualifikationen geben könnten. Schwerer als das Fehlen von Attributen wiegt jedoch der Umstand, daß Beziehungen zwischen Objekten nicht im wünschenswerten Umfang berücksichtigt werden. Das gilt einerseits für die in X.500 verzeichneten Objekte selbst: Es wäre beispielsweise hilfreich, wenn einem Mitarbeiter ein Vorgesetzter oder ein Stellvertreter zugeordnet werden könnte. Im Hinblick auf die Unternehmensmodellierung gilt dies für alle Objekte, zu denen ein Kommunikationspartner eine Beziehung haben kann. Auf diese Weise wird es möglich, die Suche nach Kommunikationspartnern in großem Umfang zu unterstützen: Jedes Objekt, zu dem das den Kommunikationspartner repräsentierende Objekt eine Beziehung unterhält, kann als Assoziation in die Suche einfließen. In diesem Zusammenhang ist auch an die Verknüpfung mit den Klassen im Teilmodell der Systemverwaltung zu denken. Dort ist ja vorgesehen, einzelnen DV-Arbeitsplätzen die jeweils verfügbaren Klassen zuzuordnen. Auf diese Weise wird es möglich, neben dem Medium auch die am besten geeignete Klasse für das Versenden von Objekten zu finden.

Die Suche nach einem Kommunikationspartner wird mitunter durch bestimmte Eigenschaften geleitet, die zwar in einem entsprechenden Objektmodell berücksichtigt sein mögen, für die es aber keine einheitlichen Bezeichnungen gibt. Hier ist beispielsweise an die Bezeichnung von Branchen oder die von Mitarbeiter-Qualifikationen zu denken. Eine Maßnahme zur Bewältigung dieser terminologischen Varianz ist die Einführung von Thesauri.

Im Hinblick auf die Auswahl von Medien sowie die betriebswirtschaftliche Analyse von Kommunikationsbeziehungen kann es sinnvoll sein, den jeweils verfügbaren Medien Nutzungskosten zuzuordnen.

Wenn man, aus den oben angeführten Gründen, ein über X.500 deutlich hinausgehendes Objektmodell verwendet, ist damit nicht notwendig der Verzicht auf die Vorteile diese Standards verbunden. Vielmehr können entsprechende Transformationen vorgesehen werden, die zwar unter Umständen mit einem Verlust an Semantik verbunden sind, aber einen X.500-konformen Nachrichtenaustausch ermöglichen.

3.4 Informationsmanagement

Bei der Arbeit im Büro- und Verwaltungsbereich spielt Information eine zentrale Rolle. Das gilt einerseits für Informationen, deren Semantik anwendungsnah (objektorientiert) konzeptualisiert ist. Daneben gibt es allerdings noch andere Informationen, die ebenfalls zu berücksichtigen sind. Dazu gehören einerseits solche Infomationen, die zwar auf Datenträgern abliegen, deren Anwendungssemantik dabei aber nicht berücksichtigt wird. Ein gängiges Beispiel dafür sind Texte oder Grafiken, die lediglich als Dateien gespeichert werden. Andererseits ist hier an jene Informationen zu denken, die nicht in maschinenlesbarer Form abliegen. Schließlich können auch solche Informationen berücksichtigt werden, die nicht im Unternehmen selbst vorhanden sind - wie etwa Literatur in externen Bibliotheken oder Texte auf externen Datenbanken. Um die gemeinsame Verwaltung all dieser Informationen und ihre zentrale Bedeutung für die Unternehmensführung zu betonen, wird seit einigen Jahren der Begriff Informationsmanagement[1] verwendet. Für die damit verbundene Sicht ist es wesentlich, auf den Inhalt, den Verwendungszweck und allgemein auf die wirtschaftliche Bedeutung von Informationen zu fokussieren.

Die Modellierung dieses Teilbereichs sollte folgende Aufgaben unterstützen:

- die Suche nach Informationen
- die sichere Archivierung von Informationen
- die Bereitstellung von Informationen
- die Wiederverwendung von Informationen
- eine den Unternehmenszielen angemessene und gegebenenfalls einheitliche Präsentation bestimmter Informationen gegenüber externen Partnern
- die betriebswirtschaftliche Beurteilung von Informationen und ihrer Verwendung

Ein wichtiges Hilfsmittel zur Unterstützung der Suche bildet die Differenzierung in verschiedene Arten von Informationsobjekten. Da die gesuchten Objekte immer in irgendeiner Form zu präsentieren sind, können wir ihre Darstellung als Dokument bezeichnen. Eine solch weite Begriffsfassung führt zu einer entsprechend großen Vielzahl von Dokumentarten. Um einige zunächst nicht systematisierte Beispiele zu nennen: Textdateien, Grafiken (als Pixeldateien oder mit Hilfe einer Beschreibungssprache definiert), Bild- und Tondokumente, Verträge, Rechnungen, Compound Documents (also Dokumente, die ein logische und eine Layout-Struktur haben), Geschäftsbriefe, Akten, Prospekte, Zeitschriften. Die Konzeptualisierung dieser Dokumente erfolgt in unserem Modell mit Hilfe von Klassen, die nach Maßgabe des oben dargestellten Metamodells Eigenschaften der

1. Vgl. etwa Heinrich/Burgholzer (1987), Wollnik (1988).

verschiedenen Dokumentarten beschreiben. Dazu gehören Angaben über die Oberklasse, das Erstellungsdatum, das Medium, sowie möglicherweise über die logische und die Layout-Struktur. Darüber hinaus können Schlagworte zugeordnet werden.

Die oben angeführten Beispiele machen deutlich, daß bei den Dokumentklassen zwischen Dokumenten, die vollständig in maschinenlesbarer Form abgelegt sind, und Referenzdokumenten zu unterscheiden ist. Referenzdokumente sind Objekte, die lediglich Angaben über die Dokumente enthalten, auf die sie hinweisen.[1] Diese Angaben können dann für Recherchezwecke genutzt werden. Im Falle maschinenlesbarer Dokumente, die sich also vollständig auf Datenträgern befinden, gilt dies darüber hinaus für den gesamten Objektzustand (beispielsweise ein Volltext). Von Referenzdokumenten in der hier verwendeten Bedeutung sind elektronische Dokumente zu unterscheiden, die andere Dokumente oder allgemeiner: Objekte beinhalten. Auch wenn es sich dabei in der technischen Realisation nur um eine Referenz handeln mag, wird diese jedoch beim Betrachten des Dokuments in einer für den Betrachter transparenten Weise befriedigt.

Im Hinblick auf die Archivierung können neben dem Medium und dem Ort Angaben über Kosten und Maßnahmen zur Sicherung und zum Schutz der Informationen erfaßt werden. Daneben kann ein verantwortlicher Mitarbeiter zugewiesen werden. Zudem können Konventionen über die Kennzeichnung oder Benennung von Dokumenten und Dokumentversionen festgelegt werden.

Eine besondere Rolle kommt der Modellierung des Berichtswesens zu. Sie werden beschrieben durch ihren Inhalt, wobei vor allem an Verweise auf Objekte (und deren Dienste) zu denken ist. Daneben sind Angaben über die Präsentation eines Berichts, die Anlässe für seine Erstellung sowie den Verteiler vorzusehen. Im Hinblick auf die Implementierung ist es dabei wichtig, zu beurteilen, ob all diese Kriterien formalisiert werden können, so daß gegebenenfalls eine automatische Berichterstellung möglich ist.

Um dem Interessenten den Zugang zu erleichtern, können bei Bedarf Angaben über die Form der Bereitstellung (Einsichtnahme im Archiv, Anfordern einer Kopie bei einem bestimmten Mitarbeiter oder direkter Zugriff mit Hilfe des Rechners) abgelegt werden. Daneben können Verteiler beschrieben werden, also eine - unter Umständen geordnete - Menge von Benutzern, die beim Erscheinen eines neuen Exemplars einer bestimmten Art ein solches erhalten.

Die Forderungen nach Wiederverwendung und unternehmensweit einheitlichen Darstellungsformen sind komplementär. Sie können durch die Vorgabe von

1. Dabei sind auch Referenzen auf Referenzdokumente denkbar: Wenn beispielsweise ein Referenzdokument auf eine externe Referenzdatenbank zeigt.

Gestaltungsrichtlinien oder entsprechend vorgefertigter, wiederverwendbarer Muster erfüllt werden. Dabei ist an Firmenlogos, das Layout von Briefen, die Bereitstellung grafischen Darstellungen oder Fotos sowie an die Vorgabe klassischer Dokumentinhalte, wie etwa Textbausteinen, zu denken. Diese Informationen sollten den möglichen Verwendern in komfortabler Weise verfügbar gemacht werden. Der Einsatz einer Compound Document Architecture ist dazu besonders hilfreich. So können einerseits Dokument-Klassen definiert werden, die durch eine bestimmte logische und Layout-Struktur gekennzeichnet sind. Außerdem können Texte, Grafiken und dergleichen in Form instanzierter Dokumentbestandteile (Kapitel, Abschnitt, Abbildung etc.) abgelegt werden.

Um eine Grundlage für ökomomische Bewertungen zu liefern, sind neben den bereits erwähnten Kosten für die Archivierung andere Kostenarten zu berücksichtigen. Dazu zählen unter anderem Kosten für den Erwerb oder die Nutzung (etwa bei externen Datenbanken) und für die interne Informationsaufbereitung. Daneben ist der Nutzungskontext zu modellieren. Dazu können einem Informationsobjekt Benutzer zugeordnet werden. Ergiebiger ist die Zuordnung von Verwendungszwecken, also beispielsweise bestimmten Aufgaben oder Vorgängen, da sich auf diese Weise eher Schlußfolgerungen über den Nutzen der Information ziehen lassen (vgl. dazu V.3.1).

Um die dargestellten Anforderung an das Informationsmanagement zu erfüllen, ist es zudem wünschenswert, einen Thesaurus, gleichsam als dediziertes Verwaltungsobjekt, einzuführen. Er leitet einerseits eine einheitliche Beschlagwortung an, andererseits unterstützt er durch die Bereitstellung von Assoziationen zwischen Begriffen die Suche nach Informationen. Potentieller Gegenstand des Thesaurus sind alle Begriffe, die im Organisationsmodell - auf konzeptueller wie auf Instanzenebene - verwendet werden. Im Hinblick auf die Integration der drei hier vorgeschlagenen Hauptperspektiven ist es zudem wünschenswert, die Begriffe der strategischen und der Informationssystem-Perspektive mit einzuflechten.

3.5 Ablauforganisation

Für das Verständnis des Unternehmensgeschehens wie auch im Hinblick auf effizienzsteigernde organisatorische Maßnahmen ist die Untersuchung von Geschäftsprozessen von zentraler Bedeutung. Dieser Umstand artikuliert sich auch darin, daß in den frühen Phasen traditioneller Systemanalyse häufig eine Betrachtung funktionaler Zusammenhänge im Vordergrund steht. In neuerer Zeit ist die Fokussierung auf Geschäftsprozesse mit dem Anliegen verbunden, Ansatzpunkte für eine Neugestaltung der Ablauforganisation zu entdecken, die gewandelten Marktgegebenheiten Rechnung tragen. In diesem Zusammenhang ist an Schlagworte wie "Responsiveness" oder "lean organization", aber auch "outsourcing" und "management of the interdependencies" zu denken.[1] Der Ein-

satz von Informationstechnik spielt dabei in zweifacher Hinsicht eine wichtige Rolle. Einerseits kann er für die Auswertung der in einer Organisationanalyse erhobenen Daten sowie für den Entwurf und die Simulation von Gestaltungsalternativen eingesetzt werden.[1] Andererseits gibt es gute Gründe dafür, daß die Leistungspotentiale von Informationstechnik durch die "Veränderung von Arbeitsabläufen und Bearbeitungsweisen erheblich erweitert"[2] werden.

Im Unterschied zu der Betrachtung von Vorgängen im Informationssystem-Modell geht es an dieser Stelle nicht nur um automatisierte Abläufe. Vielmehr sollen auch solche Geschäftsprozesse im Büro- und Verwaltungsbereich berücksichtigt werden, die nicht oder nur teilweise automatisiert sind. Die Modellierung soll einerseits eine anschauliche Dokumentation der Ablauforganisation unterstützen, andererseits soll sie die Grundlage für Organisationsanalysen und daran anknüpfende organisatorische Veränderungen bieten.

Dazu ist zunächst ein Verzeichnis aller hinreichend bedeutsamen Arten von Aufgabenerfüllungsprozessen[3] zu erstellen - zusammen mit den Ereignissen[4], die jeweils einen solchen Prozeß auslösen. Zur Modellierung einer bestimmten Prozeßart verwenden wir einen ähnlichen Ansatz wie in 4.2.1.2: Die Gesamtaufgabe wird durch eine zeitlich gerichtete Menge von Teilaufgaben repräsentiert. Des breiteren Blickwinkels wegen sind die Teilaufgaben und die sie verbindenden Zustände in anderer Weise zu beschreiben. Anders als bei der Konzeptualisierung von Vorgängen in der Informationssystem-Perspektive können die auslösenden oder zu produzierenden Zustände nicht immer als eine Kollektion von Objekten, deren Semantik in einem Objektmodell definiert ist, beschrieben werden. Vielmehr können auch papiergestützte Dokumente und deren Zustände eine Rolle spielen. Ihre Beschreibung kann unter Verweis auf die in der Teilperspektive des Informationsmanagements modellierten Dokumentarten erfolgen. Zur Modellierung der Teilaufgaben unterscheiden wir im folgenden drei Kategorien.

Aufbauorganisatorischer Rahmen

Ein Aufgabenerfüllungsprozeß vollzieht sich innerhalb einer oder mehrerer organisatorischer Einheiten. Es können mehrere Mitarbeiter in unterschiedlichen Rol-

1. Vgl. Rockart/Short (1991), Venkatraman/Short (1990), Szyperski (1990).

1. Vgl. dazu die Beiträge in Schönecker/Nippa (1987), sowie Götzer (1990), Elgass/Krcmar (1993) und Katz (1990), der dazu explizit die Bedeutung von Unternehmensmodellen unterstreicht.

2. Baethge/Overbeck (1986, S. 23). Vgl. dazu auch Frank/Klein (1992 a, S. 26 ff.) und Savage 1990.

3. Dieser Begriff wird hier verwendet, um das breite Kontinuum zwischen einfachen Aufgaben und komplexen Vorgängen abzudecken.

4. In Sutherland (1983, S. 16 f.) findet sich eine Taxanomie von Ereignissen im Rahmen von Bürovorgängen.

len daran beteiligt sein. Die Differenzierung in Teilaufgaben sollte möglichst so granuliert sein, daß eine Teilaufgabe einer Organisationseinheit und einer Stelle zugeordnet werden kann. Außerdem können einer Teilaufgabe Ressourcen im oben gefaßten Sinn zugeordnet werden - also beispielsweise Büromaschinen, Telekommunikationsgeräte und - unter Verweis auf die in der Informationssystem-Perspektive beschriebenen Konzepte - Informationstechnik. Für den gesamten Aufgabenerfüllungsprozeß sowie gegebenenfalls für jede Teilaufgabe kann angegeben werden, welche Organisationseinheit oder welche Rolle verantwortlich ist. Im Hinblick auf die mögliche Auslagerung (outsourcing) einer Teilaufgabe kann gegebenenfalls angegeben werden, wie sehr die Ausführung an das Unternehmen gebunden ist. Dabei ist allerdings auch an Kriterien zu denken, die sich nicht eben für eine Formalisierung anbieten, wie etwa unternehmensspezifische Qualifikation, Vertrauensschutz und dergleichen.

Information und Kommunikation

In einer Teilaufgabe wird ein bestimmter Informationsbestand übernommen. Zur weiteren Bearbeitung mag der Rückgriff auf zusätzliche Informationen nötig sein. Sie können in verschiedener Form abliegen: als Objekte im (computerbasierten) Informationssystem, als (Papier-) Akten, in Fachzeitschriften etc. Die daran jeweils geknüpften Zugriffswege, die Medien, die Verfügbarkeit und die Kosten sind im Teilmodell "Informationsmanagement" abgebildet. Die Modellierung kann dabei in Abhängigkeit von jeweils unterstellten Kosten/Nutzen-Relationen unterschiedlich detailliert erfolgen. So kann beispielsweise nur die Klasse eines benötigten Objekts (etwa "Kunde") angegeben werden oder aber die konkret benötigten Dienste (wie "umsatz", "name" und dergleichen). Ähnliches gilt für Akten oder Fachzeitschriften: Es können entweder nur die Akte oder die Zeitschrift oder aber bestimmte Abschnitte oder Rubriken genannt werden.

Neben den irgendwie dokumentierten Informationen erfordert die Bearbeitung einer Teilaufgabe unter Umständen die Kommunikation mit internen wie auch externen Partnern. Eine mögliche Kommunikation sollte durch ihren Anlaß ("tritt immer auf" oder "nur, wenn ....") gekennzeichnet werden. Zudem ist - unter Verweis auf die im Teilmodell "Kommunikationsverzeichnis" beschriebenen Rollen - anzugeben, um welchen Kommunikationspartner es sich jeweils handelt. Außerdem können hier der Gegenstand der Kommunikation (beispielsweise der zu klärende Sachverhalt oder die zu beschaffende Information) sowie die üblicherweise verwendeten Medien (Telefon, Fax etc.) zugeordnet werden.

Durchführung

Die Dynamik eines Aufgabenerfüllungsprozesses wird durch den Graphen der zeitlich geordneten Teilaufgaben beschrieben. Die Durchführung der Teilaufgaben selbst kann einerseits - ähnlich wie bei der in IV.2.1.2 dargestellten Definition automatisierter Vorgänge - mit Hilfe von Vor- und Nachbedingungen (die an

dieser Stelle nicht formalisiert sein müssen) beschrieben werden. Andererseits können Rahmenbedingungen für Entscheidungen vorgegeben werden. Dabei kann es sich um inhaltliche Entscheidungsregeln oder -orientierungen aber auch um Richtlinien für die Durchführung von Entscheidungsprozessen handeln (etwa "Abstimmung mit Vorgesetztem"). Darüber hinaus können jeder Teilaufgabe eine geschätzte und eine maximale Bearbeitungszeit zugeordnet werden. Schließlich können generelle Randbedingungen für den gesamten Aufgabenerfüllungsprozeß formuliert werden. Beispiel: Sämtliche Teilaufgaben eines konkreten Prozesses sind von genau einem verantwortlichen Mitarbeiter durchzuführen.

Im Hinblick auf die betriebswirtschaftliche Bewertung von Prozessen ist es zudem sinnvoll, die in einer Teilaufgabe benötigten bedeutsamen Ressourcen zuzuordnen. Die Beschreibung der Durchführung kann ergänzt werden durch ein Verzeichnis möglicher Störfälle oder Ausnahmen und der daraufhin zu ergreifenden Maßnahmen. Zur Veranschaulichung der intendierten Abwicklung einer Aufgabe können zur Dokumentation Beispiele vorbildlich durchgeführter Prozesse angehängt werden. Neben Handbüchern ist dabei an Videodokumentationen zu denken, die im Rahmen einer computergestützten Modellverwaltung mit Hilfe von Hypermedia-Ansätzen kombiniert werden mögen.

Zur Darstellung von Aufgabenerfüllungsprozessen werden in Organisationshandbüchern oder ähnlichen Dokumentationen häufig sogenannte Aufgaben-Folgepläne (Liebelt 1992) verwendet. Solche Grafiken fokussieren - ähnlich wie Programmablaufpläne - auf die Logik eines Prozesses. Sie erlauben es allerdings nicht, Nebenläufigkeit zwischen den Teilaufgaben auszudrücken. Deshalb ist - nicht zuletzt im Hinblick auf Analyse und Simulation - die bereits erwähnte Repräsentation als gerichteter Graph angemessener. Aufgaben-Folgepläne können allerdings verwendet werden, um die Abwicklung einer Teilaufgabe zu beschreiben. Abbildung 41 zeigt die Konzepte, die zur Beschreibung eines Aufgabenerfüllungsprozesse im Rahmen der organisatorischen Perspektive verwendet werden können.

Analysepotentiale

In die skizzierte Modellierung von Aufgabenerfüllungsprozessen fließen Konzepte ein, wie sie üblicherweise in dedizierten Informationsbedarfs- und Kommunikationsanalyseverfahren Verwendung finden. Im Unterschied zu solchen Verfahren, die zu einzelnen Zeitpunkten mit aufwendigen Erhebungen einhergehen, bietet die Integration dieser Sachverhalte in ein Unternehmensmodell die Chance, Organisationsanalysen zu einem permanenten Prozeß zu machen.[1]

1. Dabei kann natürlich nicht übersehen werden, daß für den Entwurf des Modells selbst entsprechende Erhebungen unerläßlich sind.

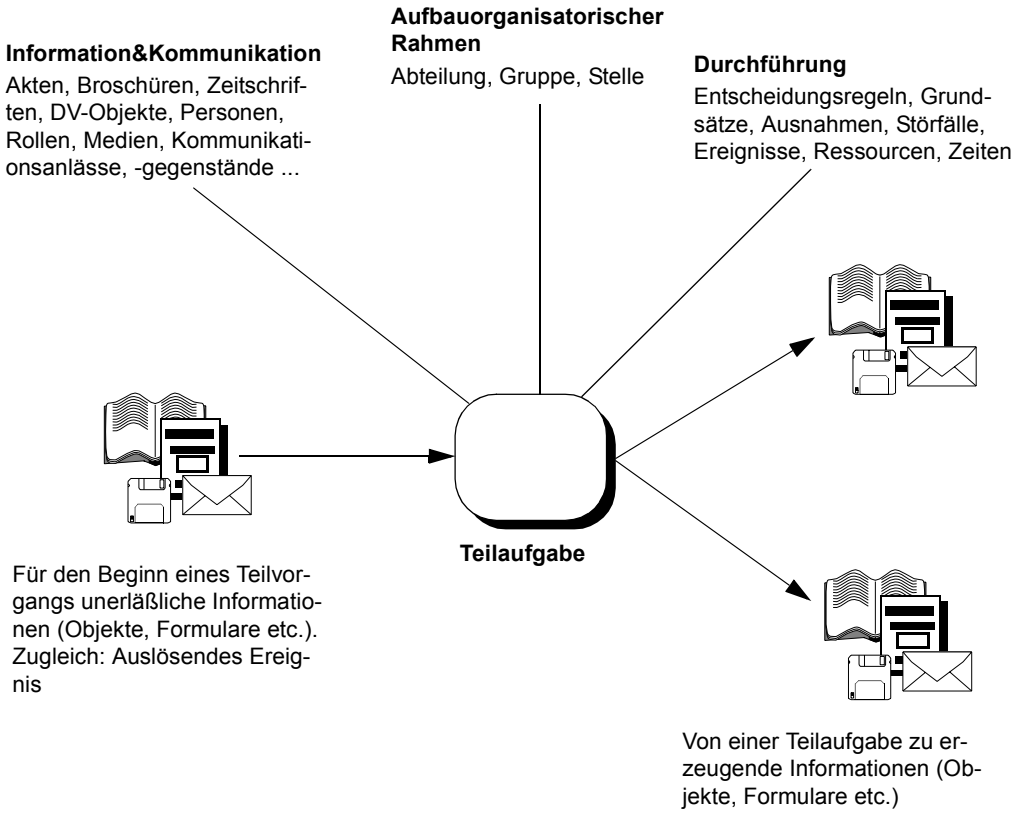


Abb. 41: Konzepte zur Beschreibung von Aufgabenerfüllungsprozessen in der organisatorischen Perspektive.

Die Analyse kann einerseits an dem (instanzierten) Modell der Aufgabenerfüllungsprozesse ansetzen, andererseits unter Rückgriff auf Simulationen mit Hilfe variender Instanzierungen des Modells erfolgen.

Je nach Detaillierungsgrad des Modells können mehr oder weniger vordefinierte Schwachstellen untersucht werden. Dazu gehören beispielsweise die Ermittlung von Teilaufgaben, die Flaschenhälse darstellen oder die Feststellung von Medienbrüchen. Daneben können potentiell problematische Zusammenhänge dargestellt werden, die dann durch einen sachkundigen Betrachter zu interpretieren sind - wie etwa Kommunikationsdiagramme oder die Auflistung von Aufgabenerfüllungsprozessen, in denen keine Informationstechnik eingesetzt wird.

Simulationen können beispielsweise Aufschluß darüber geben, welche Auswirkungen der Einsatz eines weiteren Mitarbeiters zur Bewältigung einer bestimmten Teilaufgabe auf den gesamten Durchsatz eines Aufgabenerfüllungskomplexes hat - oder welche Kosten und welcher Zeitgewinn der Einsatz von elektroni-

scher Post im Unternehmen erwarten läßt. Auch die entsprechenden Auswirkungen einer Auslagerung einzelner Aktivitäten könnten simuliert werden, wenn die dazu nötigen Sachverhalte im Modell berücksichtigt sind. Da Aufgabenerfüllungsprozesse in Organisationen häufig interdependent sind (indem sie die gleichen Ressourcen beanspruchen oder gegenseitig Vorleistungen erbringen), kann die Komplexität solcher Simulationen beträchtlich sein: Im Grenzfall umfaßt sie alle Handlungskomplexe eines Unternehmens. Die Erfahrungen mit Simulationsmodellen in der Betriebswirtschaftslehre einerseits, das Wissen um die (soziale) Kontingenz des Organisierens andererseits verbieten dabei die Vorstellung einer durch Simulationen realisierten optimalen Organisation. Simulationen, die auf einem umfassenden Unternehmensmodell basieren, bieten allerdings die Chance, die für die organisatorische Gestaltung bedeutsamen Zusammenhänge deutlicher werden zu lassen und Hinweise für konkrete Maßnahmen zu erhalten.

3.6 Modellierung und Integration der Teilaspekte

Für die Modellierung der organisatorischen Perspektive wählen wir einen objektorientierten Ansatz nach Maßgabe des oben entwickelten Metamodells. Dabei ist folgende Besonderheit zu beachten: Es werden auch Beziehungen zwischen Objekten der Realwelt beschrieben, die nicht gleichzeitig Beziehungen zwischen den Objekten des Modells sind, die also vom Informationssystem nicht verwaltet werden. So kann etwa eine Kommunikationsbeziehung zwischen einem Sachbearbeiter und einem Kunden modelliert werden. Wenn diese sich beispielsweise über (konventionelles) Telefonieren vollzieht, liegt sie außerhalb der Kontrollmöglichkeiten des Informationssystems. Daraus folgt, daß diese Beziehung im Objektmodell nicht als Beziehung beschrieben werden kann. Eine ähnliche Divergenz zwischen Realwelt und Modell ergibt sich bei der Festlegung von Informationsarten, deren Zugriff nicht rechnergestützt erfolgt: Die entsprechende Informationsart kann zwar im Modell angegeben werden, es handelt sich allerdings nicht um eine Typisierung, also die Einführung von Konsistenzbedingungen, die vom System selbst überwacht werden können.

Die folgenden Schaubilder zeigen einen Ausschnitt des vorgeschlagenen Modells zur Abbildung der organisatorischen Perspektive. Neben Teilen der Vererbungshierarchie werden dazu ausgesuchte Beziehungen zwischen Objekten dargestellt. Dabei ist zu berücksichtigen, daß die Konzepte der organisatorischen Perspektive durch eine erhebliche Heterogenität gekennzeichnet sind. Die vorgeschlagene Vererbungshierarchie ist deshalb flach.[1] Darüber hinaus ist daran zu denken, daß die Bedeutung einiger Konzepte allenfalls rudimentär formalisiert werden kann. Viemehr wird sich - für den Betrachter des Modells - die Semantik häufig vor allem in geeigneten Annotationen (beispielsweise in Form von multimedialen Darstellungen) widerspiegeln. Wie bereits in IV.2.2.3 sind abstrakte Klassen

durch Kursivdruck gekennzeichnet.

Die in Abbildung 45 dargestellten Beziehungen beschränken sich auf Objekte der organisatorischen Perspektive. Das gilt nicht zuletzt auch für die Beziehungen zu einem auf Datenträger erfaßten Geschäftsbericht. Um den Inhalt eines solchen Berichts präziser zu modellieren, bietet es sich an, Beziehungen zu den jeweils benutzten Objekten des Informationssystems (wie etwa bestimmte betriebswirtschaftliche Kennzahlen oder Statistiken) darzustellen. Solche Beziehungen sind nur dann in ein Objektmodell aufzunehmen, wenn das Objekt der Klasse "E_Geschäftsbericht" eine Referenz auf das andere Objekt benötigt. Wenn also beispielsweise der Hinweis auf den Verfasser manuell eingegeben wird, handelt es sich nicht um eine Beziehung. Wenn in diesem Fall die Angabe des Verfassers obligatorisch ist, könnte diese Integritätsbedingung dadurch gekennzeichnet werden, daß ein entsprechendes Attribut mit der Kardinalität (1,*) eingeführt wird. Im Falle einer Beziehung könnte noch - durch die Einführung geeigneter Bezeichnungen - zwischen Referenz- und Wertsemantik unterschieden werden, je nachdem, ob das Dokument im Zeitverlauf immer die aktuellen Zustände eines assoziierten Objekts wiedergegeben soll oder nicht.

1. Dabei gilt - wie grundsätzlich beim Entwurf von Objektmodellen, daß der Umfang des gewählten Spezialisierungsniveaus vom intendierten Verwendungszweck abhängt. So ist es gewiß nicht - wie im folgenden vorgeschlagen - notwendig, die Klasse "Büromaschine" in die Unterklassen "Kopiergerät", "Schreibmaschine" etc. zu spezialisieren. Vielmehr könnte man diese Differenzierung auch mit Hilfe von Attributen artikulieren - wäre dann aber nicht in der Lage, eine eindeutige Beziehung zwischen einem Arbeitsplatz und einem Kopiergerät zu definieren. Daneben ist zu berücksichtigen, daß die zu modellierenden Objekte der Realwelt im Zeitverlauf Änderungen unterliegen können. So kann eine Zeitschrift, die zunächst in Form eines Referenzdokuments verwaltet wird, in Zukunft vollständig in elektronischer Form vorliegen.

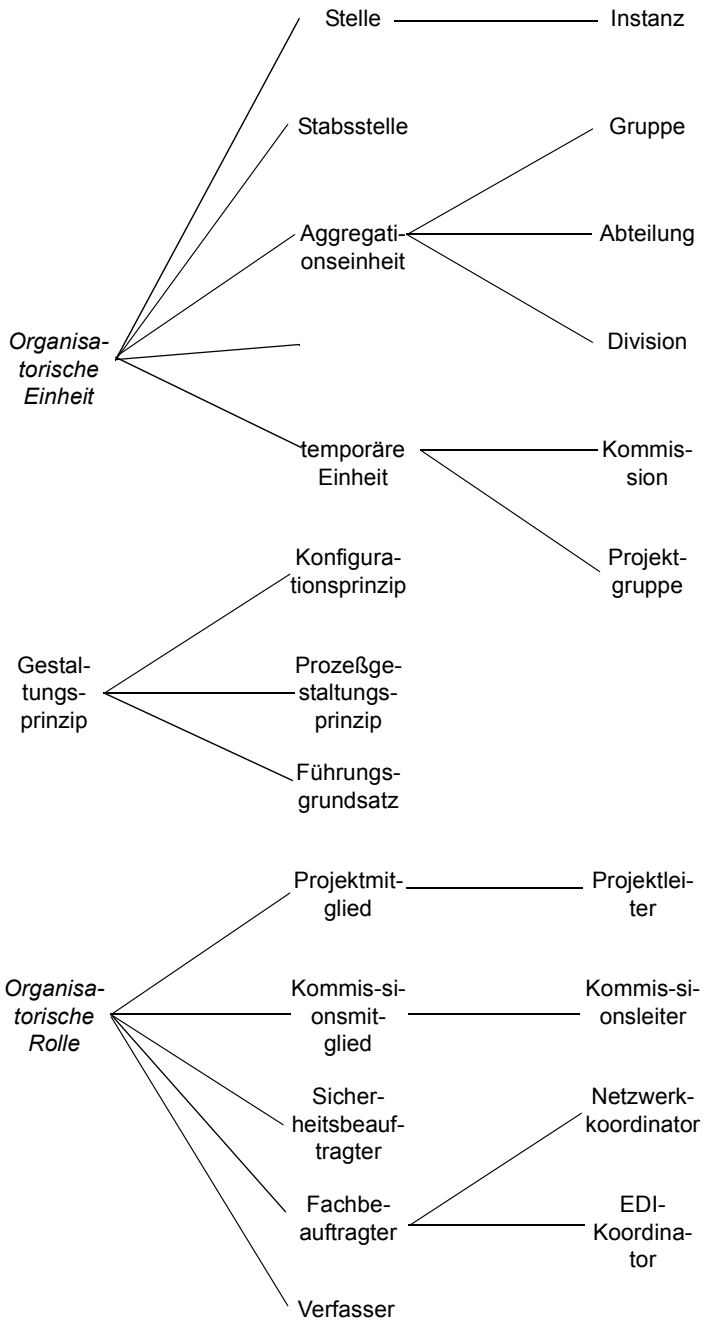


Abb. 42: Ausgewählte Generalisierungsbeziehungen (Organisationsstruktur)

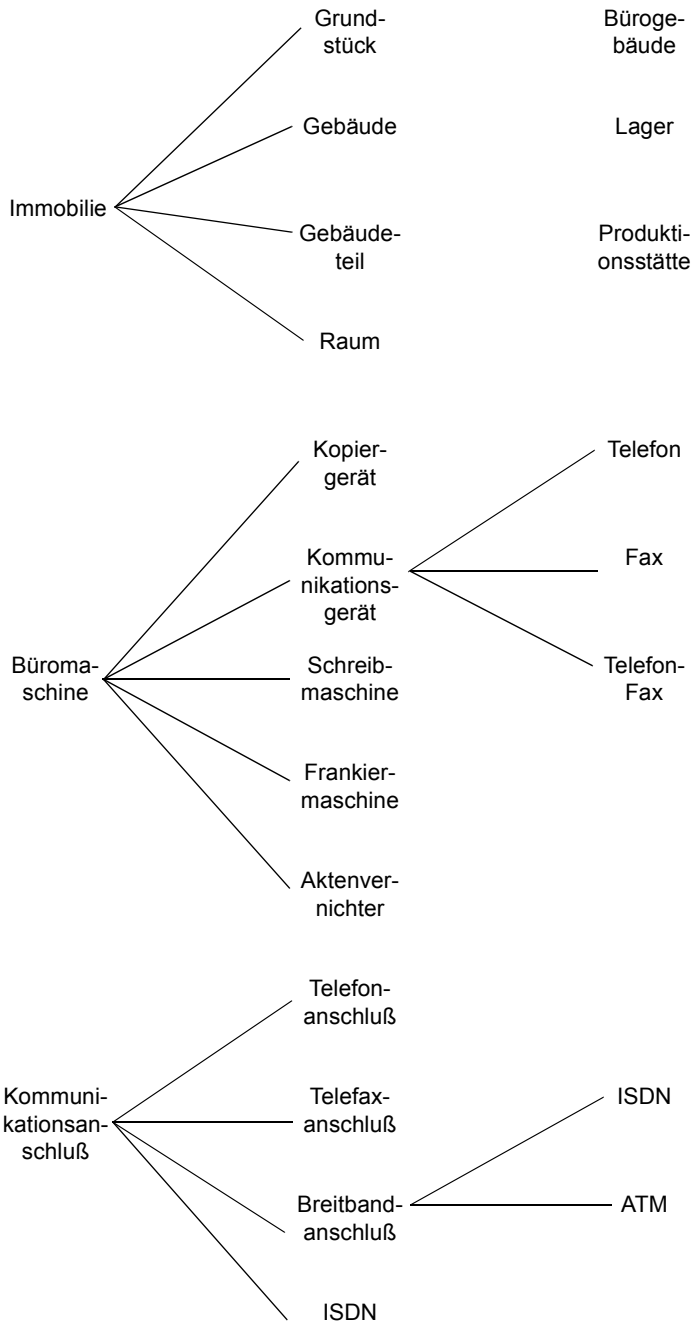


Abb. 43: Ausgewählte Generalisierungsbeziehungen (Ressourcen)

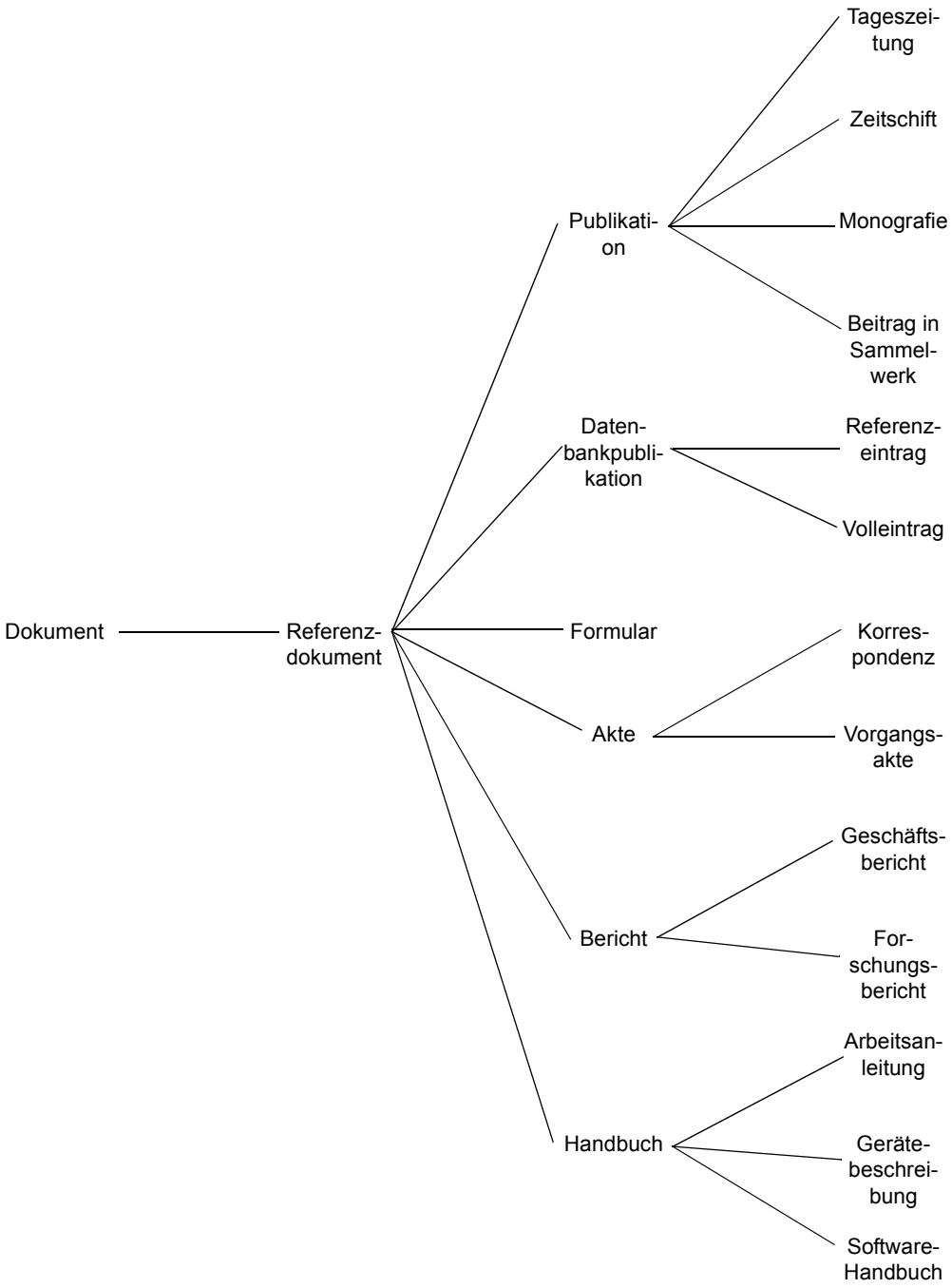


Abb. 44: Ausgewählte Generalisierungsbeziehungen (Informationsmanagement). Volldokumente können in ähnlicher Weise systematisiert werden.

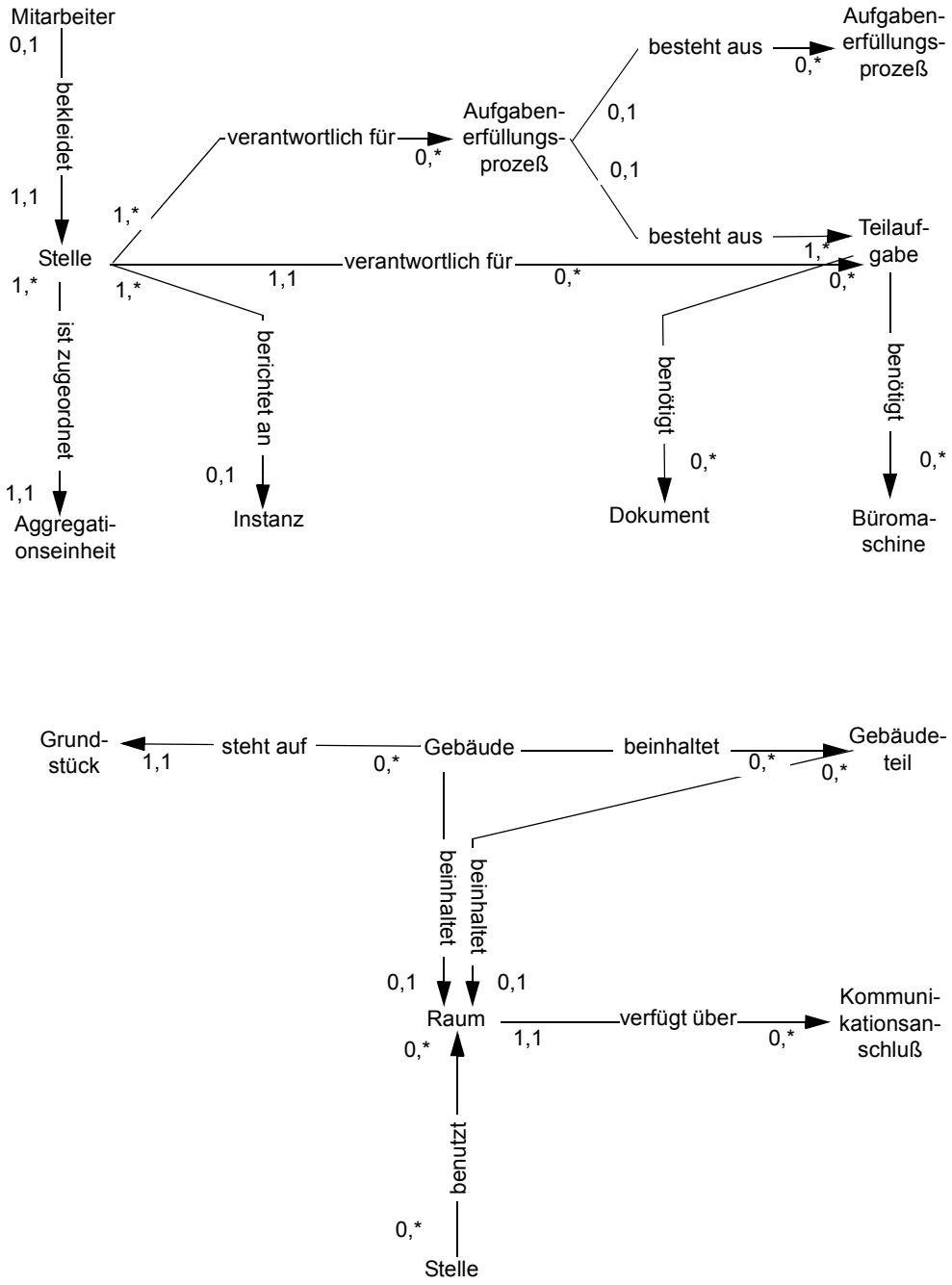
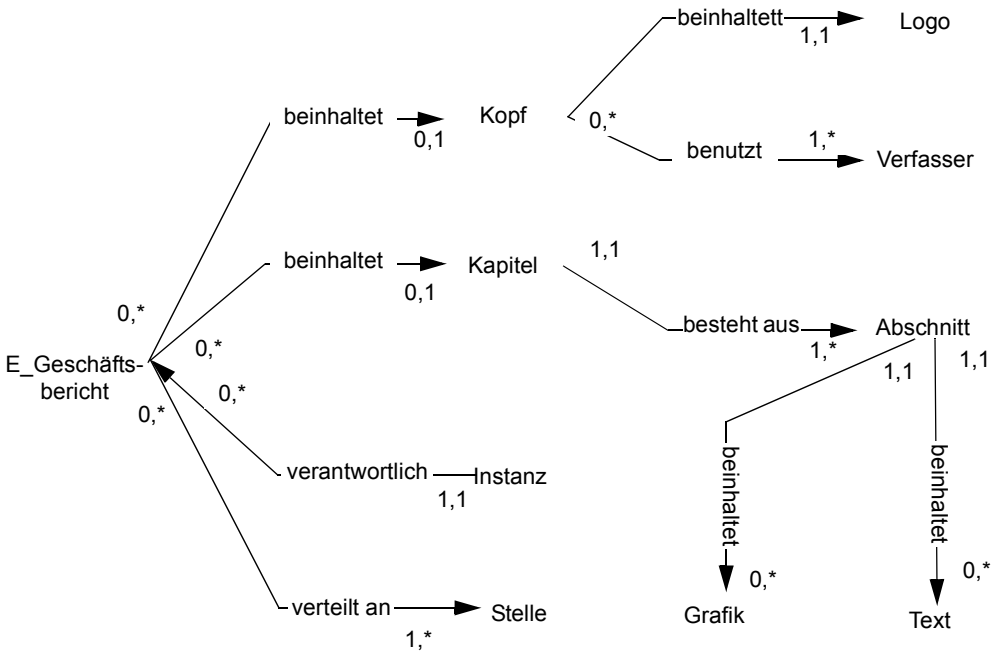


Abb. 45: Ausgewählte Beziehungen der organisatorischen Perspektive, Fortsetzung folgende Seite



Der Nutzen der Modellierung der organisatorischen Perspektive hängt wesentlich vom Umfang und Detaillierungsgrad des Modells sowie der Vollständigkeit und Aktualität der mit Hilfe des Modells verwalteten Instanzen ab. Es liegt auf der Hand, daß eine Verbesserung des Modells in diesem Sinne mit einem erheblichen Aufwand verbunden ist. Ein Umstand, der bereits für Organisationshandbücher hinreichend bekannt ist (Müller-Pleuß 1992, Sp. 1507):

> "Im Gegenteil muß festgestellt werden, daß die schriftliche Fixierung von Organisationshandbüchern bei der Mehrzahl der Unternehmen bisher an der Inhaltsfülle und dem Darstellungsaufwand gescheitert ist."

Dieser Aufwand warnt vor allzu großer Euphorie, diskreditiert den Einsatz eines Organisationsmodells aber nicht. So muß ein Modell nicht so komplex sein, wie es durch die hier skizzierten Ausschnitte angedeutet ist. Zudem kann der Aufwand für den Entwurf eines Modells vernachlässigt werden, wenn auf existierende Modelle zurückgegriffen werden kann. Der Umfang der Instanzierung solcher (wiederverwendbaren) Modelle kann dann von individuellen Kosten/Nutzen-Relationen abhängig gemacht werden.

4. Die strategische Perspektive

Es gibt bemerkenswerte empirische Belege dafür, daß die Führung von Unternehmen nicht immer auf der Grundlage einer reflektierten Planung von Zielen und daran anknüpfenden Strategien beruht. So zeichnen Cohen/March/Olsen (1976) in ihrem "garbage can model" ein Bild von (strategischen) Entscheidungsprozessen, in dem die Entscheidungsträger nicht eben durch das Bemühen um eine sorgfältige Analyse der je relevanten Zusammenhänge glänzen. Auch für Pfeffer (1978 u. 1981) ist es weniger der Entwurf konsistenter Zielsysteme, der Orientierungen für Unternehmen festlegt, als vielmehr eingefahrene Abläufe, denen durch "symbolisches Handeln" der Führungskräfte Sinn verliehen wird. Anders als in vielen Lehrbuchdefinitionen üblich, sieht Weick (1985) das Fehlen von Zielsystemen als geradezu charakteristische Eigenschaft von Organisationen an. Auch wenn dieser Umstand zum Teil als wirkungsvolle Adaption an hohe Komplexität betrachtet werden kann[1], stellt er doch für eine Betriebswirtschaftslehre, die das Handeln in Unternehmen mit Hilfe ökonomischer Prinzipien zu gestalten sucht, ein Defizit dar.

Die Durchdringung und Anleitung von Prozessen der strategischen Planung hat deshalb eine nahezu konstitutive Bedeutung für die Betriebswirtschaftslehre. Dabei steht traditionell eine technokratische Sicht im Vordergrund, die das Verhältnis von Zielen und Mitteln sowie die zwischen verschiedenen Zielen einerseits, zwischen Mitteln (die auf einer anderen Ebene selbst wieder als Ziele betrachtet werden können) andererseits bestehenden Beziehungen (wie konkurrierend oder komplementär) betont.[2] In jüngerer Zeit wird daneben verstärkt darauf hingewiesen, daß auch solche Orientierungen zu berücksichtigen seien, die vor allem darauf zielen, dem gemeinsamen Handeln Sinn zu verleihen. Hier ist an Begriffe wie "corporate identity" oder "corporate culture" zu denken.

Die mit der strategischen Planung verbundenen Herausforderungen und ihr interdependentes Verhältnis zur Realität macht Klein (1989, S. 4) mit Nachdruck deutlich:

> "Planung ist ein reflektierender Handlungsentwurf, der in einer sorgfältigen Beobachtung der Situation und einer dadurch sensibilisierten Antizipation zukünftiger Entwicklungen gründet. Grundproblem der Planung ist dabei die Bewältigung der Kontingenz der Bedingungen des Handelns sowie der doppelten Kontingenz wechselseitig aufein-

1. In diesem Sinn macht Lindblom (1964) mit der "science of muddling through" aus der Not eine Tugend.
2. Dabei soll an dieser Stelle nicht der naiven Vorstellung das Wort geredet werden, daß zunächst Ziele formuliert und dann - im Bemühen um Wirtschaftlichkeit - geeignete Mittel gewählt werden. Es ist seit langem bekannt, daß die Gestaltung von Zielen und Mitteln in einem interdependenten Verhältnis erfolgt. Ein Umstand, auf den nicht zuletzt in Werturteilsdebatten immer wieder hingewiesen wurde. Vgl. dazu Myrdal (1933), Albert (1960) oder Ortmann (1976).

ander bezogener Erwartungen der Handelnden. Zur Handhabung des Problems werden im Rahmen der Planung Selbstbeschreibungen der Unternehmung sowie Beschreibungen der Umwelt unterstellt und zugleich verändert. Die Reflexion der Wirklichkeitskonstruktion dient dazu, Handlungsmöglichkeiten zu realisieren. Im Hinblick auf gewünschte Entwicklungen formuliert die Planung - konsensuell oder institutionell legitimierte - Ansprüche an die Handelnden und verändert damit unmittelbar die *Wirklichkeit*."

Es gibt eine Reihe von Ansätzen, die die mit diesen Herausforderungen verbundene Komplexität durch eine Fokussierung auf Teilaspekte zu reduzieren trachten. Eine lange Tradition haben Arbeiten über den Einsatz von Entscheidungsmodellen.[1] Eine Reihe von Ansätzen ist im Kontext betrieblicher Informationssysteme angesiedelt. Dazu zählt die Entwicklung von Systemen, die darauf ausgerichtet sind, strategische Planung zu unterstützen. Dabei ist das Augenmerk jeweils entweder stärker auf die zur Unterstützung bereitgestellten Informationen oder auf entsprechende Verfahren gerichtet.[2] Eine Reihe von Arbeiten zielt vor allem darauf, die hohe Unsicherheit, die mit der Planung von Informationssystemen häufig verbunden ist, zu reduzieren.[3] Demgegenüber wird in Ansätzen zum "information resource planning" (auch strategisches Informationsmanagement genannt) die wachsende Bedeutung von Information für erfolgreiches Handeln in Unternehmen betont. Sie zielen deshalb auf die Ermittlung und Bereitstellung - unter Berücksichtigung dazu geeigneter Informationstechnik - von Informationen, denen eine strategische Funktion zukommt.[4] Andere Arbeiten gehen von der Hypothese aus, daß die Gestaltung betrieblicher Informationssysteme für die Wettbewerbsfähigkeit eines Unternehmens von zentraler Bedeutung ist ("strategic weapon"). Sie untersuchen deshalb Beziehungen zwischen bestimmten Ausprägungsformen von Informationssystemen und möglichen Wettbewerbsvorteilen.[5] In neuerer Zeit wird zwischenbetrieblichen Absprachen zur Gewinnung

1. Vgl. Frese (1971), Bretzke (1980), Lenz (1987).
2. Entsprechende Arbeiten reichen zurück in die Anfangszeiten der Untersuchungen über Management Information Systems (Blumenthal 1969; Prince 1970), bei denen die Betonung auf Bereitstellung und Aufbereitung entscheidungsrelevanter Information lag. Im Rahmen von Decision Support Systems (Keen/Scott Morton 1978) wurden einige unrealistische Annahmen, die mit Management Information Systems verbunden waren, revidiert (ein Vergleich der Ansätze findet sich in Frank 1988). In jüngeren Ansätzen werden Werkzeuge präsentiert, die auf fortschrittlicheren softwaretechnischen Konzepten beruhen (Mockler/Dologite 1987; Gibson/Snyder/Carr 1990).
3. Vgl. Klotz/Strauch (1990), Ward/Griffiths/Whitmore (1990), Boynton/Zmud (1987), Henderson/Sifonis (1988), Curtice (1987). Werkzeuggestützte Ansätze finden sich in Hoyer/Kölzer (1986) sowie in diversen Beiträgen in Schönecker/Nippa (1987).
4. Vgl. Martiny/Klotz (1989), Gongla et al. (1988), Lederer/Mendelow (1987), Fischbacher (1986).
5. So in Wiseman (1985), Porter/Millar (1985), Ives/Learmonth (1984). Eine kritische Stellungnahme dazu findet sich Benjamin et al. (1990).

strategischer Vorteile besondere Aufmerksamkeit zuteil. Hier ist an Schlagworte wie "strategische Allianzen" oder "outsourcing" zu denken.[1]

Die in den skizzierten Ansätzen thematisierten Sachverhalte sind für die Ausgestaltung der strategischen Perspektive allesamt von Bedeutung. Keiner der Ansätze ist allerdings hinreichend für unseren Zweck. So scheint es nicht sinnvoll, eine explizite Einschränkung auf die Planung von Informationssystemen vorzunehmen: Die Auswahl und Beurteilung von Informationssystemen steht in Wechselwirkung mit Gesamtstrategien. Eine getrennte Behandlung der Planung von Informationssystemen wird zudem durch die hohe (und weiter zunehmende) Durchdringung von Unternehmen mit Informationstechnik in Frage gestellt. Stattdessen benötigen wir einen Ansatz, der die mit der strategischen Planung verbundenen Aufgaben zu strukturieren und für die Modellbildung aufzubereiten gestattet. In der Literatur ist eine Fülle solcher generellen Planungsmethoden zu finden.[2] Darüber hinaus haben viele Unternehmensberater eigene Methoden entwickelt.[3] Vor dem Hintergrund des hier favorisierten Perspektivenbegriffs und der mit Unternehmensmodellen verbundenen Anforderungen sollte eine Methode zur Unterstützung der strategischen Planung vor allem zwei Anforderungen erfüllen:

- Sie sollte so detailliert sein, daß die wesentlichen Zusammenhänge erfaßt und analysiert werden können. Da Planung auch durch das Bemühen um Differenzierung gekennzeichnet ist, sollten Möglichkeiten zu unternehmensindividuellen Spezialisierungen vorgesehen sein.
- Die verwendeten Konzepte sollten deutliche Parallelen zu den Konzeptualisierungen aufweisen, die die Wahrnehmung der in Unternehmen mit strategischer Planung beschäftigten Manager leitet. Sie sollten also konkrete Bezüge zur Planungspraxis aufweisen und in anschaulicher Weise präsentiert werden.

Porter (1985) hat einen Ansatz zur Unterstützung der strategischen Planung entwickelt, der diesen Forderungen wohl am ehesten entspricht. Er ist weit verbreitet und erfreut sich einer bemerkenswert großen Akzeptanz in der Praxis. Diese Akzeptanz dürfte nicht zuletzt daher rühren, daß Porter keine völlig neue Sichtweise einführt, sondern vielmehr weitgehend auf bekannte Strukturierungen[4]

1. Vgl. Badaracco (1991), Rosenberg/Saloner (1991), Jarillo/Stevenson (1991), Nielsen (1988).
2. Ein umfassender Überblick findet sich in Hassey (1992). Scott Morton (1986) bietet ebenfalls eine vergleichende Analyse prominenter Ansätze. Neben dem hier ausgewählten Ansatz von Porter hat vor allem die von Rockart (1979) vorgeschlagene Methode der "critical success factors" eine weite Verbreitung gefunden.
3. Eine der wenigen in rudimentärer Form veröffentlichen Methoden ist das "business system concept" von McKinsey&Co. (Gluck 1980).
4. Das gilt sowohl für Ansätze aus dem wissenschaftlichen Bereich wie auch für Methoden aus der Beratungspraxis.

zurückgreift, die er in einheitlicher Systematik ordnet. Im folgenden wird der Ansatz zunächst im Überblick und in seiner Anwendung dargestellt. Anschließend sind die verwendeten Konzepte in einer für die Modellierung angemessenen Weise semantisch zu rekonstruieren.

4.1 Das Konzept der Wertkette

Porter geht davon aus, daß die strategische Unternehmensplanung im wesentlichen darauf zu richten ist, die Wettbewerbsposition so zu profilieren, daß daraus ein größtmöglicher Vorteil resultiert:

> "Competitive strategy is the search for a favorable competitive position in an industry, the fundamental arena in which competition occurs. Competitive strategy aims to establish a profitable and sustainable position against the forces that determine industry competition." (Porter 1985, S. 1)

Als wesentliche externe Einflußfaktoren der Wettbewerbssituation ("industry structure") nennt Porter die unmittelbaren Konkurrenten, die Käufer und ihre Präferenzen, die Lieferanten sowie die Anbieter von Substitutionsgütern und Unternehmen, die in Zukunft als Konkurrenten auftreten können ("potential entrants"). Um die internen und damit eher zu kontrollierenden Einflußfaktoren zu erfassen und sie gegebenenfalls durch geeignete Maßnahmen zu verbessern, schlägt Porter das Konzept einer "value chain" vor:

> "The value chain disaggregates a firm into its strategically relevant activities in order to understand the behavior of costs and the existing and potential sources of differentiation. A firm gains competitive advantage by performing these strategically important activities more cheaply or better than its competitors." (Porter 1985, S. 33 f.)

Eine Wertkette beschreibt die verschiedenen Stufen des Wertschöpfungsprozesses in einem Unternehmen. Dazu unterscheidet Porter fünf sogenannte "primary activities". Die Primäraktivitäten dienen unmittelbar der physischen Erstellung der vom Unternehmen am Markt angebotenen Produkte bzw. Leistungen. Sie umfassen Eingangslogistik ("inbound logistics"), Leistungserstellung ("operations"), Ausgangslogistik ("outbound logistics"), Marketing und Vertrieb ("marketing&sales") und Kundendienst ("service"). Im Unterschied dazu stehen Unterstützungsaktivitäten ("support activities") allen Stufen des Wertschöpfungsprozesses zur Verfügung. Porter zählt dazu Infrastruktur ("firm infrastructure"), Personalwesen ("human resource management"), Forschung und Entwicklung ("technology development") und Beschaffung "procurement".

Der Unterschied zwischen Eingangslogistik und Beschaffung ist danach darin zu sehen, daß Eingangslogistik unmittelbar auf die zu erstellenden Produkte oder Dienstleistungen gerichtet ist, während Beschaffung unter anderem die generellen Verwaltungsdienste zur Abwicklung von Beschaffungsvorgängen bereitstellt. Die Infrastruktur umfaßt unter anderem die Unternehmensführung, das Finanzwesen, die Planung und das Rechnungswesen. Die Wertkette ist darauf gerichtet,

deutlich zu machen, welchen Beitrag einzelne Wertaktivitäten dazu leisten, den Wert der vom Unternehmen angebotenen Leistungen aus der Sicht der Kunden anzureichern ("added value"). Daneben sind die Aktivitäten durch die jeweils verursachten Kosten zu kennzeichnen - soweit dies möglich ist. Dabei sind auch die (konkurrierenden oder komplementären) Beziehungen zwischen Wertaktivitäten zu berücksichtigen. Die Gestaltung der Aktivitäten zielt (natürlich) darauf, den angebotenen Leistungen einen solchen Wert zu verleihen, daß die Kunden bereit sind, dafür mehr zu bezahlen als die für die Leistungserstellung erforderlichen Kosten.[1] Die skizzierten Zusammenhänge, die auf dieser Ebene kaum über betriebswirtschaftliche Propädeutik hinausgehen, werden in einer Grafik (Abb. 46) veranschaulicht.

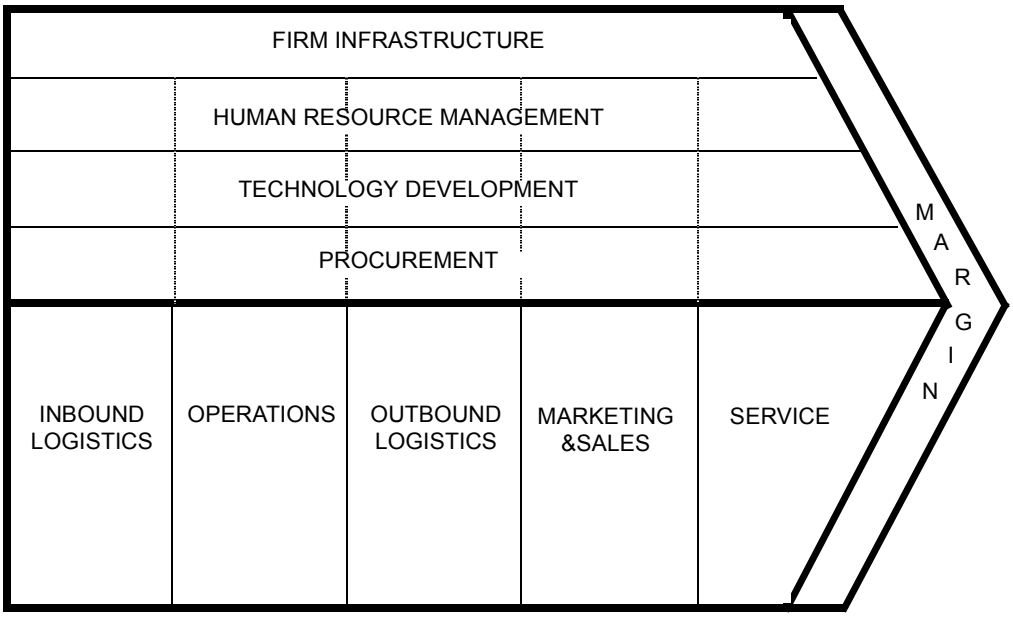


Abb. 46: Darstellung der Wertkette nach Porter (1985, S. 37)

Die eigentliche Bedeutung des Wertketten-Ansatzes liegt darin, daß Porter eine Reihe von Richtlinien zur Analyse von Wertketten bereitstellt. In einer Ist-Analyse sind die Aktivitäten eines Unternehmens zu erfassen. Dazu werden die Aktivitäten der Wertkette in konkrete Aktivitäten des Unternehmens differenziert. Dabei wird nicht genau festgelegt, wie weit eine solche Differenzierung gehen sollte.[2] Die Teilaktivitäten sind dann durch den von ihnen verursachten Ressour-

1. "Margin is the difference between total value and the collecitve cost of performing the value activities." (Porter 1985, S. 38)
2. "... the basic principle is that activities should be isolated and separated that (1) have different economics, (2) have a high potential impact of differentiation, or (3) represent a significant or growing proportion of cost." (Porter 1985, S. 45)

cenbedarf und die durch sie erbrachten Leistungen zu kennzeichnen. Anschließend sind diese Ein- und Ausgänge zu bewerten. Porter weist nachdrücklich darauf hin, daß hier unterschiedliche Bewertungsansätze angewendet werden können (wie beispielsweise Buchungswert oder Wiederbeschaffungswert). In jedem Fall zielt die Bewertung darauf, eine prozentuale Verteilung von Nutzen (in Geldeinheiten bewertet) und Kosten auf die einzelnen Aktivitäten zu realisieren.

Anschließend ist eine von drei "generic strategies" auszuwählen, auf die dann die gegebenenfalls nötige Rekonfiguration der Wertkette auszurichten ist. Zu den generischen Strategien zählt Porter Kostenführerschaft, Differenzierung und Fokussierung (die Spezialisierung auf ein bestimmtes Marktsegment). Die einzelnen Aktivitäten sind nun im Hinblick auf die gewählte Strategie zu beurteilen. Dazu ist im Einzelfall zu prüfen, ob sie anders gestaltet oder ersetzt werden können. Um Hinweise darauf zu erhalten, schlägt Porter eine Differenzierung in "direct activities", "indirect activities" und "quality assurance" vor. Direkte Aktivitäten führen unmittelbar zu einem für den Kunden feststellbaren Wert. Auch wenn es sich dabei vorrangig um Primäraktivitäten handelt, können auch Unterstützungsaktivitäten direkte Aktivitäten sein. So bezeichnet Porter Forschungs- und Entwicklungsarbeiten als direkte Aktivitäten (sie beeinflussen direkt die Qualität eines Produkts), gleichwohl sie der Unterstützungsaktivität "technology development" angehören. Indirekte Aktivitäten dienen dazu, direkte Aktivitäten zu ermöglichen.

Neben der Bewertung von Aktivitäten durch die Bewertung des je zugerechneten Inputs und Outputs ist die Untersuchung der Beziehungen zwischen Aktivitäten von zentraler Bedeutung. Dabei ist daran zu denken, daß Kostenänderungen in einer Aktivität zu Kosten- oder Nutzenänderungen in anderen Aktivitäten führen können: "At the essence of value analysis are the trade-offs to be made among price, quality, design, manufacturability, standardization, and cost." (Hax/ Majluf 1991, S. 305). Daneben können Aktivitäten zusammengefaßt werden, um Einsparungen zu realisieren. Für jede Aktivität, die sich dafür anbietet, kann geprüft werden, ob die von ihr erbrachte Wertschöpfung nicht an einen externen Partner ausgelagert werden kann. Um darüber hinaus die Beziehungen der Aktivitäten zu externen Faktoren berücksichtigen zu können, schlägt Porter vor, die Wertketten von Kunden und Lieferanten so weit wie sinnvoll mit zu berücksichtigen.[1] Daneben erörtert er Möglichkeiten durch Allianzen mit anderen Unternehmen strategische Vorteile zu erlangen, indem auf diese Weise Kosten von Aktivitäten gesenkt werden bzw. ihr Wert erhöht wird. Die kurze Skizze macht deutlich, daß die Analyse von Beziehungen wichtige Erkenntnisse für die Verbesserung von Wertketten liefert, gleichzeitig aber auch mit erheblichen Anforderun-

1. Ein Gefüge mehrerer zusammenhängender Wertketten nennt Porter (1985, S. 34) "value system".

gen verbunden ist. Dazu Porter (1985, S. 50): "Managing linkages thus is a more complex organizational task than managing value acitivities in themselves."

Die Betrachtung von Beziehungen zwischen Wertaktivitäten wird ergänzt durch eine Reihe von Maßnahmen zur Erreichung strategischer Vorteile, die jeweils unter Hinweis auf mögliche Vor- und Nachteile beurteilt werden. Darüber hinaus gibt Porter Handlungsempfehlungen für das Vorgehen bei der Analyse von Kunden und Konkurrenten oder der Abgrenzung von Segmenten.[1]

Die Wertkette liefert keine unmittelbaren Planungsempfehlungen. Das Konzept stellt vielmehr einen Bezugsrahmen dar, der es gestattet, Strategien in systematischer Weise zu entwerfen - unter Rückgriff auf unternehmensspezifische Gegebenheiten und deren Beurteilung. In diesem Sinne betont Scott Morton (1986, S. 14) die Bedeutung von "management's informed judgement".

Die Darstellung erfolgt in einem didaktisch motivierten Stil, in dem sich analytische und narrative Elemente einander abwechseln. Dazu reichert Porter die abstrakte Beschreibung von Konzepten immer wieder mit konkreten Beispielen an - ein Umstand, der für die Unternehmensmodellierung zu berücksichtigen sein wird. Auf diese Weise stellen die mit dem Konzept einhergehenden Erörterungen einen auf die Anforderungen der Planung fokussierten Querschnitt durch betriebswirtschaftliches Wissen dar - das im Einzelfall durch den Hinweis auf speziellere Abhandlungen gewiß noch angereichert werden könnte.

4.2 Aufbereitung des Ansatzes für die Unternehmensmodellierung

Der Ansatz von Porter zielt vor allem darauf, einen Bezugsrahmen für den Entwurf strategischer Pläne zu liefern. Dabei werden gewisse betriebswirtschaftliche Kenntnisse vorausgesetzt, andere durch kurze Darstellungen, die häufig durch Beispiele angereichert sind, erläutert. Diese lehrbuchartige Präsentation des Ansatzes ist für eine unmittelbare Umsetzung in ein Modell nicht geeignet - zumal dann, wenn das Modell Grundlage für die Entwicklung eines Werkzeugs sein soll. Die für den Ansatz charakteristischen Begriff sind also in einer für die Modellbildung angemessenen Weise semantisch zu rekonstruieren. Die besondere Problematik einer solchen Rekonstruktion liegt darin, daß nicht alle von Porter behandelten Zusammenhänge für alle Unternehmen invariant sind. Die Übertragung auf ein konkretes Unternehmen - also die spezifische Interpretation - erfordert eine entsprechende Kompetenz der beteiligten Manager.[2] Dieser Umstand hat zur Folge, daß ein vollständiges Modell mit allen für die strategi-

1. Ein ausführliches Beispiel für den Entwurf und die Bewertung von Segmenten findet sich in Porter (1985, S. 255).

2. Scott Morton (1986, S. 14) spricht in diesem Zusammenhang - durchaus wohlwollend - von "vehicles for management to discuss the potential of their businesses".

sche Planung bedeutenden Zusammenhängen und deren Wirkungsweise nicht angestrebt werden sollte. Ein solches Anliegen stößt auf ein bekanntes Problem: Die Formalisierung betriebswirtschaftlicher Entscheidungskompetenz zum Zweck automatisierter Entscheidungsprozesse. Es gab eine Reihe ambitionierter Ansätze, dieses Problem zu überwinden.[1] Eine vollständige Automatisierung von Entscheidungen unter Unsicherheit bzw. Kontingenz ist jedoch allenfalls durch vereinfachende Annahmen zu realisieren, die dazu führen, daß die so ermittelten Empfehlungen gar nicht oder jedenfalls nicht unmittelbar umzusetzen sind. In diesem Sinne Bretzke (1980, S. 35):

> "Entscheidungsmodelle können nicht als *Rekonstruktionen* unabhängig vorgegebener Strukturkomplexe gedacht werden, sie sind vielmehr als Konstruktionen zu denken, mit denen einem Problem regelmäßig eine Eigenschaft hinzugefügt wird, die ihm ursprünglich nicht zukam: Entscheidbarkeit."

Um bei der Verwendung des Ansatzes von Porter solche dysfunktionalen Verkürzungen zu vermeiden, ist es angeraten, die Konzepte danach zu unterscheiden, ob sie einer Formalisierung zugänglich sind oder nicht. Dazu differenzieren wir im folgenden zwei Ebenen des Ansatzes. Auf der *Beschreibungsebene* verwendet Porter Begriffe, deren Bedeutung sich eher formal beschreiben läßt als die Begriffe auf der *Analyse-Ebene*.[2] Die Integration von strategischer, organisatorischer und Informationssystem-Perspektive verspricht eine erhebliche Anreicherung des Wertketten-Ansatzes. So kann der Durchgriff auf die organisatorische Ebene unmittelbar Aufschluß über Kosten liefern, die sonst erst berechnet werden müßten. Dazu sind den Aktivitäten beispielsweise einzelne Vorgänge oder sonstige Kostenträger auf der organisatorischen Ebene zuzuordnen. Die Beziehungen zwischen den Konzepten der einzelnen Ebenen werden allerdings erst nach der isolierten Darstellung der Perspektiven skizziert.

4.2.1 Konzepte der Beschreibungsebene

Die wesentlichen Konzepte des Wertketten-Ansatzes sind Aktivitäten. Im Hinblick auf die Modellierung ist dabei zwischen Aktivitätsgruppen (also "inbound logistics", "operations" etc.) und den ihnen zugeordnenten Aktivitäten (denen wiederum andere Aktivitäten untergeordnet sein können) zu unterscheiden. Der *Typ* einer Aktivität (bzw. der zugehörigen Aktivitätsgruppe) ist entweder "pri-

1. Dazu zählen frühe Forschungsarbeiten im Bereich Management Information Systems oder auch kühne Übertragungen der Homunkulus-Metapher auf Manager. In diesem Sinne prophezeite Simon (1964, S. 594): "Within the very near future - much less than twenty five years - we shall have the *technical* capability for substituting machines for any and all human functions in organizations." Eine ausführliche Analyse der Formalisierung von Entscheidungskompetenz findet sich in Frank (1988).
2. Eine solche Trennung stellt eine bewußte analytische Vereinfachung dar, denn auch zur Beschreibung gehören unter Umständen Sachverhalte (wie Kosten oder Werte), deren Ermittlung eine Analyse voraussetzt.

mär" oder "unterstützend". Die Unterscheidung in "direkt", "indirekt" und "qualitätssichernd" soll hier als *Ausrichtung* bezeichnet werden.

Die Konzepte der strategischen Ebene weisen mitunter eine deutliche Korrespondenz zu Konzepten der anderen beiden Ebenen auf. Sie sind allerdings auf einer anderen Abstraktions- beziehungsweise Aggregationsstufe angesiedelt. So zielt beispielsweise das Konzept "Human-Ressource" auf eine vom einzelnen Mitarbeiter abstrahierende Beurteilung des Leistungspotentials aller Mitarbeiter eines bestimmten Bereichs. Bei der Spezifikation der Klassen ist einerseits zu berücksichtigen, daß in der strategischen Perspektive einer monetären Bewertung eine herausragende Rolle zukommt. Daneben ist dem Konflikt zwischen den Vor- und Nachteilen einer formalisierten Beschreibung von Sachverhalten Rechnung zu tragen: Eine formale Beschreibung schafft einerseits günstige Voraussetzungen für weitere Analysen, ist aber andererseits mit der Gefahr verzerrender Verkürzungen verbunden. Das folgende Beispiel zeigt ausgewählte Merkmale eines Konzepts, die durch einen unterschiedlichen Formalisierungsgrad gekennzeichnet sind. Die Klasse "Multimedia" drückt dabei lediglich den Freiheitsgrad der Präsentation aus, nicht die Notwendigkeit, verschiedene Darstellungsformen zu verwenden.

Human-Ressource

Merkmal	*Alternative Formalisierungen*	
Bezeichnung	Zeichenkette	Zeichenkette aus vorgegebener Liste
Arbeitsmarktsituation	Multimedia	< günstig, unkritisch, problematisch >
Qualifikationsstand	Multimedia	< herausragend, durchschnittlich, unterentwickelt >
Investitionen in Weiterbildung	Geldbetrag	Geldbetrag

Abb. 47: Alternative Formalisierungen von Merkmalen der Klasse "Human-Ressource"

Im folgenden werden ausgewählte Generalisierungsbeziehungen[1] zwischen den Konzepten (Klassen) der Beschreibungsebene sowie Beziehungen zwischen den zugehörigen Exemplaren (Instanzen) dargestellt. Auch wenn die wesentlichen Konzepte des Ansatzes von Porter berücksichtigt werden, wird jedoch - wie auch bei der Beschreibung der beiden anderen Hauptperspektiven - kein Anspruch auf

1. Wie bei den Konzepten der organisatorischen Perspektive handelt es sich dabei um eine flache Vererbungshierarchie.

Vollständigkeit erhoben. So lassen sich beispielsweise im Hinblick auf die Beschreibung der Kostenstruktur erheblich feinere Differenzierungen einführen. Die Bezeichnungen der skizzierten Klassen stimmen zum Teil mit Bezeichnungen von Klassen in anderen Teilmodellen überein (so ist etwa eine Klasse "Betriebsmittel" auch in den Modellen der organisatorischen Perspektive sowie der Informationssystem-Perspektive verzeichnet - allerdings jeweils mit anderer Bedeutung).

Eine besondere Schwierigkeit der Modellierung des Wertketten-Ansatzes ergibt sich daraus, daß einzelne Gegenstände kontextabhängig betrachtet werden. Das gilt generell für die Differenzierung von Input und Output, spezieller für die Unterscheidung von Kosten und Umsatz oder benötigten Ressourcen und erbrachten Leistungen. Im Kontext einer Aktivitätsgruppe ist die Unterscheidung problemlos durchzuführen. Sobald man aber den Austausch zwischen Aktivitätsgruppen betrachtet, wird es problematisch: So ist beispielsweise eine im Unternehmen erstellte und weiterverwendete Statistik am Ort der Erstellung eine Leistung, am Ort der Nutzung eine Ressource. Während Konzepte wie "Ressource" oder "Erbrachte Leistung" bei einer statischen Betrachtung unabhängig voneinander definiert werden können, sind bei einer Betrachtung der Prozesse in einer Wertkette inhaltliche Beziehungen zwischen den Konzepten herzustellen. Im Hinblick auf eine werkzeuggestützte Aufbereitung der Zusammenhänge ist zu berücksichtigen, daß ein Objekt während seiner Lebenszeit seine Klasse nicht ändern kann.[1] Es wird deshalb eine Klasse "Leistung" eingeführt, deren Instanzen durch entsprechende Beziehungen sowohl als Ressourcen wie auch als erbrachte Leistungen bewertet werden können (vgl. Abb. 48).

1. In der Literatur findet sich eine Reihe von Beiträgen, in denen die Migraton eines Objekts von einer Klasse in eine andere diskutiert wird (Odell 1992 a, Velho/Carapuca 1992). Jenseits der dazu erforderlichen sprachtechnischen Voraussetzungen (die beispielsweise in Smalltalk mit vertretbarem Aufwand erfüllt werden können) bleiben aber noch gewichtige Probleme, so daß hier von der Möglichkeit solcher Migrationen abgesehen wird.

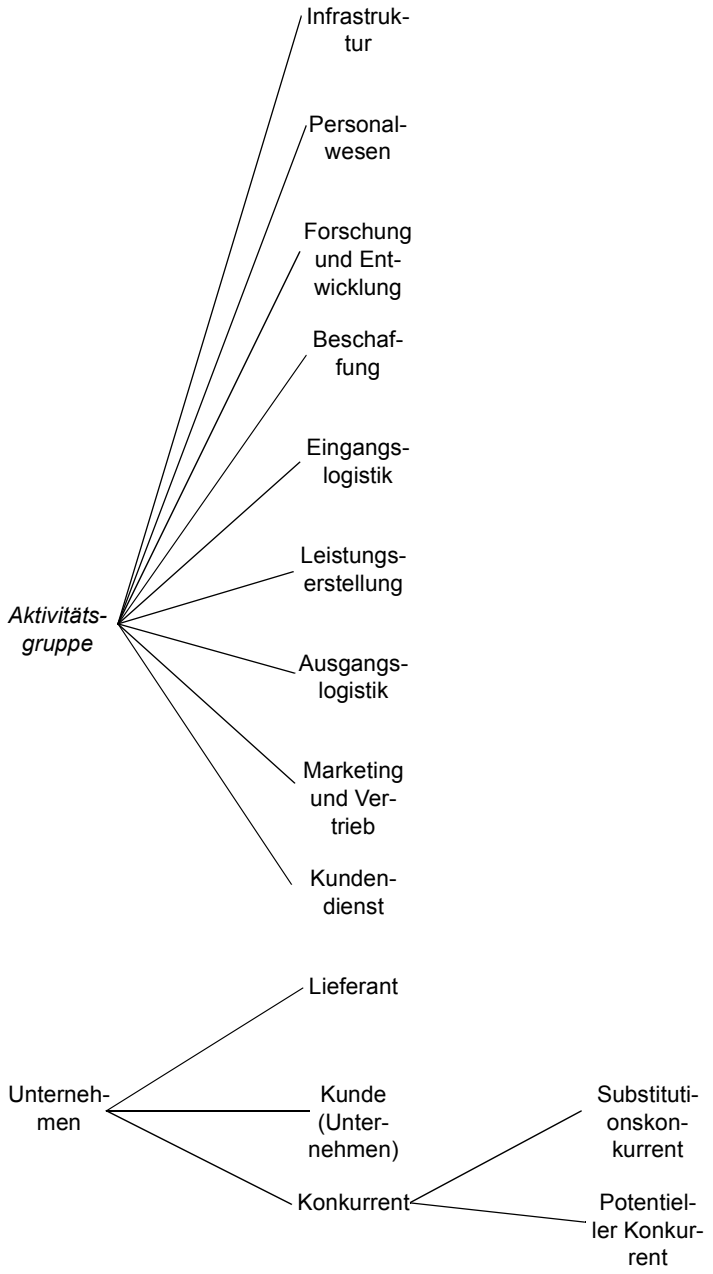
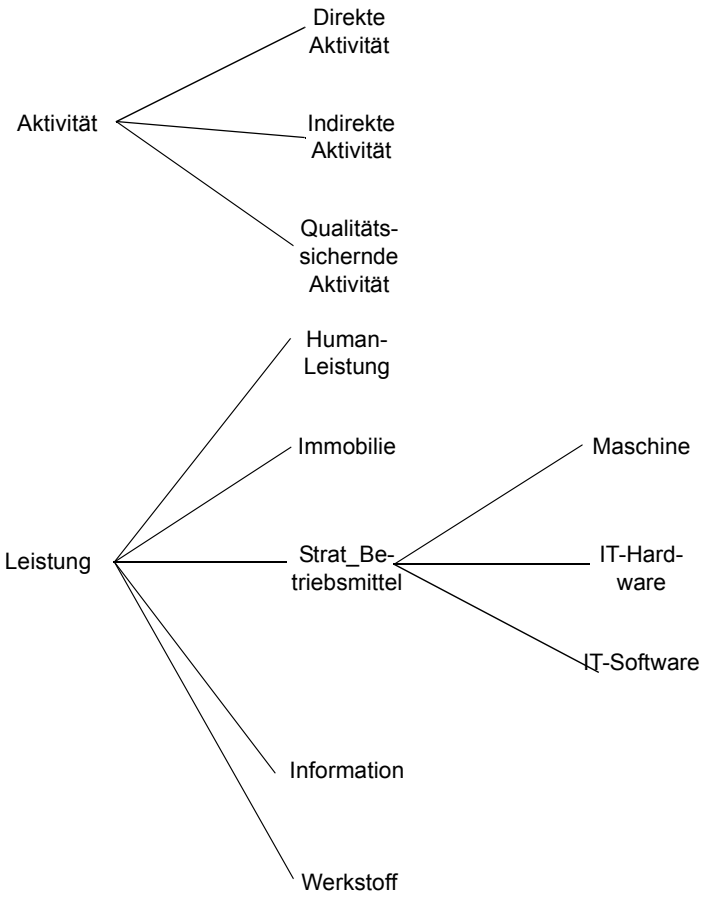


Abb. 48: Ausgewählte Generalisierungsbeziehungen der strategischen Perspektive, Fortsetzung folgende Seite



Bei der Anwendung des Porterschen Ansatzes auf ein Unternehmen ist der besondere Aufwand zu berücksichtigen, der mit der individuellen Instanzierung der vorgeschlagenen Konzepte in der Regel verbunden ist. In der Praxis äußert sich dieser Umstand darin, daß die Konkretisierung einer Wertkette typischerweise im Rahmen dedizierter Projekte erfolgt. Porter (1985, S. 65) selbst macht unmißverständlich deutlich, daß sein Ansatz lediglich eine Orientierung bietet. Die Identifikation und Ausgestaltung einer Wertkette hat unternehmensindividuell zu erfolgen.

Zur Beschreibung einer Wertkette ist die ökonomische Bewertung der in den einzelnen Aktivitäten verzehrten Ressourcen und erbrachten Leistungen von zentraler Bedeutung. Um den damit verbundenen Aufwand zu reduzieren, können Schätzungen durchgeführt werden: "It is important to remember that assigning costs and assets does not require the precision needed for financial reporting purposes." (Porter 1985, S. 67). Demgegenüber bietet eine werkzeuggestützte Inte-

gration mit operativen Ebenen (hier: mit den Konzepten der organisatorischen und der Informationssystem-Perspektive) die Chance, einen Teil der für die Wertketten-Darstellung benötigten Daten durch den Zugriff auf diese Ebenen (wo sie in aktueller und präziser Form verwaltet werden) zu erhalten (vgl. dazu V.4).

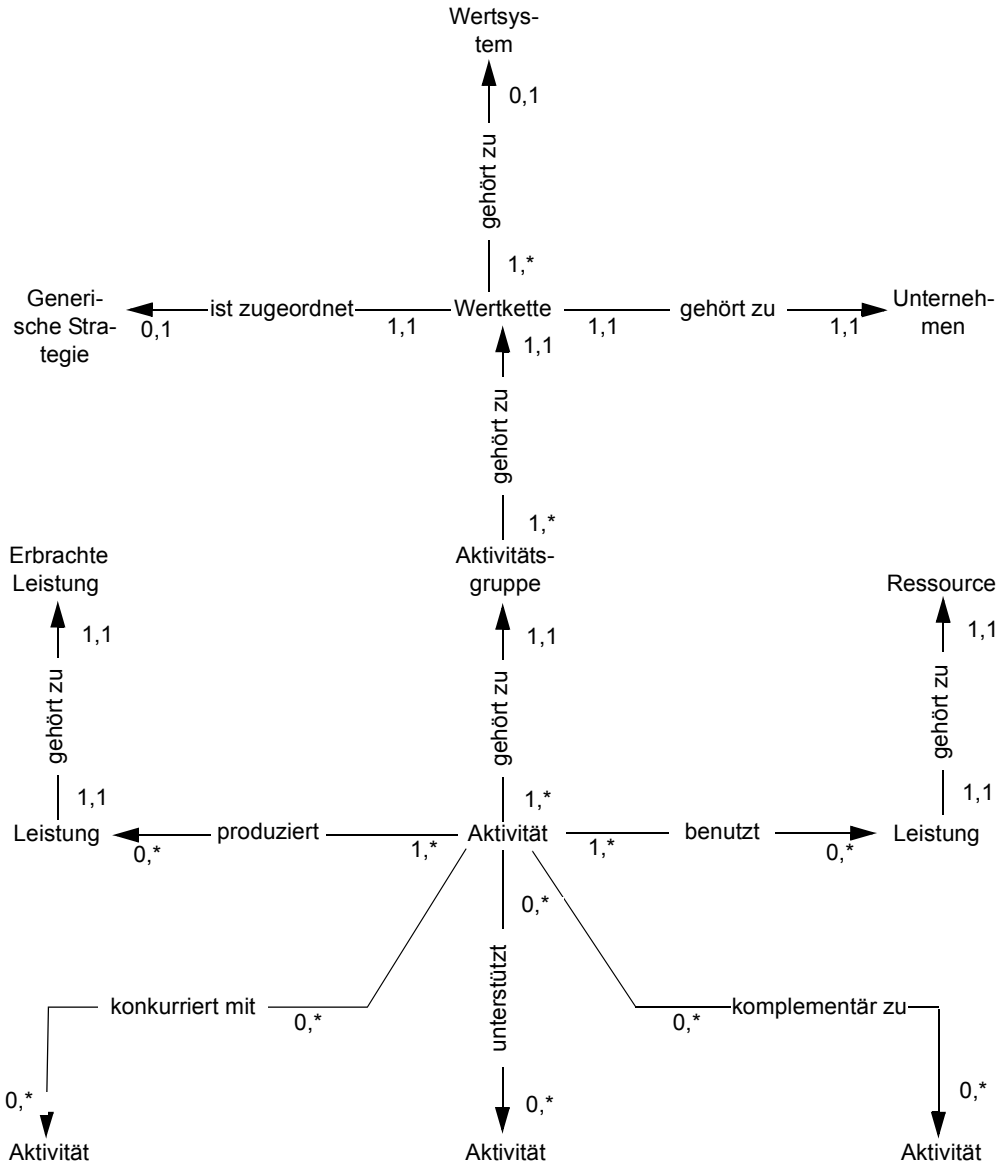


Abb. 49: Ausgewählte Beziehungen zwischen Objekten der strategischen Perspektive.

4.2.2 Konzepte der Analyse-Ebene

Ähnlich wie für die deskriptive Rekonstruktion des Ist-Zustands einer Unternehmung liefert der Wertketten-Ansatz auch für die daran anknüpfende Analyse nur einen abstrakten Bezugsrahmen, zu dessen detaillierter Ausfüllung auf die gesamte Bandbreite betriebswirtschaftlicher Theorien rekurriert werden kann. Im folgenden wird denn auch nur anhand einiger von Porter besonders betonter Beispiele dargestellt, wie die Analysekonzepte in die Modellbildung einfließen können. Dabei ist zu berücksichtigen, daß sich diese Konzepte weitgehend gegen eine Formalisierung sperren. Es geht also weniger um automatisierte Evaluation, sondern eher darum, dem Betrachter im Modell Analyse- und Interpretationshilfen anzubieten.

Porter verwendet vier - zum Teil miteinander assoziierte - Ansätze zur Darstellung und Vermittlung von Analysekonzepten: Checklisten, Heuristiken[1], generalisierte Wirkungszusammenhänge und Beispiele. Um die Suche nach möglichen Quellen des angestrebten Wettbewerbsvorteils zu strukturieren, werden diese Ansätze vor allem auf die drei generischen Strategien (Kostenführerschaft, Differenzierung und Fokussierung) angewandt. Dabei können neben der eigenen Wertkette auch die von Kunden, Lieferanten und Konkurrenten berücksichtigt werden. Die Analyse vollzieht sich im Wechselspiel zwischen der Beurteilung der Ist-Situation (wo liegen Stärken und Schwächen) und dem Entwurf und der Beurteilung von Alternativen.

Die Analyse einer Wertkette setzt an den drei von Porter genannten generischen Strategien an. In vereinfachter Form läßt sich die Vorgehensweise dabei so skizzieren (vgl. Abb. 50): Zunächst wird nach einer sorgfältigen Evaluation eine generische Strategie ausgewählt. Anschließend wird die zuvor identifizierte Wertkette eines Unternehmens im Lichte dieser Strategie unter Rückgriff der von Porter angebotenen Checklisten und Heuristiken beurteilt. Danach sind gegebenenfalls strategische Optionen zu prüfen. Dazu kann wiederum auf Heuristiken sowie auf Beispiele zurückgegriffen werden. Die Anwendung der von Porter vorgeschlagenen Analyse wird im folgenden anhand der generischen Strategie "Kostenführerschaft" veranschaulicht.

1. Es ist grundsätzlich gewiß nicht nötig, zwischen Checklisten und Heuristiken zu unterscheiden. Wenn dies hier dennoch geschieht, so um zwischen einer bloßen Auflistung von Kriterien und (mehr oder weniger detaillierten) Handlungsanleitungen zu unterscheiden.

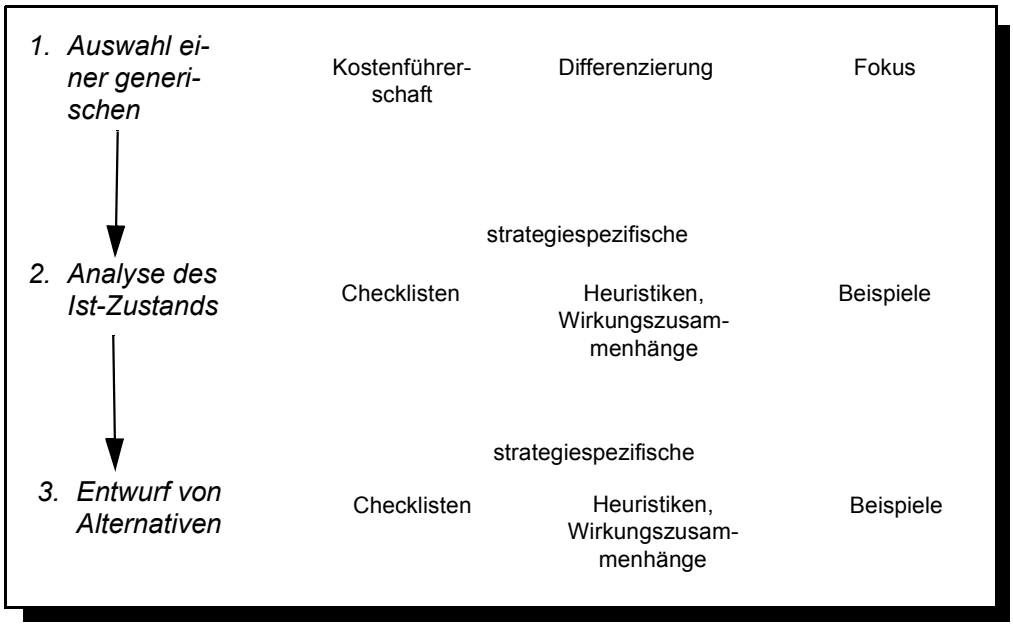


Abb. 50: Prototypische Vorgehensweise bei der Analyse von Wertketten

Um Kostenführerschaft zu erreichen, ist zunächst die eigene Kostensituation zu beurteilen. Dazu stellt Porter (1985, S. 70 ff.) eine Liste von zehn sogenannten wesentlichen Kostentreibern[1] ("major cost drivers") bereit. Sie sind auf einem relativ abstrakten Niveau angesiedelt und größtenteils in der Betriebswirtschaftslehre - besonders in der Produktionstheorie - hinlänglich bekannt. Zu ihnen gehören unter anderen: die Losgrößen der Produktion, die Position und Weiterentwicklung der Lernkurve, die Auslastung diverser Kapazitäten und der Standort. Von besonderer Komplexität ist die für den Wertketten-Ansatz spezifische Untersuchung der Kostenbeziehungen zwischen den einzelnen Aktivitäten der eigenen Wertkette sowie zu den Aktivitäten der Wertketten von Lieferanten und Abnehmern. Im Hinblick auf die Modellierung ist zu berücksichtigen, daß die Kostentreiber lediglich Hinweise darauf liefern, wo Kosten reduziert werden können: "Identifying cost drivers and quantifying their effect on cost may not be easy ..." (Porter 1985, S. 87). Für die Beurteilung der Kostentreiber empfiehlt Porter neben unternehmensinternen ex post-Analysen Experteninterviews und, angesichts des Ziels, Kostenführerschaft zu erreichen, von zentraler Bedeutung, den Vergleich mit entsprechenden Werten von Wettbewerbsunternehmen - sofern sie verfügbar sind.

Zur Senkung der Kosten bietet Porter für jeden Kostentreiber eine Liste von

1. Es wäre inhaltlich angemessener, sprachlich allerdings ein wenig bizarr, von "Kostenbeeinflussungsfaktoren" zu sprechen.

Maßnahmen. Sie sind allerdings abstrakt gehalten, so daß ihre Anwendung auf eine spezifische Wertkette ein erhebliches Maß an zusätzlicher Analyse erfordert. Dieser Umstand wird durch die folgenden Maßnahmen zur Kostenkontrolle von Losgrößen und Lernkurven veranschaulicht (Porter 1985, S. 100 f.), die jeweils durch einen kurzen Text erläutert werden:

Controlling Scale

- Gain the Appropriate Type of Scale.
- Set Policies to Reinforce Scale Economies in Scale-Sensitive Activities.
- Exploit the Types of Scale Economies Where the Firm is Favored.
- Emphasize Value Activities Driven by Types of Scale Where the Firm has an Advantage.

Controlling Learning

- Manage with the Learning Curve
- Keep Learning Proprietary
- Learn from Competitors

Neben der Modifikation einzelner Aktivitäten kann zur Umsetzung der jeweils gewählten generischen Strategie auch eine Rekonfiguration der gesamten Wertkette vorgenommen werden. Es liegt auf der Hand, daß es sich dabei um eine komplexe Aufgabe handelt - und daß die Theorie der strategischen Planung im allgemeinen, der Portersche Ansatz im besonderen hier keine konkrete Handlungsanleitung erwarten lassen.

Porter (1985, S. 107 ff.) nennt eine Reihe von Veränderungen, die eine Rekonfiguration der Wertkette mit sich bringen. Dazu gehören unter anderem Veränderungen des Produktionsprozesses, neue Vertriebskanäle sowie neue Werbemedien. Die Heuristik zur Ermittlung einer alternativen Wertkette präsentiert sich ausgesprochen dürftig (Porter 1985, S. 110). Danach sollten die folgenden Fragen sorgfältig beantwortet werden:

- How can the activity be performed differently or even eliminated?
- How can a group of linked value acitivities be reordered or regrouped?
- How might coalitions with other firms lower or eliminate costs?

Der Informationsgehalt der von Porter skizzierten Analysemethoden macht deutlich, daß eine automatische Analyse von Wertketten oder gar die Generierung alternativer Wertketten in der Regel nicht möglich ist. Dennoch können die Konzepte im Rahmen einer werkzeuggestützten Modellierung genutzt werden. Dazu könnten Analyseobjekte eingeführt werden. Da die für die Analyse bedeutsamen Zusammenhänge allenfalls zu einem geringen Teil in generalisierter Form vorliegen und der größte Teil unternehmensspezifisch zu erheben ist, bietet es sich an,

einen deklarativen Repräsentationsformalismus zu verwenden. Eine solche deklarative Darstellung des Analysewissens bietet - Monotonie des verwendeten Formalismus vorausgesetzt - den Vorteil, neue Zusammenhänge relativ komfortabel hinzufügen zu können. Die Analyseobjekte verwalten also Wissensbasen und bieten Schnittstellen zu deren Aktualisierung und zur Durchführung von Inferenzprozessen. In einem ersten Schritt könnten einfache Regeln zur Verwaltung der von Porter vorgeschlagenen Checklisten implementiert werden - wie beispielsweise: "Wenn Strategie ist Kostenführerschaft, dann präsentiere Kostenkontroll-Kriterien". Abbildung 51 zeigt eine mögliche Generalisierungshierarchie entsprechender Klassen. In der abstrakten Oberklasse sind die genannten generellen Methoden implementiert. Daneben ist an die Einführung von Klassen zu denken, deren Instanzen der Überwachung konfigurierbarer Indikatoren dienen - wie etwa das Erreichen relativer oder absoluter Kosten bestimmter Aktivitäten.

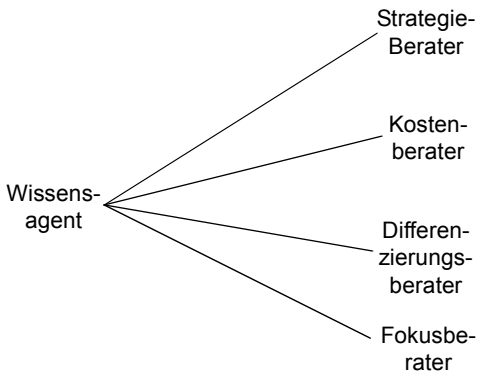


Abb. 51: Mögliche Generalisierungshierarchie für Klassen zur Anleitung von Analyseprozessen.

5. Die Integration der Perspektiven

Unter den drei hier gewählten Hauptperspektiven der Unternehmensmodellierung nimmt - wie bereits erwähnt - die Informationssystem-Perspektive eine herausragende Stellung ein. So liefert sie einerseits das Metamodell zur Beschreibung von Objektmodellen und von Vorgängen beziehungsweise Transaktionen, mithin die Konstrukte zur einheitlichen Konzeptualisierung - und damit: zur softwaretechnischen Integration - der Begriffe der drei Perspektiven. Andererseits stellte das Bemühen um die ökonomische Gestaltung und Nutzung unternehmensweiter Informationssysteme den Ausgangspunkt unserer Untersuchung dar. Schließlich werden in der Konkretisierung der Informationssystem-Perspektive,

also dem Informationssystem, auch die Objekte der beiden anderen Perspektiven verwaltet. Im folgenden werden die Beziehungen zwischen den Konzepten der drei Hauptperspektiven beschrieben. Dazu wird neben der Darstellung ausgewählter Beziehungen ein Szenario entworfen, das eine mögliche informationstechnologische Aufbereitung der Perspektiven und ihrer Verbindungen aufzeigt.

5.1 Beziehungen zwischen den Konzepten der Perspektiven

Die Konzepte der drei Perspektiven sind mehr oder weniger eng miteinander assoziiert. Die Darstellung in Abbildung 52 soll allerdings nicht suggerieren, daß das strategische Modell die Ableitung des organisatorischen Modells und das wiederum die Ableitung des Informationssystem-Modells erlaubt. Wie die Darstellung der drei Perspektiven gezeigt hat, ist ein großer Teil der Konzepte in der organisatorischen und vor allem in der strategischen Ebene allenfalls rudimentär formalisierbar. Ihre Beschreibung erfolgt deshalb wesentlich jenseits maschinell interpretierbarer Semantik unter Verweis auf Texte oder allgemeiner: auf multimediale Darstellungen, die miteinander verknüpft werden können.

Die Beziehungen zwischen den Konzepten (oder Klassen) der drei Hauptperspektiven kann man nach ihrer Semantik unterscheiden. Zwei Klassen sind eng gekoppelt, wenn es eine formale Transformation zwischen ihren Instanzen gibt. Die Kopplung nimmt dabei tendenziell mit der relativen Zahl der Instanzeigenschaften, die durch eine Transformation gewonnen werden können, zu.[1] Demgegenüber liegt eine schwache oder unscharfe Kopplung vor, wenn eine Beziehung ausgezeichnet ist, ihre Semantik sich allerdings allein durch die Interpretation des Betrachters (welche durch geeignete Annotationen gefördert werden kann) ergibt. Daneben können direkte und indirekte Beziehungen zwischen den Konzepten unterschiedlicher Perspektiven differenziert werden.

Auch wenn der Grad der Kopplung letztlich durch ein Kontinuum wiederzugeben ist, werden im folgenden aus der Vielzahl möglicher Beziehungen Beispiele aus vier prototypischen Kategorien dargestellt.

1. Damit ist allerdings kein formales Maß eines Kopplungsgrades intendiert. Es geht hier vielmehr um eine qualitative Betrachtung.

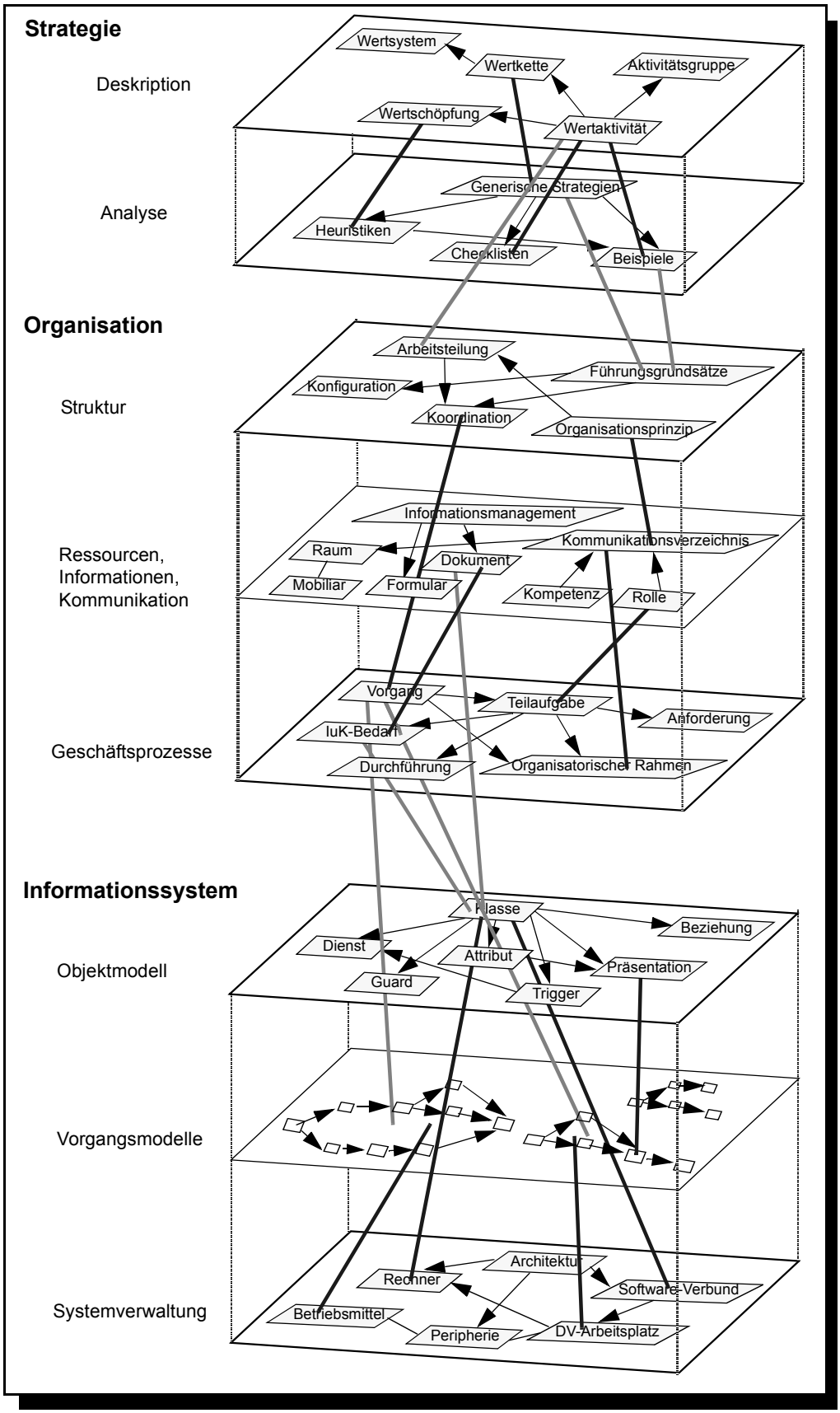


Abb. 52: Perspektiven der Unternehmensmodellierung und ihre Differenzierung

Direkte enge Kopplung

Im Organisationsmodell sind Aufgaben verzeichnet. Eine Aufgabe kann eng mit entsprechenden Instanzen des Informationssystem-Modells verbunden sein. Das gilt vor allem dann, wenn die Aufgabe vollständig automatisiert ist. Dieser Umstand könnte dann durch eine Beziehung zu Objekten der Klasse "Transaktion" (oder "Vorgang") ausgedrückt werden. Eigenschaften eines solchen Objekts (wie "auslösendes Ereignis", "maximale Dauer" etc.) können dann unmittelbar in entsprechende Eigenschaften eines Objekts der Klasse "Aufgabe" transformiert werden. Umgekehrt kann die im Organisationsmodell verwendete Beschreibung der Aufgabe als ergänzender Kommentar für die Beschreibung der Transaktion genutzt werden. Im Falle eines nur teilweise automatisierten Aufgabenerfüllungsprozesses können enge Kopplungen zwischen den automatisierten Teilaufgaben und den entsprechenden Transaktionen hergestellt werden.

Eine direkte enge Kopplung besteht auch zwischen organisatorischen Ressourcen und den Hardware- und Software-Objekten, die im Verwaltungsmodell der Informationssystem-Perspektive beschrieben sind. Hier ist an einen direkten, perspektivenübergreifenden Verweis zu denken: Die Objekte der Informationssystem-Perspektive können unmittelbar im Organisationsmodell genutzt werden. Beispiele: "Raum beinhaltet Laserdrucker", "Aufgabe erfordert DV-Arbeitsplatz", "Rolle verantwortlich für Software".

Indirekte enge Kopplung

Die Konzepte des Strategie-Modells sind zumeist auf einem wesentlich höheren Abstraktionsniveau angesiedelt als die der beiden anderen Modelle. In den Fällen, in denen dennoch eine enge Kopplung (aber keine direkte Korrespondenz zwischen zwei Konzepten) vorliegt, sind deshalb geeignete Aggregationsregeln anzugeben. So kann beispielsweise das Objekt "Informationstechnik" der strategischen Ebene durch eine Aggregation der benötigten Eigenschaften von Objekten der Klassen im Modell der Informationssystem-Verwaltung gewonnen werden. Dabei ist an Eigenschaften wie Kosten pro Zeitintervall, Kapitalbindung, Kapazitätsauslastung etc. zu denken. Daneben sind für die strategische Ebene Eigenschaften bedeutsam, die für die Verwaltung nicht notwendig sind: technologisches Niveau, Reifegrad etc. Da diese Eigenschaften an einzelne Objekte gebunden sind, ist zu überlegen, schon bei der Konzeptualisierung dieser Objekte die Anforderungen der strategischen Perspektive zu berücksichtigen.

Um die Kosten und erbrachten Leistungen von Aktivitäten innerhalb einer Wertkette zu berechnen, können Beziehungen zu den zugehörigen Aufgabenerfüllungsprozessen etabliert werden. Aus den Eigenschaften dieser Prozesse (benötigte Ressourcen, Ausführungszeiten etc.) lassen sich Kosten aggregieren. Im Hinblick auf die erbrachten Leistungen ist zu berücksichtigen, daß Leistungen im Strategie-Modell abstrakter beschrieben werden. So wird beispielsweise die Lei-

stung eines Prozesses zur Prüfung von Reklamationen aus strategischer Sicht in die Kategorie "Service" eingeordnet. Auch hier zeigt sich, daß bei der Beschreibung von Konzepten einer Ebene an die Anforderungen anderer Ebenen gedacht werden sollte - indem beispielsweise die Aufgabenerfüllungsprozesse der organisatorischen Ebene durch Leistungskategorien der strategischen Ebene gekennzeichnet werden. Es bleibt daran zu erinnern, daß Aggregationen mitunter mit komplexen Randbedingungen verbunden sind. So können beispielsweise die Kosten, die ein Aufgabenerfüllungsprozeß verursacht, nicht immer unmittelbar durch die benötigte Kapazität der zugehörigen Ressourcen berechnet werden, da der Ressourceneinsatz mitunter - hier ist vor allem an Personal zu denken - mit diskreten Kostenänderungen verbunden ist.

Zur Ermittlung der in einer Aktivität benötigten Informationstechnik kann eine indirekte Beziehung zum Modell der Informationssystem-Verwaltung genutzt werden: Die Aktivität ist unmittelbar mit den zugeordneten Aufgabenerfüllungsprozessen der organisatorischen Ebene assoziiert, die wiederum verknüpft sind mit den benötigten Informationstechnik-Objekten.

Direkte schwache Kopplung

Hier ist vor allem an Beziehungen zu (vagen) Konzepten des Strategie-Modells zu denken. So ist es unstrittig, daß die Organisation eines Unternehmens eine erhebliche Bedeutung für die Umsetzung seiner Strategien hat. In diesem Sinne Porter (1985, S. 61):

> "An organizational structure that corresponds to the value chain will improve a firm's ability to create and sustain competitive advantage. While this subject cannot be treated in detail here, it remains an important issue in the implementation of strategy."

Zwingende (mono-) kausale Zusammenhänge sind allerdings kaum bekannt. Dessen ungeachtet mögen bestimmte Ausprägungen der Unternehmensorganisation durch gewisse strategische Orientierungen motiviert sein. Die Kopplung von Strategiebestandteil (beispielsweise: "Differenzierung durch starke Kundenorientierung") und Organisation (beispielsweise: "Kunden-Center") ist insofern schwach als sich die konkrete Ausprägung des Konzepts der einen aus dem der anderen Ebene nicht transformieren läßt. Sie ist also vergleichbar mit einem Hypertext- oder Hypermedia-Link: Es besteht eine Assoziation, deren Semantik ist jedoch nicht näher formalisiert.[1]

Ähnliche Beziehungen lassen sich zwischen Strategie-Modell und Informationssystem-Modell entwerfen. Beispiel: "Differenzierung durch Einsatz fortschrittli-

1. Hypermedia-Techniken werden seit einiger Zeit zur Unterstützung der Dokumentation von Informationssystemen verwendet. Dabei handelt es sich vor allem um aktive Hilfen für Betriebssysteme oder CASE-Werkzeuge. In einer neueren Veröffentlichung skizzieren Nüttgens/Scheer (1993) ein System zur Dokumentation "unternehmensweiter Informationsmodelle".

cher Technologie" wird unterstützt durch "Verteiltes Objektverwaltungssystem". Auch für das Verhältnis von Organisationsmodell und Informationssystem-Modell können solche schwachen Kopplungen durchgeführt werden: "Flache Hierarchie" wird unterstützt durch "Electronic Mail System".

Indirekte schwache Kopplung

Die Strategie "Kundenorientierung" mit der konkreteren Ausrichtung "Berücksichtigung wirtschaftlicher Interessen der Kunden" könnte unmittelbar mit den Aufgabenerfüllungsprozessen zur Abwicklung von Aufträgen assoziiert werden. Von dort könnte dann einerseits eine Beziehung zu den Klassen des Informationssystem-Modells etabliert werden, die die Einhaltung gewisser Standards zur Übermittlung elektronischer Dokumente gewährleisten. Andererseits können Beziehungen zu den benötigten Geräten (etwa DV-Arbeitsplatz) und Übertragungsdiensten (etwa X.400) modelliert werden. Auf diese Weise wird zwischen der Strategie "Kundenorientierung" und bestimmten Klassen, die X.400-Dienste verfügbar machen, eine indirekte schwache Kopplung etabliert.

Ein anderes Beispiel: Ebenfalls ausgehend von der Strategie "Kundenorientierung" wird eine Beziehung zu der organisatorischen Regelung gelegt, wonach die komplette Betreuung eines Kunden von genau einem Mitarbeiter durchzuführen ist. Von dort wird dann auf entsprechende Personalentwicklungsmaßnahmen verwiesen.

5.2 Szenario: Anwendung eines Unternehmensreferenzmodells im Rahmen der Einführung von Informationssystemen

Neben dem individuellen Entwurf eines multiperspektivischen Modells für ein bestimmtes Unternehmen ist auch an den Einsatz beispielhafter Referenzmodelle zu denken. Sie sind vor allem dann besonders erfolgversprechend, wenn sich ein Unternehmen in einer Phase des Umbruchs befindet, das Referenzmodells also nicht unbedingt eine treffende Generalisierung des Unternehmens darstellen muß, sondern als Orientierung für Reorganisationsmaßnahmen dienen kann.

Das folgende Szenario[1] geht von einer solchen Situation aus. Dabei ist der Fokus auf die Situation kleiner und mittlerer Unternehmen gerichtet: Der Bedarf an wirksamer DV-Unterstützung ist erkannt, gleichzeitig gibt es im Unternehmen keine DV-Kompetenz. Das Szenario entstand vor dem Hintergrund eines von der GMD im Jahre 1991 durchgeführten Beratungsprojekts. Es beleuchtet den Fall eines Versicherungsmaklers mit 25 Mitarbeitern und einem Bestand von ca. 12.000 Verträgen. Der Firmeninhaber zeigte bisher wenig Interesse an einem computergestützten Informationssystem, sieht aber nunmehr ein, daß ein solches

1. Eine umfangreichere Darstellung des Szenarios findet sich in Frank/Klein (1992 a, S. 51 ff.)

System in Zukunft unumgänglich ist. Dafür gibt es unter anderem folgende Gründe. So ist es betriebswirtschaftlich angeraten, ein wirksames Controlling zu etablieren. Kunden sollen noch intensiver als bisher betreut werden. Die dazu notwendigen Daten müssen in aktueller Form schnell verfügbar sein. Die Versicherungsgesellschaften drängen auf elektronischen Datenaustausch. Die Einführung eines DV-Systems soll zudem zum Anlaß genommen werden, eine Reorganisation durchzuführen, die auf eine stärkere Kunden- wie auch Erfolgsorientierung zielt.

In dieser Situation würde für den Makler heute ein mehr oder weniger leidvoller Suchprozeß beginnen - vielleicht eingeleitet durch die Empfehlungen anderer Makler oder die Vertriebsaktivitäten eines Anbieters. Die Beurteilung eines Systems - hier ist an die ganze Bandbreite von Software-Technologie bis zur organisatorischen Implementierung zu denken - wird zum Vabanque-Spiel.

Wie könnte ein Unternehmensreferenzmodell - also ein generisches Modell von Makler-Unternehmen - diesen Prozeß erleichtern? Nehmen wir an, der Makler wendet sich an ein Systemhaus, das Informationssysteme für Versicherungsmakler auf der Basis eines bestimmten Referenzmodells anbietet. Gegenstand des Gesprächs mit einem Systemberater ist vor allem die betriebswirtschaftliche Situation des Unternehmens sowie zweckmäßige Organisationsformen. Der Berater startet ein Programm, das diesen Diskurs begleitet. Zunächst werden einige Unternehmensdaten erfaßt: Zahl der Mitarbeiter, Zahl der Vertäge, Vertragsstruktur, Zahl der Kunden etc. Anschließend kann der Makler, unterstützt vom Systemberater, durch das Modell navigieren - auf einer für ihn verständlichen Abstraktionsebene.

Zur näheren Bestimmung der gewünschten Unternehmensphilosophie beginnt der Dialog auf der strategischen Ebene mit der Durchsicht entsprechender Vorschläge. Für jeden Vorschlag wird ein Zielsystem geführt, von dem aus auf eine Liste korrespondierender operationaler Ziele (sie gehören zu der organisatorischen Perspektive) verwiesen wird (Abb. 53).

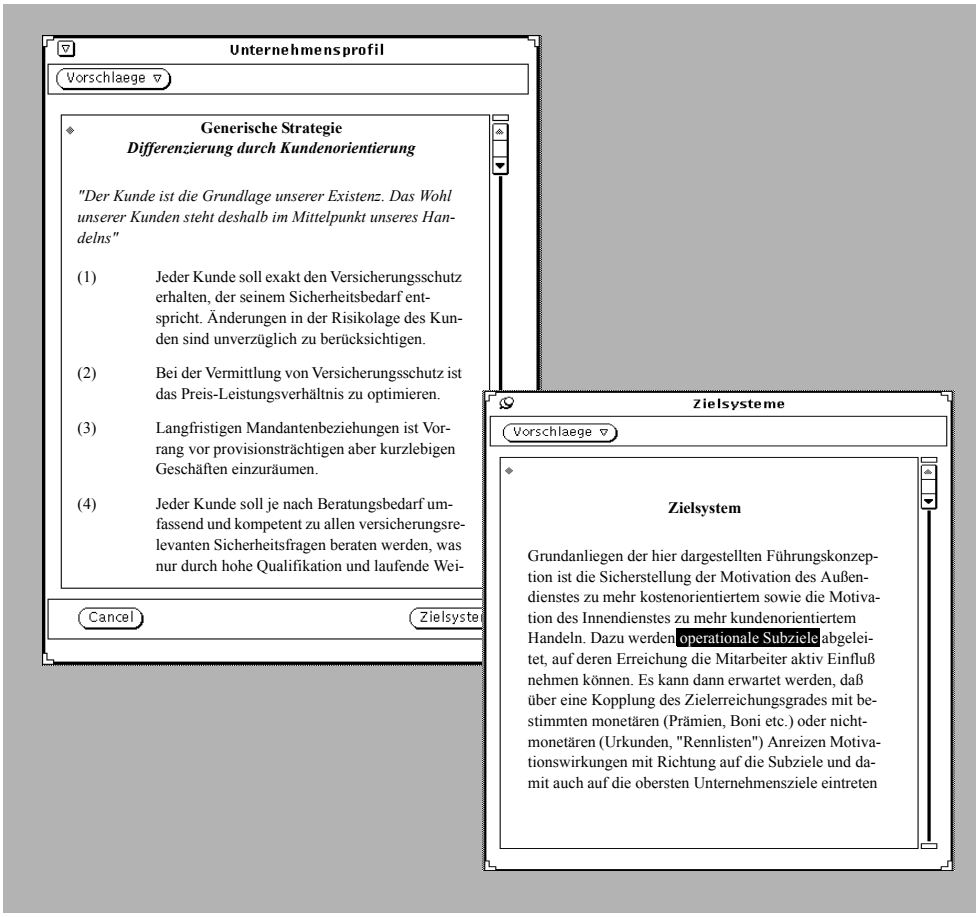


Abb. 53: Darstellung von Teilen des strategischen Modells (Markierte Textbestandteile weisen auf die Existenz einer weiteren Erläuterung).

Die Erläuterung von Alternativen organisatorischer Gestaltung kann einerseits durch die Darstellung beispielhafter Szenen[1], andererseits durch grafische Darstellung von Organisationstrukturen (Abb. 54) erfolgen.

1. In Abbildung 54 ist eine mögliche Argumentationstaktik für die Durchführung von Beratungsgesprächen skizziert, das bei Verfügbarkeit entsprechender Technik durch ein Video ergänzt werden könnte.

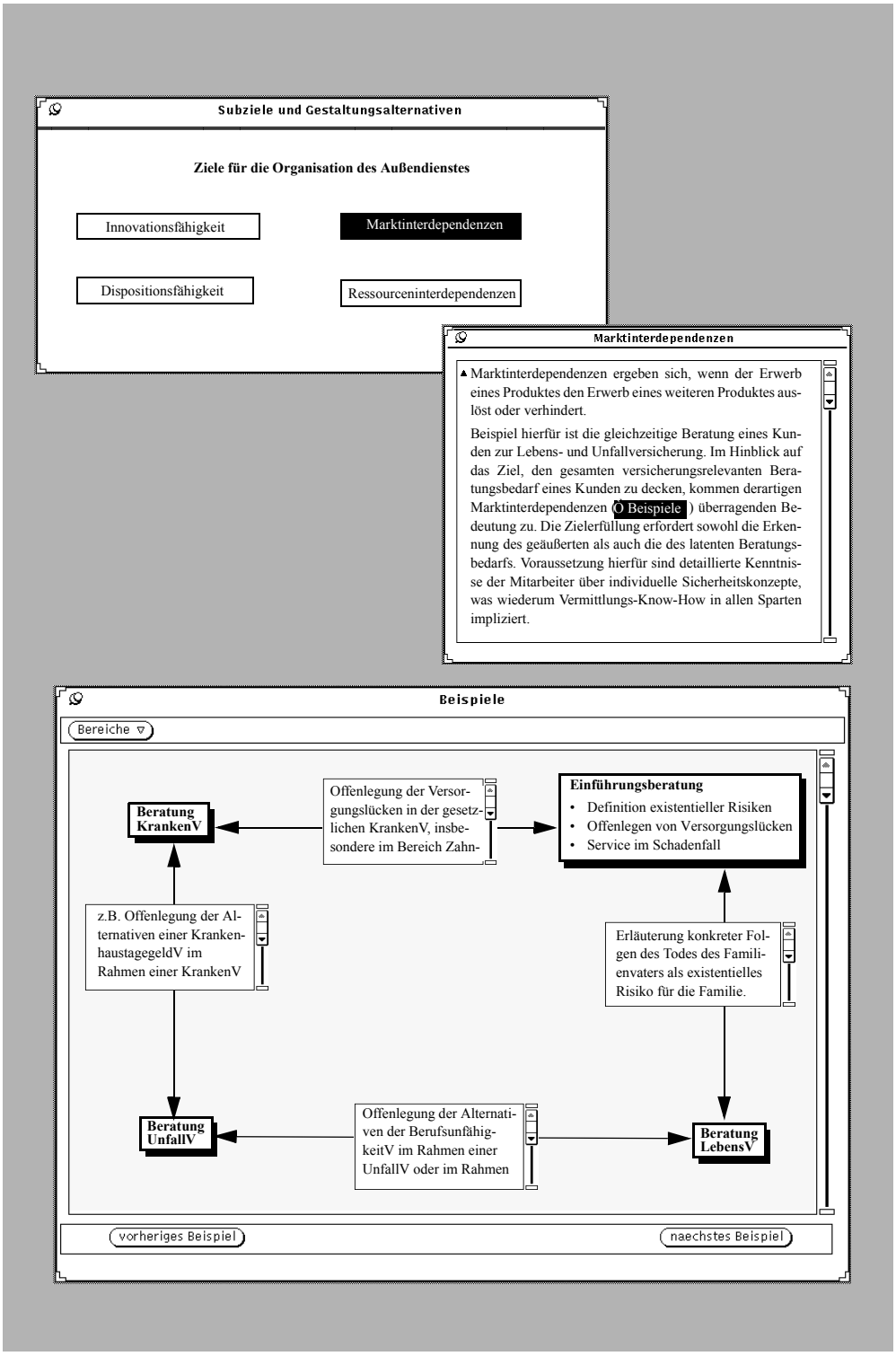


Abb. 54: Operationale Ziele für den Außendienst und beispielhafte Umsetzung.

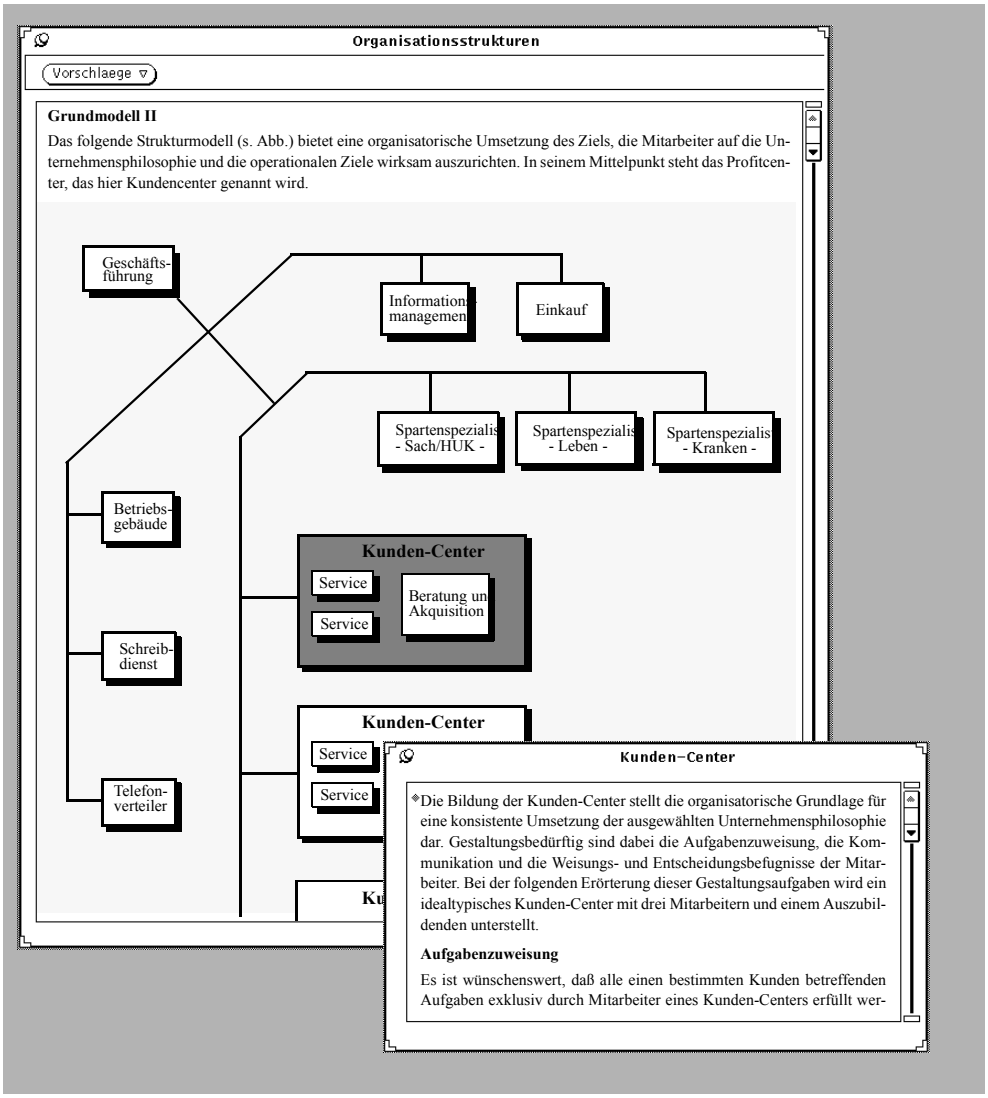


Abb. 55: Darstellung einer Organisationsstruktur zur Umsetzung der ausgewählten Ziele.

Der Dialog zielt darauf, eine geeignete Organisationsform zu identifizieren. Die Thematisierung konkreter Details der dazu passenden Software wird dem Makler erspart. Er hat allerdings die Möglichkeit, sich mit den vorgeschlagenen organisatorischen Alternativen durch geeignete Simulationen vertraut zu machen - etwa in Form von Spreadsheets (vgl. Abb. 56).

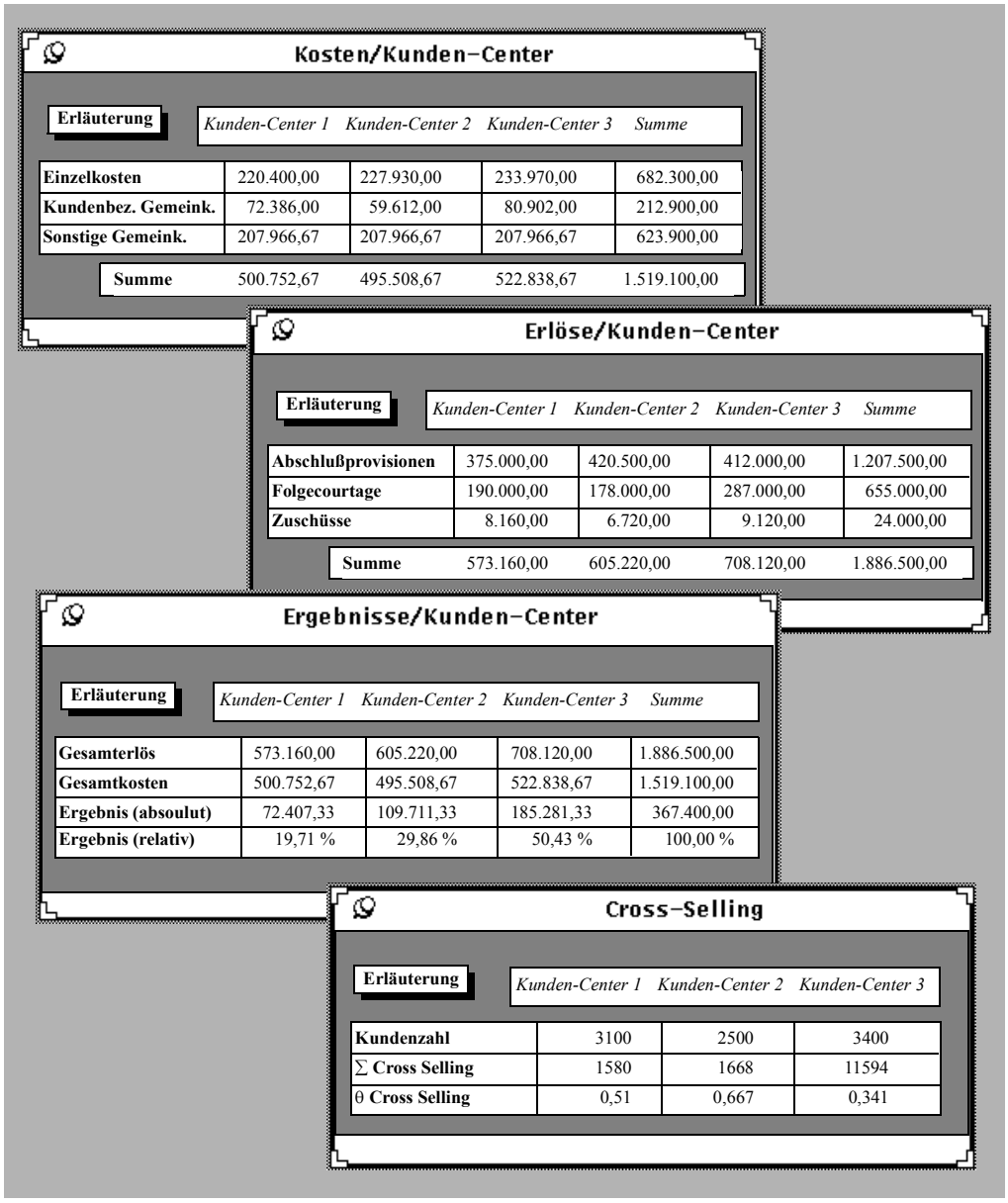


Abb. 56: Veranschaulichung betriebswirtschaftlicher Konsequenzen einer Gestaltungsoption.

Nach der Auswahl einer Organisationsform und damit des zugehörigen Objektmodells wird ein funktionsfähiges Informationssystem durch Instanzierung und Initialisierung mit geeigneten Demonstrationsdaten generiert. Dieser erste Prototyp wird anschließend den zukünftigen Benutzern präsentiert und gegebenenfalls verfeinert. Damit ergibt sich folgendes Ablaufmodell:

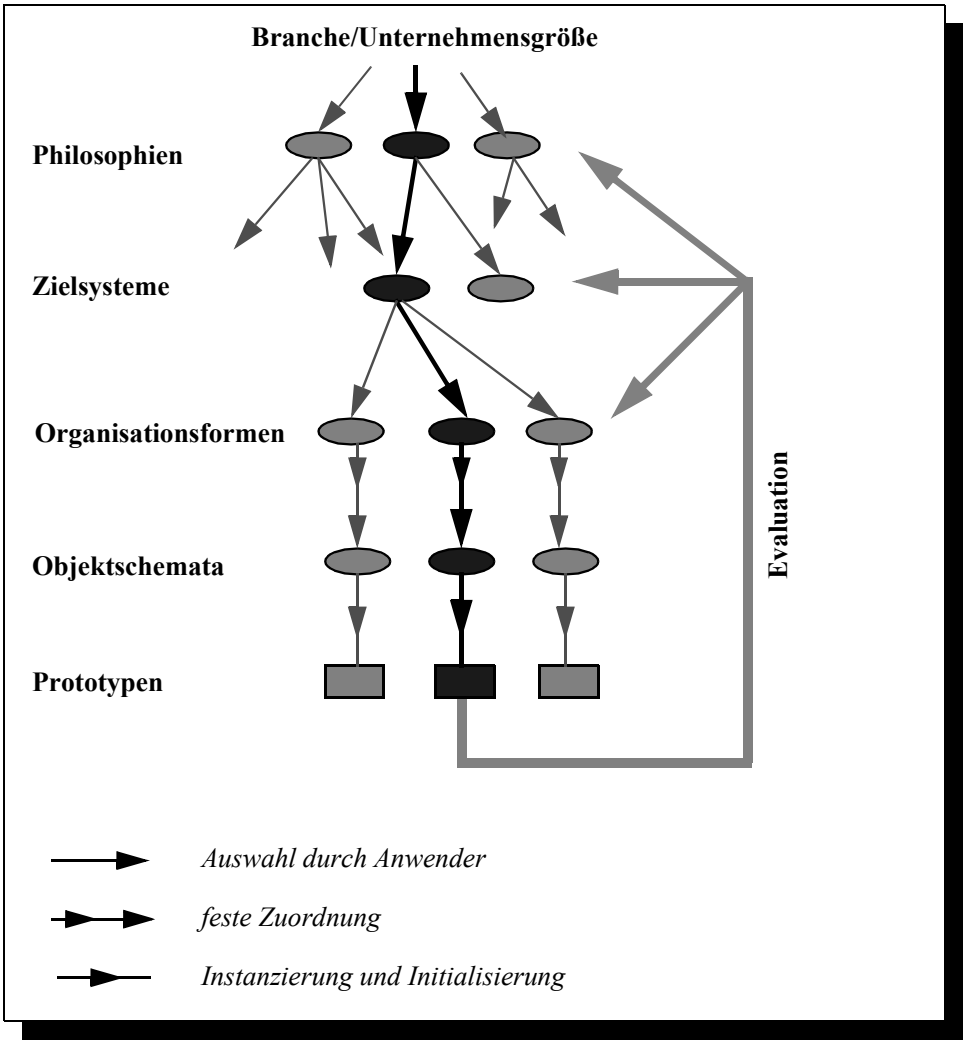


Abb. 57: Zyklus der Auswahl eines Informationssystems aus vorgegebenen Alternativen

Es handelt sich dabei nicht um einen streng sequentiellen Ablauf. Die Konkretisierung einer Abstraktionsstufe kann nicht vollständig aus der einer anderen logisch deduziert werden. Die Zuordnung einer oder mehrerer Konkretisierungen einer Stufe zu der einer anderen erfolgt in der Regel explizit: beim Entwurf und der Pflege des Referenzmodells. Es wird also immer zusätzliche Information eingeführt. Außerdem kann von einzelnen Aspekten einer Ebene beim Übergang auf die nächste abstrahiert werden. Der Kunde muß also nicht unbedingt zunächst das Zielsystem neu wählen, um anschließend eine Organisationsform zu spezifizieren. Nur solche Änderungen, die eine dem Anwender nähere Darstellungsebene betreffen, sollten auch dort ausgeführt werden.

V. Eine Entwicklungsumgebung für den Entwurf von Unternehmensmodellen

"... the aim is to provide tools for the development of descriptions of a world enterprise/slice of reality which correspond directly and naturally to our own conceptualizations of the object of these descriptions."

John Mylopoulos und Hector J. Levesque

"I want a system that is enjoyable to use ... This means tools that reveal their underlying conceptual model and allow for interaction, tools that emphasize comfort, ease, and pleasure of use."

Donald A. Norman

Der Entwurf und die Pflege von Unternehmensmodellen macht den Einsatz von Werkzeugen unumgänglich. So ist ein größeres Modell ohne eine automatisierte Überwachung von Integritätsbedingungen im Zeitverlauf kaum konsistent zu halten. Auch das zielgeleitete Navigieren durch das Modell erfordert eine Werkzeugunterstützung. Wenn zudem schnelles Prototyping ermöglicht werden soll, ist eine computergestützte Entwicklungsumgebung absolut unerläßlich. Einer solchen Entwicklungsumgebung kommt darüber hinaus die Funktion zu, den oben entworfenen Bezugsrahmen in anschaulicher Weise darzustellen und die aktive Auseinandersetzung mit den darin enthaltenen Konzepten anzuregen.

Vor diesem Hintergrund wurde 1990 in der GMD mit der Implementierung verschiedener Komponenten eines Werkzeugs für den Entwurf von Unternehmensmodellen begonnen. Die verschiedenen Entwicklungsstufen sind in einer Reihe von Veröffentlichungen[1] dokumentiert. Die im folgenden beschriebene Version knüpft an diese Vorläufer an, ist jedoch erheblich erweitert und in großen Teilen neu implementiert.

1. Die Komponenten der Entwicklungsumgebung im Überblick

Nachdem sich die ersten Vorstellungen über die Funktionsweise der Entwicklungsumgebung konkretisiert hatten, stellte sich die Frage nach den für die Implementierung zu verwendenden Sprachen und Werkzeugen. Eine Anforderung war dabei conditio sine qua non: Die eingesetzten Werkzeuge sollten die wesentlichen Merkmale der Objektorientierung (Verkapselung, Vererbung,

1. Frank/Klein (1992 b), Frank (1992 a), Frank (1992 b), Frank (1993 a), Frank (1993 b)

Polymorphie) erfüllen - schließlich sollte die Vorteilhaftigkeit objektorientierte Software-Entwicklung auch durch den gewählten Implementierungsansatz demonstriert werden. Weitere wichtige Anforderungen:

- *Hohe Produktivität*: Der Schwerpunkt unserer Aktivitäten sollte auf den Entwurf von Konzepten gerichtet sein, nicht auf umfangreichen Implementierungsarbeiten.
- *Flexibilität*: Da der Bezugsrahmen für die Modellierung in Wechselwirkung mit der Implementierung weiterentwickelt werden sollte, waren weitreichende Änderungen zu erwarten. Die Implementierungswerkzeuge sollten solche Modifikationen unterstützen - etwa durch den (optionalen) Verzicht auf Typisierung.
- *Grafik*: Die anschauliche Darstellung des Modells erfordert grafische Präsentationen sowie die Interaktion mit grafischen Elementen. Deshalb sollte eine wirksame Unterstützung der Implementierung solcher grafischen Benutzerschnittstellen geboten werden.
- *Portierbarkeit*: Es war von Beginn an vorgesehen, die Entwicklungsumgebung in Praxistests zu erproben. Das erfordert die leichte Portierbarkeit auf die wichtigsten in der Praxis eingesetzten Plattformen.

Nach einer mehrmonatigen Evaluationsphase fiel die Wahl schließlich auf eine Implementierungsumgebung, die auf Smalltalk-80[1] basiert. Sie setzt sich aus folgenden Komponenten zusammen:

- Objectworks® 4.0, eine Entwicklungsumgebung für Smalltalk-80 mit einer großen Zahl wiederverwendbarer Klassen.
- ObjectKit®, eine Smalltalk-Klassenbibliothek, die unter anderem Klassen enthält, die es erlauben, Smalltalk-Objekte persistent zu machen.
- ApplicationOrganizer®, eine Klassenbibliothek, die neben erweiterten Browsern eine Versionsverwaltung bietet.
- Objectforms®, ein User-Interface Management System. Es basiert auf dem Model/View/Controller-Konzept und erlaubt (ähnlich wie VisualWorks®) eine Veränderung des Look&Feel (beispielsweise von OpenLook zu Windows) während der Laufzeit.
- OnlineDoc®, eine Klassenbibliothek zur Implementierung von Hypertext-Dokumentationen.
- NEDT®, eine Bibliothek von Klassen, die die Erstellung dedizierter Grafik-Editoren unterstützen.

1. Eine (mittlerweile allerdings nicht mehr aktuelle) Einführung in die Sprache findet sich in Goldberg/Robson (1989). Die im folgenden an einzelnen Stellen verwendeten einschlägigen Begriffe (wie Image, Browser etc.) sind dort erläutert.

Für die prototypische Transformation des mit der Entwicklungsumgebung zu erstellenden Modells ist Smalltalk unmittelbar wenig geeignet, da es keine Sprachmittel zur direkten Abbildung der im Modell beschriebenen Integritätsbedingungen (wie Typisierung, Pre- und Postconditions) bietet. Aus diesem Grund wurde mit dem sogenannten Smalltalk Frame Kit (SFK) eine von Kollegen in der GMD für die Erstellung und Pflege von Thesauri entwickelte Klassenbibliothek verwendet. Sie bietet unter anderem eine Objekt-Definitionssprache, die eine komfortable Beschreibung von Integritätsbedingungen erlaubt (Fischer/Rostek 1991 a).

Objectworks® ist auf einer Reihe von Unix®-Maschinen, sowie auf Apple Macintosh® und unter Windows® verfügbar. Wir haben hauptsächlich Unix®-Systeme (Sun® in Sparc®-Architektur) eingesetzt. Eine Portierung auf andere Plattformen ist in komfortabler Weise möglich: Da Smalltalk eine Kompilierung des Quellcodes in einen maschinenunabhängigen Pseudo-Code vorsieht, kann das auf diese Weise beschriebene sogenannte "Image" (die Gesamtheit der Klassen und Objekte) im Zuge eines Datei-Transfers auf eine andere Plattform gebracht werden. Dort kann das Image dann unmittelbar von dem jeweilgen Smalltalk-Interpreter verarbeitet werden - allenfalls Referenzen auf das Dateisystem müssen eventuell den lokalen Gegebenheiten angepaßt werden. Daneben können kleinere Teile durch das Übertragen (und anschließende Kompilieren) von Quellcode portiert werden.

Die im Laufe von nunmehr zwei Jahren gesammelten Erfahrungen bestätigen die Wahl nachdrücklich:[1] Die Fülle der wiederverwendbaren Klassen und der Komfort von Objektworks führten schon nach kurzer Einarbeitung zu einer hohen Produktivität. Zwar wurde immer noch - entgegen den ursprünglichen Absichten - viel programmiert[2], der Aufwand muß allerdings in Relation zu dem großen Leistungsumfang des Systems gesehen werden. Nicht zuletzt weil Smalltalk keine Typisierung vorsieht, konnten auch umfangreiche Änderungen relativ schnell durchgeführt werden.[3]

Die zu implementierende Entwicklungsumgebung soll die Erstellung und Pflege der den drei Perspektiven entsprechenden Teilmodelle sowie deren Integration zu einem Gesamtmodell unterstützen. Auch wenn die in den drei Teilmodellen ver-

1. Diese Beurteilung ist natürlich in Teilen subjektiv: Die Arbeit mit einem bestimmten softwaretechnischen Ansatz beeinflußt - so oder so - die eigene Sichtweise.
2. Als Indikator dafür mag der Umfang des erstellten Codes dienen. Er beträgt zur Zeit ungefähr 700 KB.
3. Die fehlende Typisierung ist allerdings auch mit einer Reihe von Nachteilen verbunden. Abgesehen von der für unsere Belange nicht im Vordergrund stehenden Laufzeitsicherheit ist dabei an die mangelnde Unterstützung einer zielgerichteten Suchen nach einzelnen Klassen oder Methoden zu denken. Vgl. dazu VI.2.1.2.

wendeten Konzepte jeweils in gleicher Weise - nämlich in Form von Klassen - beschrieben werden, ist doch der Umgang mit den Modellen so unterschiedlich, daß jeweils eine dedizierte Benutzerschnittstelle erforderlich ist. Die Entwicklungsumgebung wurde deshalb in drei Komponenten unterteilt. Bis auf das Werkzeug zur Gestaltung und Pflege von Wertketten ("Value Chain Designer", kurz "VCD") sind sie allerdings nicht exklusiv einer Perspektive zugeordnet. Der "Object Model Designer" ("OMD") dient dem Entwurf von Objektmodellen nach Maßgabe des in IV.2.1 dargestellten Metamodells. Er ist im wesentlichen der Informationssystem-Perspektive zuzuordnen, bietet aber auch die Möglichkeit, organisatorische Sachverhalte zu modellieren und in anschaulicher Weise (etwa als grafische Organigramme) zu präsentieren. Für die Modellierung und Analyse dynamischer Aspekte ist der "Office Procedure Designer" ("OPD") vorgesehen - sowohl auf der Informationssystem-Ebene, als auch auf der Organisations-Ebene. Die konzeptuelle Integration der drei Komponenten erfolgt durch ein gemeinsames Objektmodell. Ein Hypertext-System kann für eine ergänzende Dokumentation des Modells genutzt werden. Die systemtechnische Integration erfolgt dadurch, daß alle Komponenten in einem Smalltalk-Image residieren.

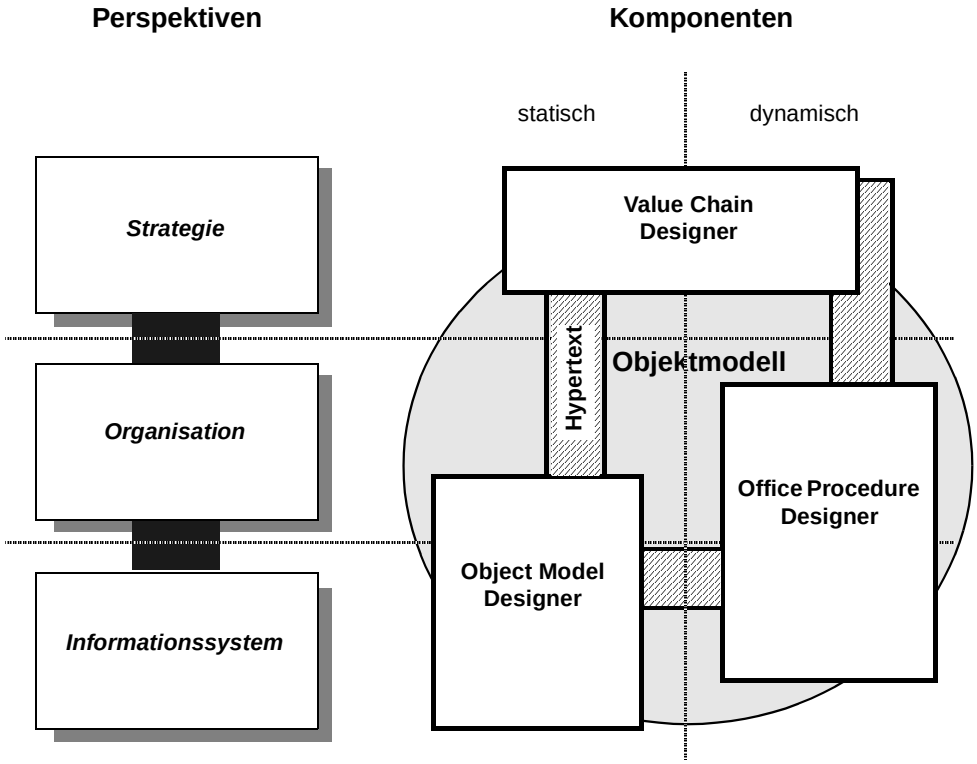


Abb. 58: Die Komponenten der Entwicklungsumgebung

Im folgenden werden die softwaretechnische Realisation und die Benutzung der Komponenten beschrieben. Von einer detaillierten Beschreibung der verwendeten - zum Teil sehr komplexen Klassen - wird allerdings abgesehen: Sie würde den Rahmen der Arbeit sprengen und ist zudem für das Verständnis der Komponenten eher hinderlich. Der weitaus größte Aufwand floß in die Implementierung des Object Model Designers - was sich auch im Umfang der folgenden Darstellungen widerspiegelt

2. Der Object Model Designer

Der OMD unterstützt den komfortablen Entwurf und die konsistente Pflege von Objektmodellen. Darüber hinaus erlaubt er die Präsentation eines Objektmodells in verschiedenen Darstellungsformen und auf unterschiedlichen Abstraktionsstufen. Um die Anschaulichkeit des Modells für den Anwender zu erhöhen, können die im Modell beschriebenen Klassen prototypisch instanziert werden.

2.1 Die Verwaltung von Objektmodellen

Mit der Verwaltung eines Objektmodells sind diverse Formen (referentieller) Integritätsbedingungen verbunden. Der evolutorische Ansatz beim Entwurf eines Modells bringt Änderungen des Modells mit sich, die komfortabel und konsistent durchführbar sein sollten. Als wesentliche - in V.2.1.1 näher dargestellte - Maßnahme zur Berücksichtigung dieser Anforderungen wurde Redundanz durch die konsequente Verwendung von Referenzen minimiert.

Für die Bezeichnung der Modellelemente wurden Konventionen eingeführt, die durch Smalltalk inspiriert sind. Danach wird eine Klasse durch eine Zeichenkette beliebiger Länge, die mit einem Großbuchstaben beginnt, bezeichnet. In gleicher Weise wird ein Attribut bezeichnet, während die Bezeichnung eines Dienstes stets mit einem Kleinbuchstaben beginnt. Die Übergabe von Parametern wird in der Bezeichnung eines Dienstes durch einen Doppelpunkt gekennzeichnet (Beispiel: "erhoeheBestandUm:").

2.1.1 Die Umsetzung des Metamodells

Der Aufbau eines Objektmodells orientiert sich wesentlich an dem oben vorgestellten Metamodell. In einzelnen Teilen wurden die expressiven Möglichkeiten des Metamodells allerdings im Hinblick auf eine komfortable Nutzung eingeschränkt. Dabei handelt es sich neben der Erfassung von Wertebereichen und Diensten vor allem um die Konzeptualisierung von Triggers und Guards. Die skizzierten Ansätze (deren Handhabung in V.2.2 verdeutlicht wird) markieren gleichsam die Grenze zwischen Modellieren und Programmieren. Sie stellen vorläufige Kompromisse dar, die gewiß nicht allen Anforderungen genügen können.

Um die Definition von Bedingungen unter Rückgriff auf Dienste zu unterstützen, bietet der OMD die Möglichkeit, Zugriffsdienste für Attribute und assoziierte Objekte generieren zu lassen. Sie sind mit einem "*" gekennzeichnet (vgl. Abb. 61).

Wertebereiche

Grundsätzlich könnte die Menge der zulässigen Instanzen einer Klasse in beliebiger Weise - etwa durch Auflistung oder formale Festlegungen - beschrieben werden. Im OMD sind zwei alternative Regelungen vorgesehen, zwischen denen sich der Benutzer jeweils entscheiden muß. Im ersten Fall wird zunächst eine Klasse gewählt, deren Wertemenge eine Obermenge der zu definierenden Wertemenge darstellt. Dabei muß es sich in der gegenwärtigen Version des OMD um eine Klasse handeln, die bestimmte Zahlenmengen repräsentiert. Anschließend kann der Bereich durch die Angabe des kleinsten und des größten zulässigen Wertes bestimmt werden. Im zweiten Fall wird die Menge der zulässigen Werte definiert, indem eine Liste dieser Werte angelegt wird. Die Werte werden mit Hilfe von Zeichenketten beschrieben (etwa: [rot, grün, gelb]). Sie haben allerdings nicht die Semantik von Zeichenketten, sondern werden lediglich als Symbole behandelt.

Dienste

Das Metamodell läßt für die Spezifikation von Pre- und Postconditions beliebig komplexe Bedingungen zu. Für die Modellierung könnte dazu eine geeignete Spezifikationssprache verwendet werden. Davon wurde aber zunächst - im Hinblick auf die Ziele Nutzungskomfort und Konsistenz - abgesehen. Stattdessen wurde eine weniger mächtige, stark dialoggestützte Spezifikation der Bedingungen gewählt. Der Benutzer kann dazu für jeden Eingabeparameter des jeweiligen Dienstes eine Bedingung formulieren, der dieser Parameter genügen muß. Die Eigenschaften der Parameter werden durch die Liste der Dienste ihre Klasse verfügbar gemacht. Alternative dazu kann der Parameterwert selbst gewählt werden - was sich bei nicht strukturierten Klassen wie sie etwa für die Abbildung von Zahlen verwendet werden. Durch einen booleschen Operator kann die ausgewählte Eigenschaft mit einem Vergleichswert - entweder einer Eigenschaft der aktuell bearbeiteten Klasse ("Self") oder einer Konstante - zu der Precondition verknüpft werden:

< Wert1 (boolescher Operator, Wert2) >

dabei gilt:

Wert1 := {Parameterklasse, Parameterwert}, (Dienst)

Wert2 := {({Parameterklasse, Self}, Dienst), Konstante}

() : Inhalt nicht notwendig

{}: geklammerte Elemente sind durch exklusives ODER verknüpft

Boolescher Operator und Vergleichswert sind nicht notwendig, wenn der Dienst selbst einen booleschen Wert zurückliefert. Auf diese Weise könnte eine Precondition für einen Dienst "hatGeheiratet", durch den eine entsprechende Beziehung zwischen zwei Objekten der Klasse "Person" etabliert wird, so formuliert werden: < Person geschlecht != Self geschlecht >.

In ähnlicher Weise können Nachbedingungen spezifiziert werden, wobei dazu entweder auf die (anderen) Dienste der jeweiligen Klasse rekurriert werden kann oder auf die Dienste des ggfs. zurückgelieferten Objekts ("ReturnedObject"):

< Wert1 (boolescher Operator, Wert2) >

dabei gilt:

Wert1 := {ReturnedObject, Self}, Dienst

Wert2 := {({ReturnedObject, Self}, Dienst), Konstante}

Also beispielsweise für einen Dienst, durch den ein Lagerbestand verringert wird: < Self bestand >= 0 >.

Trigger

Im Metamodell wird ein Trigger durch ein (bedingtes) Ereignis und eine zugeordnete Aktion beschrieben. Um die mögliche Komplexität eines solchen Konstruktes einzuschränken, werden im OMD lediglich zwei Formen der Spezifikation von Triggers angeboten. Für beide Formen besteht die Festlegung der Aktion darin, den Dienst der jeweiligen Klasse auszuwählen (gegebenenfalls ergänzt um einen Parameter), der im Ereignisfall auszuführen ist. Das auslösende Ereignis wird im ersten Fall unter Rückgriff auf die Frequenz, mit der ein auszuwählender Dienst der Klasse ausgeführt wird, definiert:

< Dienst, boolescher Operator, Grenzhäufigkeit, Zeitintervall >

Auf diese Weise ließe sich etwa folgendes Ereignis beschreiben: "Wenn der Verkaufspreis innerhalb von 30 Tagen mehr als zehnmal geändert wird". Im zweiten Fall wird das Ereignis unter Verweis auf einen Wert definiert, den ein Dienst der Klasse zurückliefert:

< Dienst, (boolescher Operator, Vergleichswert) >

Der Vergleichswert kann dabei als Konstante angegeben werden oder aber selbst durch den Output eines weiteren Dienstes bestimmt werden. Da die durch einen Trigger auszulösende Aktion mitunter darin besteht, irgendein Objekt oder einen Benutzer von dem jeweiligen Ereignis in Kenntnis zu setzen, kann auch jeweils die Aktion "benachrichtige Ereignis-Manager" gewählt werden, deren Implementierung beispielsweise in der obersten Klasse der jeweiligen Generalisierungshierarchie vorgesehen sein könnte.

Die Beschreibung der Trigger dient in der gegenwärtigen Version des OMD vor

allem der Anforderungsanalyse. Damit wird dem Umstand Rechnung getragen, daß für eine vollständige Spezifikation detaillierte und zum Teil komplexe Angaben über die Triggersemantik gemacht werden müssen, die ohne die Verwendung einer formalen Sprache kaum gelingen können. So ist jeweils zu klären, wie die das Ereignis festlegende Bedingung nach der erfolgreichen Ausführung des Triggers zu handhaben ist: Mitunter wird es angemessen sein, sie für den Rest der Lebenszeit des Objekts zu deaktivieren, in anderen Fällen mögen aufwendigere Regelungen sinnvoll sein. Auch wird an dieser Stelle von der für die Implementierung wichtigen Unterscheidung, ob der von einem Dienst gelieferte Wert einen Zustand des Objekts widerspiegelt oder aber unter Rückgriff auf externe Objekte (wie etwa die Zeit) abgeleitet wurde, abstrahiert (vgl. dazu IV.2.1.1.1.3). Entsprechende Angaben können als Kommentar abgelegt werden.

Guard

Aus der Vielfalt und Komplexität der Bedingungen, die zur Definition von Guards verwendet werden können, wird dem Benutzer im OMD nur eine einfache Form angeboten. Er kann dazu unter Rückgriff auf die Dienste (genauer: der zurückgelieferten Werte) der jeweiligen Klasse eine Bedingung formulieren, die immer erfüllt sein muß:

< Dienst, (boolescher Operator, Dienst) >

Auf diese Weise ließe sich also beispielsweise ausdrücken, daß der Einkaufspreis eines Artikels immer kleiner sein muß als der Verkaufspreis:

< einkaufspreis < verkaufspreis >

Es kann nicht davon ausgegangen werden kann, daß die gegenwärtig im OMD angebotenen Möglichkeiten zur Beschreibung der verschiedenen Bedingungen immer zufriedenstellend sein werden. Um einen günstigen Kompromiß zwischen Komplexität und Komfort zu finden, sind die tatsächlichen Anforderungen zu berücksichtigen. Deshalb kann - wenn die vorgesehenen Ausdrucksmöglichkeiten nicht hinreichend erscheinen - ein entsprechender Kommentar angelegt werden. Damit ist die Hoffnung verbunden, daß die Auwertung solcher Kommentare im Zeitverlauf Aufschlüsse über die Gestaltung des genannten Kompromisses liefert.

2.1.2 Der Aufbau eines Objektmodell-Verzeichnisses

Zur Ablage eines Objektmodells kann in Smalltalk auf eine Reihe von Klassen zur Verwaltung von Objekten zurückgegriffen werden. Für den im folgenden beschriebenen Aufbau eines Objektmodell-Verzeichnisses ist die Klasse "Dictionary" von zentraler Bedeutung. Eine Instanz dieser Klasse besteht aus einer beliebigen Zahl von Tupeln, die jeweils aus zwei Objekten (genauer: Referenzen auf dieselben) beliebiger Klasse zusammengesetzt sind. Ein Objekt fungiert

dabei als Schlüssel. Da das durch einen Schlüssel identifizierte Objekt selbst eine Kollektion sein kann, können komplexe Graphen nachgebildet werden. Ein Objektmodell wird denn auch mit Hilfe eines solchen Graphen verwaltet. Er setzt sich aus einer Reihe miteinander verknüpfter Verzeichnisse zusammen (vgl. Abb. 59). Im Hinblick auf die erwähnte Minimierung von Redundanz ist eine Reihe von Verzeichnissen vorgesehen, auf die bei der Spezifikation einer Klasse referiert werden kann:

Verzeichnis der Klassennamen

Jede neu eingeführte Klasse erhält durch den OMD eine eindeutige Identifikation (der dazu verwendete einfache Algorithmus sieht vor, gelöschte Werte neu zu besetzen). In einem zentralen Verzeichnis ist jeder Identifikation ein Name zugeordnet. Wenn im Modell eine Klasse angegeben wird (etwa zur Beschreibung von Attributen oder Parametern), wird dabei immer nur die Identifikation der Klasse verwendet - die dann als Schlüssel dient, um im Verzeichnis der Klassennamen den zugehörigen Namen zu finden. Auf diese Weise sind Änderungen von Namen ohne Probleme möglich.

Verzeichnis von Kategorienamen

Ähnlich wie in Smalltalk können die im OMD verwalteten Klassen (eindeutig) Kategorien zugeordnet werden, die vom Benutzer frei bezeichnet werden können. Bei der Einführung einer neuen Kategorie wird eine Identifikation generiert, unter der ihr Name in einem Verzeichnis abgelegt wird.

Verzeichnis der Beziehungen

In ähnlicher Weise wie Klassen können Beziehungen an verschiedenen Stellen im Modell verwendet werden. In einem zentralen Verzeichnis liegen unter generierten Schlüsseln die Bezeichnung der Beziehung sowie ihre inverse Bezeichnung und ihr Typ ("uses", "used by", "part of", "contains") ab.

Verzeichnis von Ausnahmen

Um eine einheitliche Behandlung von Ausnahmen zu ermöglichen, ist ein zentrales Verzeichnis vorgesehen, in dem unter einem generierten Schlüssel Bezeichnungen von Ausnahmen ("Drucker hat kein Papier", "Massenspeicher voll" etc.) und eine Beschreibung der Ausnahmebehandlung in Form eines freien Textes abliegen.

Im Unterschied zu den genannten Bezeichnungen erfolgt die Benennung von Attributen und Diensten für jede Klasse isoliert - eine notwendige Voraussetzung für Polymorphie. Da aber auch Attribute oder Dienste mehrfach referenziert (etwa bei der Beschreibung einers Triggers oder eines Guards) werden können[1],

1. Bei Attributen beschränkt sich dies mit Rücksicht auf die Verkapselung auf Orte innerhalb der Klasse selbst. In der gegenwärtigen Implementierung kann ein Attribut nur bei der Beschreibung eines Guards referenziert werden.

werden für jede Klasse Verzeichnisse von Attribut- und Dienstnamen angelegt, auf die jeweils mit Hilfe generierter Identifikatoren zugegriffen werden kann.

Neben diesen der Redundanzbeseitigung dienenden Verzeichnissen sieht der OMD einige Verzeichnisse vor, die *kontrollierte* Redundanz mit sich bringen. Sie dienen der Beschleunigung des Zugriffs auf einzelne Objekte. Zu ihnen gehört ein Verzeichnis der Beziehungen, in dem nicht die generierten Identifikationen, sondern die Typen als Schlüssel dienen. Auf diese Weise wird ein schneller Zugriff auf alle Beziehungen eines bestimmten Typs unterstützt. Die gleiche Funktion erfüllen Verzeichnisse, die unter dem Namen (von Klassen, Beziehungen, Kategorien) als Schlüssel einen direkten Zugriff auf die Identifikation erlauben. Daneben gibt es für jede Klasse ein Verzeichnis, in dem angegeben wird, an welchen Stellen im Modell sie referenziert wird. Entsprechende Verzeichnisse existieren auch für die Attribute und Dienste einer jeden Klasse (sie können beispielsweise bei der Beschreibung von Triggers oder Guards referenziert werden) sowie von Triggers und Guards (sie werden von dem oben genannten Verzeichnis der Referenzen auf eine Klasse referenziert). Schließlich wird ein Verzeichnis geführt, in dem über die Kategorie auf die zugehörigen Klassen zugegriffen werden kann. Abbildung 59 zeigt die Struktur des Objektmodell-Verzeichnisses sowie einiger Unterverzeichnisse. Ein Objektmodell kann mit mehreren Vorgangsmodellen integriert sein. Dazu wird im Verzeichnis eine Liste von Vorgangsidentifikationen geführt, die jeweils auf ein Verzeichnis für die Beschreibung eines bestimmten Vorgangstyps (vgl. 3.3) weisen.

Den wesentlichen Teil des Objektmodells bildet ein Verzeichnis der Klassen (nicht der Klassennamen). Es enthält unter jeder Klassenidentifikation eine detaillierte Beschreibung der Klasse. Dazu gehören neben der Oberklasse die bereits erwähnten Verzeichnisse von Dienst- und Attributnamen. Daneben gibt es Verzeichnisse für die Verwaltung der Eigenschaften von Attributen, Diensten, Beziehungen, Triggers und Guards. In Abbildung 60 ist skizziert, wie die Spezifikation einer Klasse in einem Verzeichnis abgelegt wird. Die übrigen Verzeichnisse sind im Anhang dargestellt.

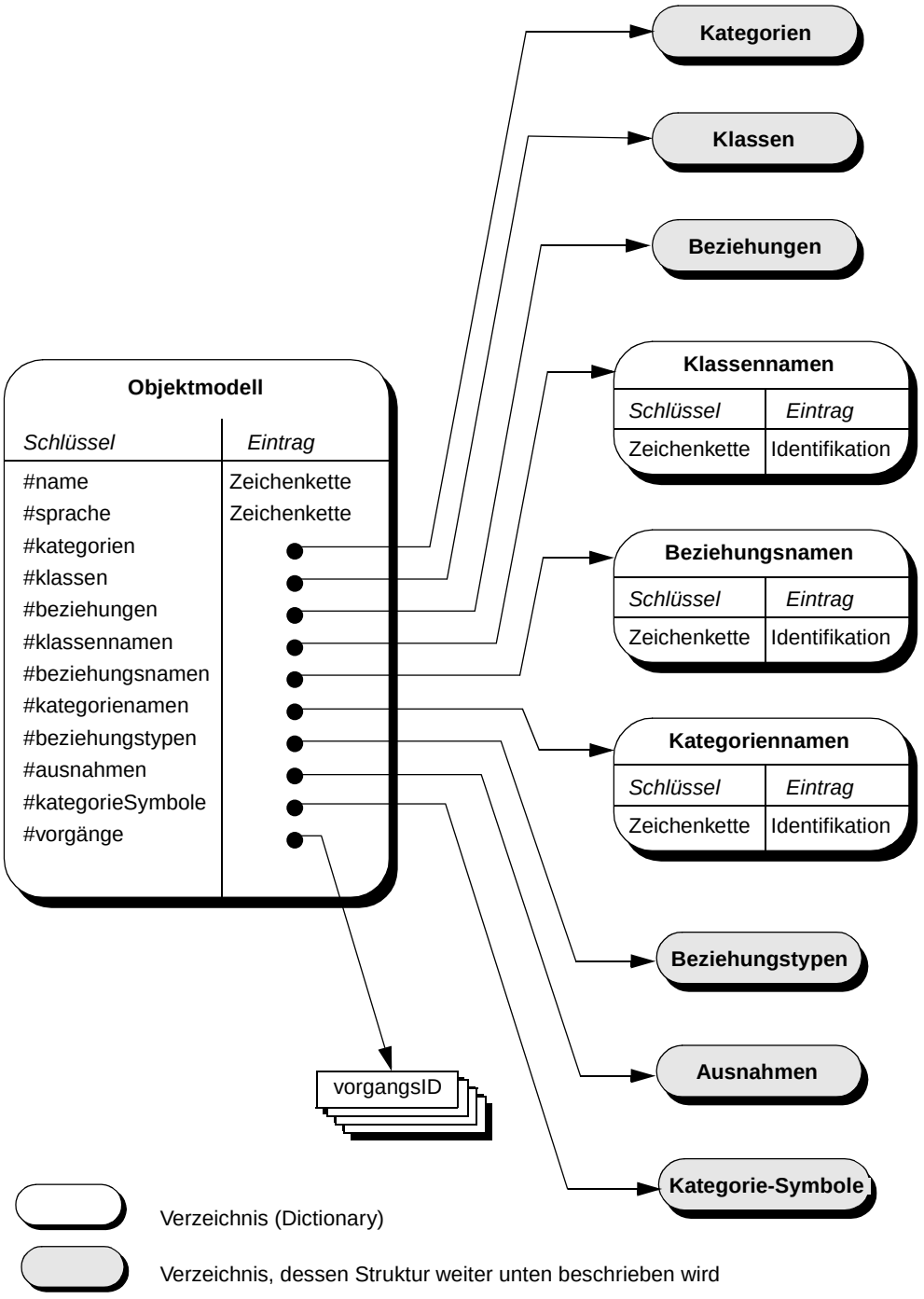


Abb. 59: Einträge in das Objektmodell-Verzeichnis. Schlüssel, die mit einem "#" beginnnen, markieren konstante Bezeichner, es existiert also genau ein Schlüssel dieser Art. Andere Bezeichnungen (z.B. "Zeichenkette") stellen Variablen dar. In diesem Fall sind all jene konkreten Schlüssel möglich, die durch den Wertebereich der Variable abgedeckt sind.

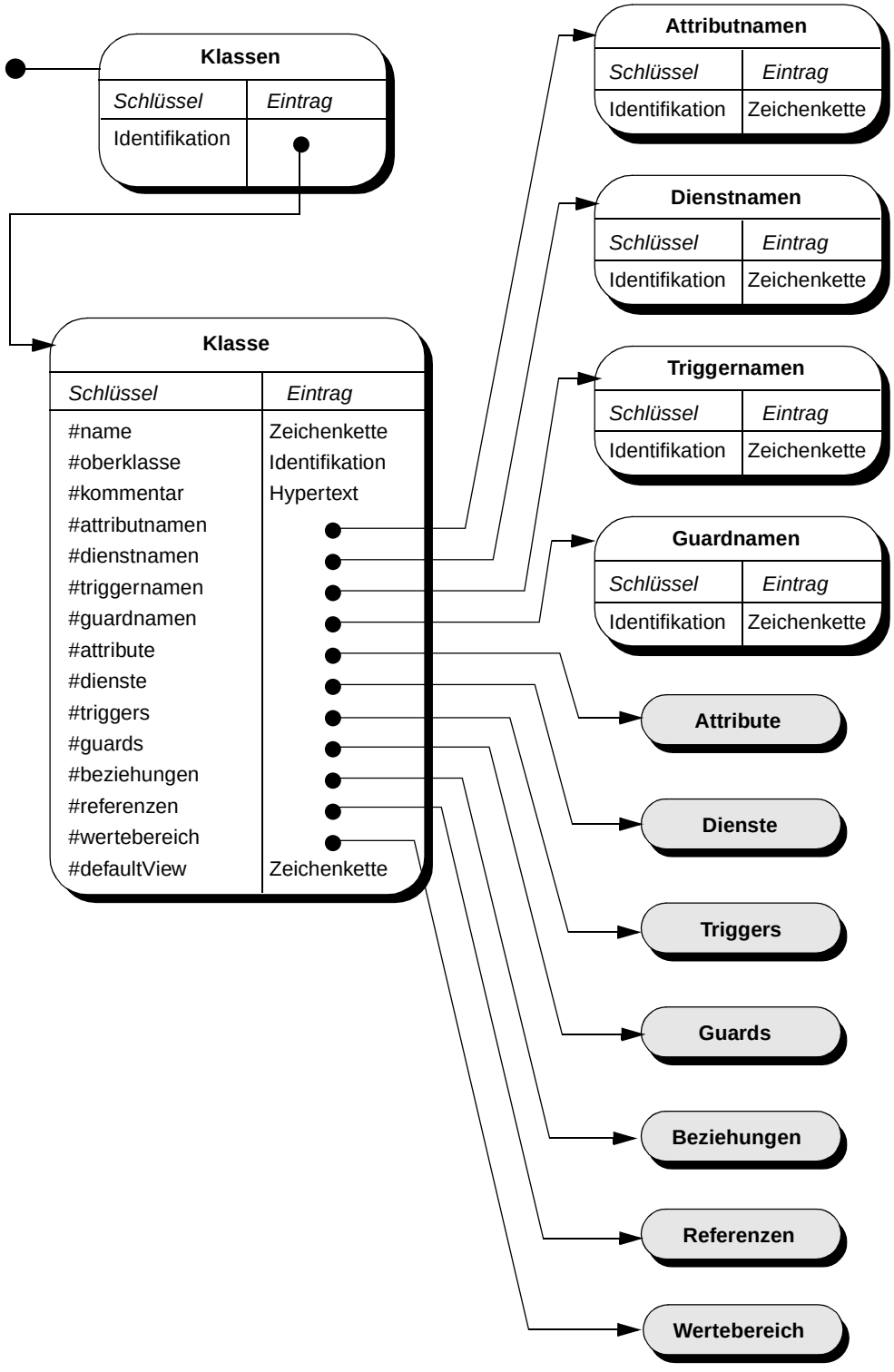


Abb. 60: Struktur des Verzeichnisses einer Klasse

2.1.3 Die Überwachung von Integritätsbedingungen

Die Unterstützung von Änderungen eines Modell ist eine wesentliche Funktion des OMD. Damit sind erhebliche Anforderungen an die Integrität des Modells verbunden, da das hier verwendete Metamodell sehr viel mehr Referenzen zwischen den Modellelementen vorsieht als etwa die traditionelle Datenmodellierung. Im folgenden ist skizziert, wie die wichtigsten Integritätsbedingungen behandelt werden.

Ein Teil der zu erwartenden Modifikationen betrifft die Veränderung von Bezeichnern (für Klassen, Dienste, Beziehungen etc.). Da diese Bezeichner nur an einer Stelle definiert sind, kann sich die Integritätsprüfung des OMD darauf beschränken, festzustellen, ob die neue Zeichenkette den jeweils geltenden Konventionen (beispielsweise für die Benennung von Klassen) entspricht und ob sie nicht bereits verwendet wird. Daraufhin wird nur - unter der unveränderten Identifikation - der Eintrag geändert. Auch wenn die Bezeichnung an anderer Stelle noch als Schlüssel verwendet werden sollte, ist keine weitere Änderung nötig, da es sich bei dem Schlüssel tatsächlich um eine Referenz auf die Bezeichnung handelt.

Demgegenüber ist das Löschen von Elementen mit zum Teil umfangreichen Überprüfungen verbunden. Durch den Rückgriff auf Verzeichnisse, in denen für jedes relevante Element abgelegt ist, wo es referenziert wird, können diese Kontrollen allerdings sehr schnell durchgeführt werden. Grundsätzlich gilt: Jedes Element (Klasse, Attribut, Dienst, Beziehung), das referenziert ist, darf nicht gelöscht werden. In diesem Fall wird dem Benutzer eine Liste der referenzierenden Stellen präsentiert. Daraufhin kann er gegebenenfalls schrittweise die Referenzen löschen, um dann das Element zu löschen.

Der OMD unterstützt einen iterativen Entwurf, in dem der Benutzer nicht gezwungen werden sollte, immerzu konsistente Angaben zu machen. So mag es sinnvoll sein, eine Klasse, auf die an einer anderen, aktuell bearbeiteten Stelle referenziert werden soll, zunächst nur rudimentär anzulegen. Eine formal unvollständig beschriebene Klasse[1] wird allerdings an der Benutzerschnittstelle als solche gekennzeichnet.

Verschiedene Elemente im Modell werden durch Verweis auf eine Klasse beschrieben. Bei einigen solcher Kennzeichnungen sind Integritätsbedingungen zu beachten, die vom OMD kontrolliert werden. So darf es sich bei einer Oberklasse nicht um die Klasse selbst oder eine Unterklasse handeln. Gleiches gilt für die Klassifizierung von Attributen (vgl. IV.2.1.1.1.1). Dabei sind allerdings

1. Eine Klasse gilt als (formal) unvollständig, wenn ihre Oberklasse nicht benannt ist oder aber zu ihren Eigenschaften (Attribute, Dienste etc.) - sofern sie benannt sind - nicht sämtliche notwendigen Angaben gemacht wurden.

zusätzlich noch abstrakte Klassen auszuschließen. Wenn die Oberklasse einer Klasse durch eine andere ersetzt werden soll, ist eine aufwendige Prüfung nötig: Der OMD untersucht die Klasse selbst und alle ihre Unterklassen darauf, ob Referenzen auf von der betroffenen Oberklasse geerbte Merkmale (wie Dienste und Attribute) existieren. Nur dann, wenn es solche Referenzen nicht gibt, erlaubt der OMD eine Veränderung der Generalisierungshierarchie.

Dienste können an verschiedenen Stellen (wie in anderen Diensten, Triggers etc.) referenziert werden. Eine Referenzierung aus einer anderen Klasse (die nicht Unterklasse ist) heraus wird allerdings nur dann gestattet, wenn der Dienst nicht mit dem Zugriffsrecht "private" gekennzeichnet ist. Wenn bei der Spezifikation von Bedingungen Vergleiche angegeben werden, prüft der OMD, ob die verglichenen Werte die gleiche Klasse aufweisen. Falls dies nicht zutrifft, wird der Benutzer gewarnt.

Bei der Erfassung der Standard-Präsentation einer Klasse stellt der OMD sicher, daß der Default View nur dann ein Widget sein darf, wenn die Klasse nicht über mehr als ein Attribut verfügt oder aber explizit ein Instance View definiert ist (vgl. IV.2.1.1.1.4).

Ein Wertebereich kann für eine Klasse nur dann angegeben werden, wenn keine Attribute spezifiziert sind - was auch für gegebenenfalls ererbte Attribute gilt. Umgekehrt können keine Attribute definiert werden, wenn zuvor ein Wertebereich festgelegt wurde. Da für die Spezifikation von Triggers verschiedene, einander ausschließende Möglichkeiten vorgesehen sind, verhindert der OMD, daß mehr als eine Möglichkeit genutzt wird.

2.1.4 Retrieval- und Navigationshilfen

Mit der Größe und Komplexität eines Modells wächst das Risiko, einzelne Komponenten nicht mehr zu finden. Während der Implementierungsarbeiten wurde dieser Umstand mitunter in beängstigender Weise verdeutlicht. Die verwendete Smalltalk-Klassenbibliothek enthält mittlerweile mehr als 2.000 Klassen mit fast 20.000 Methoden. Die Suche nach einer Methode, von deren Existenz man weiß, kann dabei sehr frustrierend werden - ganz zu schweigen von Methoden, deren Existenz man allenfalls vermutet. Die Smalltalk-Entwicklungsumgebung bietet eine Reihe von Verfahren, die die Suche nach Teilen der Klassenbiblithek unterstützen. So kann mit rechtstrunkierten Zeichenketten nach Klassen gesucht werden - was allerdings entsprechende Kenntnisse der in Frage kommenden Bezeichner voraussetzt. Wichtiger sind denn auch Verfahren, die die Analyse von Verweisen unterstützen. So kann man alle Klassen ausgeben lassen, in denen eine bestimmte Methode aufgerufen wird. Daneben ist es möglich, alle Klassen ermitteln zu lassen, in denen eine Methode implementiert ist.

Wie die Erfahrung gezeigt hat, sind diese Verfahren jedoch nicht zufriedenstellend. Im OMD sollte deshalb ein leistungsfähigeres Retrieval ermöglicht werden. Dabei ist zu berücksichtigen, daß die Unzulänglichkeiten der in Smalltalk verfügbaren Verfahren zum Teil darin begründet sind, daß Smalltalk keine Typisierung vorsieht. Es gibt also keine Möglichkeit, umittelbar herauszufinden, von welcher Klasse eine bestimmte Nachricht an ein Objekt einer bestimmten anderen Klasse gesendet wurde - da die Klasse des Empfangsobjekts beim Sender nicht deklariert ist.[1] Diese Schwierigkeiten können allerdings vernachlässigt werden, da im OMD sämtliche Referenzen auf Objekte zu typisieren sind. Deshalb konnten folgende Suchverfahren realisiert werden:

- Die Suche nach allen Klassen, die von einer bestimmten Klasse referenziert werden.
- Die Suche nach allen Klassen, die von einer bestimmten Klasse in einer bestimmten Weise (wie etwa in Attributen, Diensten oder über eine Beziehung) referenziert werden.
- Die Suche nach allen Klassen, von denen eine bestimmte Klasse referenziert wird. Für jede der gefundenen Klasse wird dann auf Wunsch der Ort (Attribut, Dienst etc.) ausgegeben.
- Die Suche nach allen Klassen, von denen eine bestimmte Klasse in einer bestimmten Weise referenziert wird.

Daneben ist es möglich, alle Klassen ausgeben zu lassen, die nicht vollständig beschrieben sind - also gleichsam eine "to-do-list".

Jenseits der fehlenden Typisierung werden in Objectworks® die vorhandenen Retrievalmöglichkeiten nicht umfassend genutzt. So ist keine Volltextsuche möglich. Im OMD ist eine solche Suche vorgesehen. Sie ist in zwei Versionen implementiert. In der einen Version werden nur die Kommentare berücksichtigt. In der anderen Version werden daneben auch alle in einer Klasse verwendeten Bezeichner in die Suche einbezogen. Dabei wird gegenwärtig der gesamte Text (sowie gegebenenfalls alle Bezeichner) invertiert. Es wird keine Stoppwortliste angelegt. Somit kann nach beliebigen Zeichenketten (die rechtstrunkiert sein können) gesucht werden. Außerdem können mit Hilfe der logischen Verknüpfungen "und", "oder" oder "und nicht" komplexere Anfragen formuliert werden. Eine einfache Beschlagwortung kann innerhalb der Kommentare vorgenommen werden. Es ist allerdings gegenwärtig nicht vorgesehen, die Schlagworte als solche zu kennzeichnen.

Der mit Smalltalk verbundene Browser ist immer noch eine beispielhafte Navi-

1. Dabei soll nicht verschwiegen werden, daß die mit der fehlenden Typisierung verbundene Einschränkung von Suchmöglichkeiten in Smalltalk zum Teil durch einen sehr komfortablen Debugger aufgehoben wird.

gationshilfe für Klassenbibliotheken. Das gilt vor allem für den im Application-Organizer® enthaltenen "Inheritance-Browser", der es erlaubt, geerbte Methoden bis zu einem wählbaren Niveau anzeigen zu lassen. Dieser Browser hat den Browser des OMD wesentlich inspiriert. Auch er erlaubt die Zuordnung von Klassen zu Kategorien und das Traversieren durch die Generalisierungshierarchie. Der konkrete Aufbau ist in V.2.2.1 beschrieben. Alternativ zum Browser bietet der OMD eine weitere Navigationshilfe: Generalisierungshierarchien und Beziehungen können grafisch dargestellt werden. Auf diese Weise können andere und häufig größere Mengen von Klassen im Zusammenhang (auf Wunsch können alle Klassen eines Modells in einer Grafik dargestellt werden) betrachtet werden als es der Browser erlaubt (vgl. V.2.2.2).

2.2 Die Benutzerschnittstelle

Der OMD soll einen komfortablen - und das heißt nicht zuletzt: anschaulichen - Umgang mit komplexen Objektmodellen erlauben. Die Gestaltung der Benutzerschnittstelle ist dabei von zentraler Bedeutung. Sie prägt nicht nur die Sicht auf das Modell, von ihr hängt auch wesentlich die Benutzbarkeit des OMD, also seine Eignung als Werkzeug[1] ab.

Beim Entwurf der Benutzerschnittstelle standen die folgenden Anforderungen im Vordergrund:

- *Möglichst deutliche Anlehnung an das Metamodell*. Es handelt sich nicht um ein Werkzeug für Laien, sondern für Fachleute, die mit den Grundzügen des Metamodells vertraut sein sollten. Beim Modellieren sollte das Metamodell als gedankliche Orientierung gleichsam im Hintergrund präsent sein.
- *Verschiedene Abstraktions- oder Aggregationsstufen*. In verschiedenen Phasen der Modellierung ist der Blickwinkel nicht immer gleich: Mitunter interessieren lediglich Klassen und zwischen ihnen bestehende Beziehungen, in anderen Kontexten ist eine detaillierte Betrachtung wichtiger. Der Benutzer sollte in komfortabler Weise zwischen den verschiedenen Betrachtungsebenen wechseln können.
- *Anschauliche Präsentationsformen*. Dabei ist nicht nur an die unmittelbaren Benutzer zu denken, sondern auch an Betrachter (wie etwa Domänenexperten), mit denen der Benutzer das Modell diskutiert. Die Umsetzung dieser Anforderung artikuliert sich in grafischen Darstellungen, aber auch in der Möglichkeit, Klassen zu instanzieren, um über prototypische Instanzen reden

1. Eine ontologische Reflektion des Werkzeugbegriffs findet sich in Budde/Züllighoven (1990 S. 82 ff.). Sie verwenden dabei Heideggersche Kategorien wie "Zuhandenheit" und "Geworfenheit" zur Durchdringung der Nutzungsbedingungen von Software-Werkzeugen.

zu können.

Der Portierbarkeit des Werkzeugs wurde zwar große Bedeutung beigemessen. Angesichts der Anforderungen an die Benutzerschnittstelle wurden hier jedoch gewisse Abstriche gemacht: Die Darstellung der verschiedenen Fenster setzt einen hochauflösenden Grafik-Bildschirm mit einer Diagonale von mindestens 17 Zoll voraus. Wie auch bei den beiden anderen Komponenten ist die Sprache der Benutzerschnittstelle während der Laufzeit änderbar. Diese Internationalisierung wird durch die Verwendung eines Wörterbuchs erreicht, in dem symbolischen Bezeichnern die entsprechenden Texte beliebig vieler Landessprachen zugeordnet werden können. Zur Zeit ist jeweils eine deutsche und eine englische Fassung abgelegt. Die Benutzerschnittstelle wurde mit Hilfe von ObjectForms® implementiert, wodurch es möglich ist, zwischen fünf verschiedenen Look&Feel-Optionen[1] dynamisch zu wählen.

2.2.1 Die Erfassung und Präsentation von Klasseneigenschaften

Für die textuelle Darstellung des Modells sind zwei Abstraktionsebenen vorgesehen. Im zentralen Browser werden Kategorien und Klassen gelistet. Daneben wird für die ausgewählte Klasse angezeigt, welches ihre Oberklasse ist und ob sie vollständig beschrieben ist. Vor allem aber werden die Bezeichner ihrer Eigenschaften (Attribute, Dienste, Triggers, Guards, Beziehungen) gelistet. Dabei kann ausgewählt werden bis zu welcher Oberklasse die geerbten Eigenschaften angezeigt werden sollen. Auf der zweiten Abstraktionsebene können diese Eigenschaften - jeweils in einem dedizierten Fenster - näher beschrieben werden. Da der Bildschirm nicht hinreicht, um sämtliche fünf Fenster nebeneinander zu plazieren, sind die Fenster übereinander gelegt. Um ein bestimmtes Fenster in den Vordergrund zu holen, muß lediglich die Liste der entsprechenden Eigenschaft mit dem Maus-Zeiger berührt werden.

An mehreren Stellen im Modell werden Klassennamen zur Kennzeichnung von Eigenschaften benötigt (bei Attributen, Parameter etc.). Um der Eingabe überflüssiger oder versehentlich falsch eingegebener Klassenbezeichner entgegenzuwirken, ist der Benutzer angehalten, solche Bezeichner nicht manuell einzugeben, sondern sie aus einem zentralen Verzeichnis zu transferieren.

Abbildung 61 zeigt den zentralen Browser und - um die Verbindung zu der zweiten Abstraktionsebene deutlich zu machen - die verkleinerten Eigenschaftsfenster. Geerbte Eigenschaften sind im zentralen Browser mit einem ">" markiert. Daneben sind solche Dienste besonders gekennzeichnet, die vom OMD generiert und auf Wunsch gezeigt werden. Dabei handelt es sich um Dienste für den

1. Dabei handelt es sich um OpenLook, Motif, Macintosh, Windows und Presentation Manager.

Zugriff auf Attribute (gekennzeichnet mit "*") und assoziierte Objekte (gekennzeichnet mit "@"). Wenn zu einer Klasse oder einer Eigenschaft ein Kommentar existiert, ist der Button zum Öffnen des Kommentar-Fensters als geöffnetes Buch, ansonsten als geschlossenes Buch dargestellt.

Die Beschreibung der verschiedenen Eigenschaften ist unterschiedlich umfangreich. Die Fenster erscheinen dennoch alle in der gleichen Größe (die allerdings verändert werden kann), ihr Inhalt kann gerollt werden. Abbildung 61 zeigt den Inhalt der Fenster für Attribute und Beziehungen. Bei der Beschreibung eines Attributs werden - nach dem Eintrag der Klasse - die Dienste, die von der Klasse des Attributs bereitgestellt werden, in der Listbox unten links gezeigt. Die Dienste, die in die aktuell beschriebene Klasse übernommen werden sollen, können dann einfach ausgewählt werden.

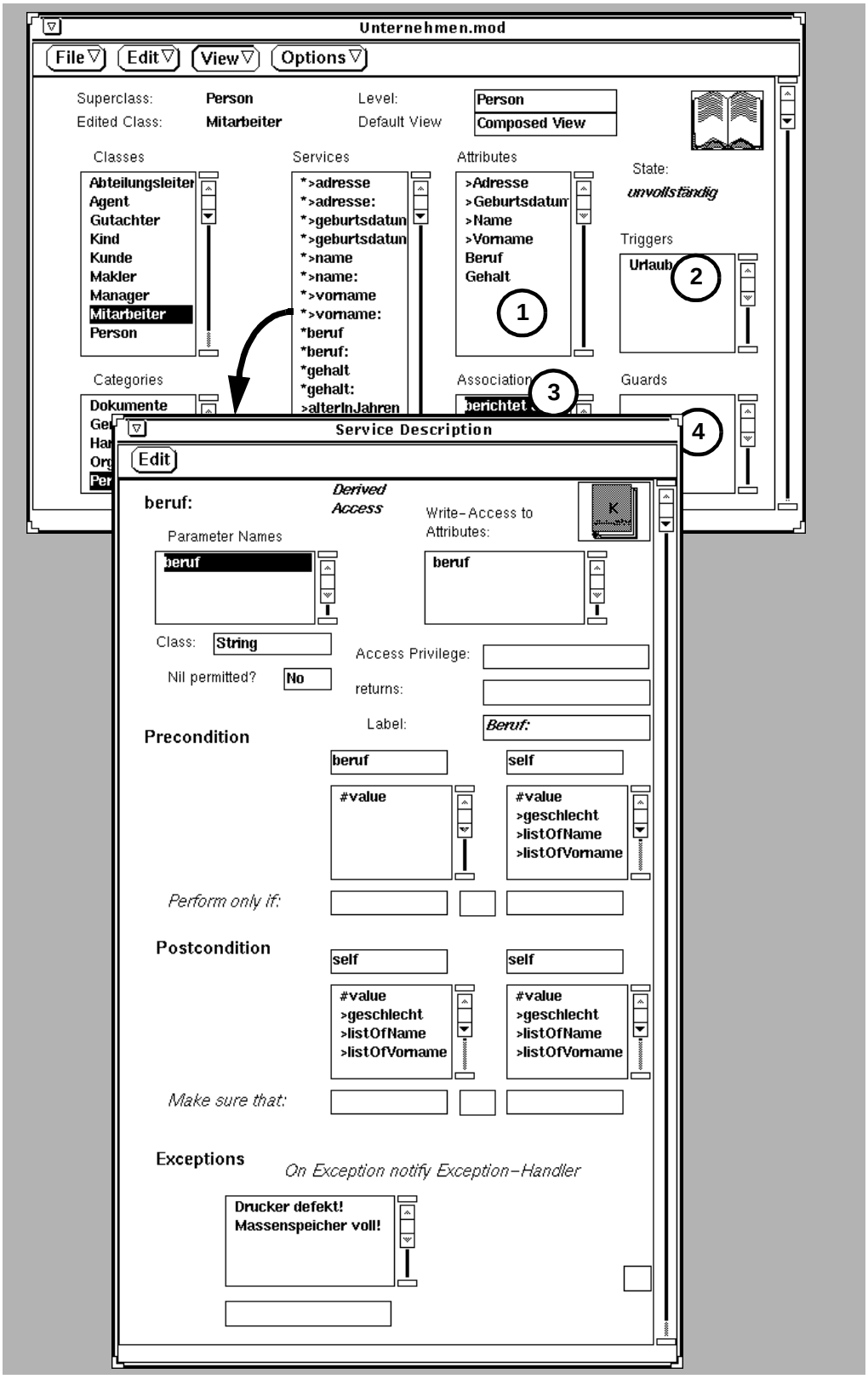


Abb. 61: Die Abstraktionsebenen der textuellen Beschreibung

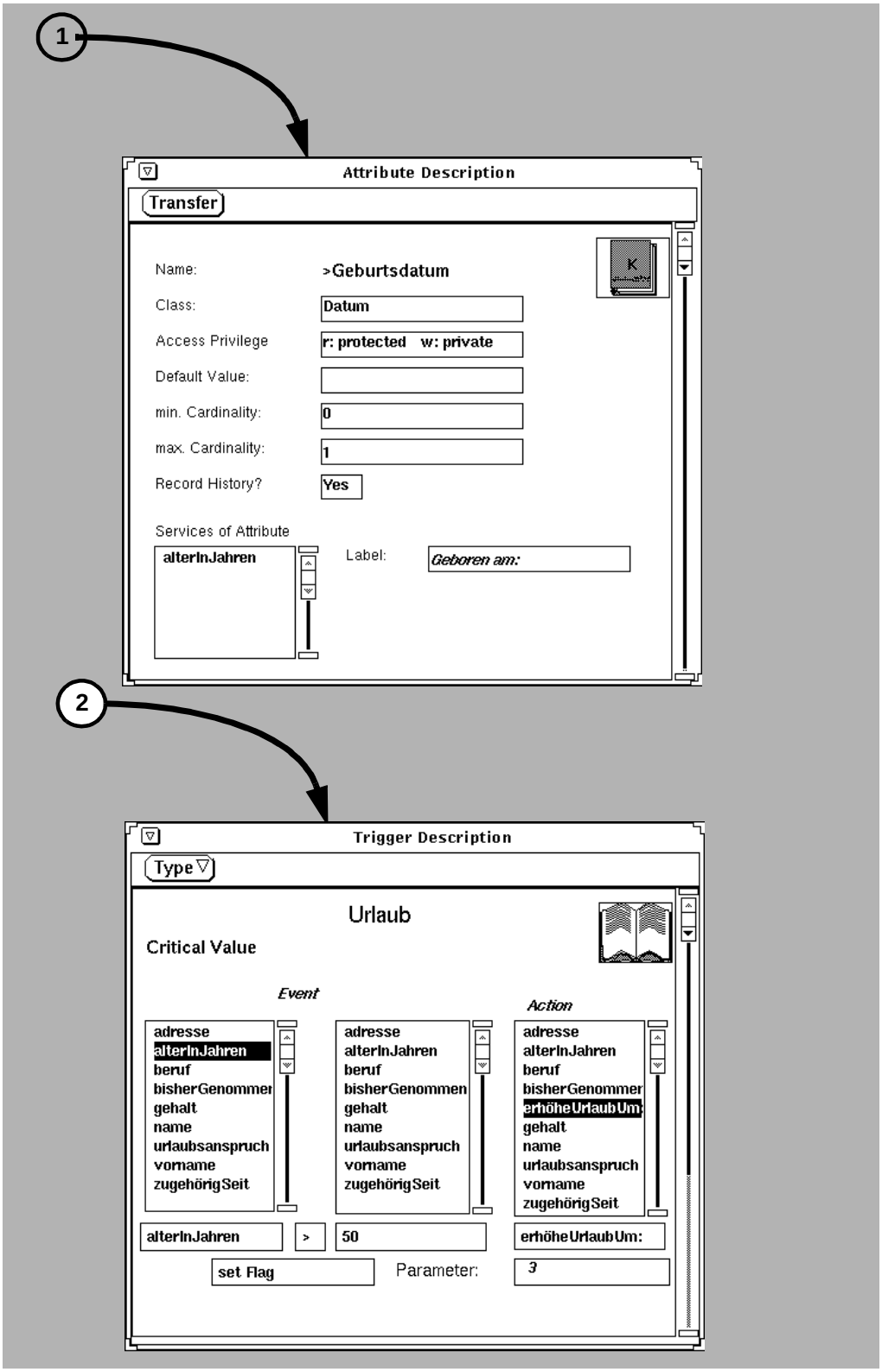
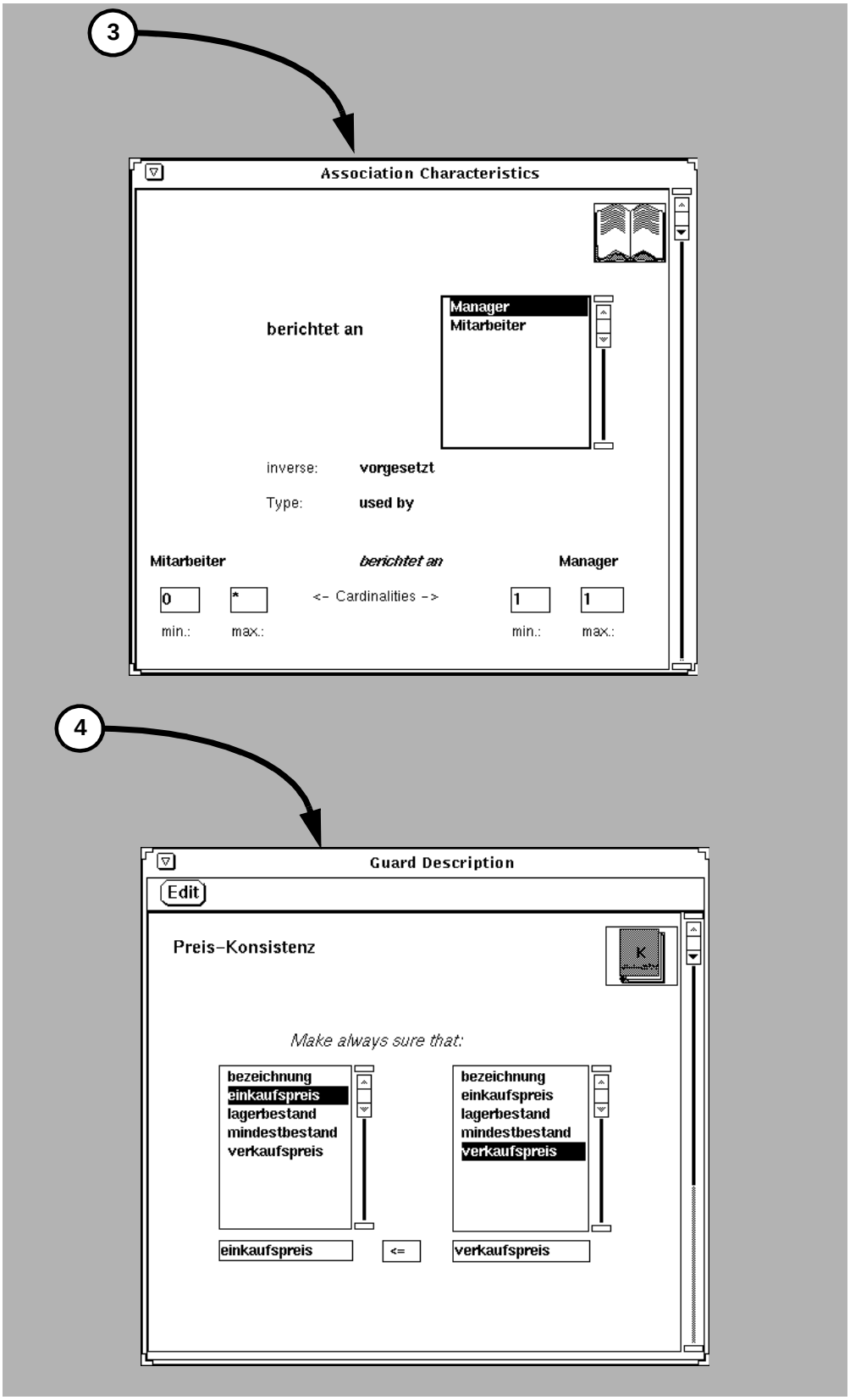


Abb. 62: Fenster zur Charakterisierung der Klasseneigenschaften (Forts. folgende Seite)


3
Association Characteristics
berichtet an
Manager
Mitarbeiter
inverse:
vorgesetzt
Type:
used by
Mitarbeiter
berichtet an
Manager
0
*
<- Cardinalities ->
1
1
min.:
max.:
min.:
max.:
4
Guard Description
Edit
Preis-Konsistenz
K
Make always sure that:
bezeichnung
einkaufspreis
lagerbestand
mindestbestand
verkaufspreis
bezeichnung
einkaufspreis
lagerbestand
mindestbestand
verkaufspreis
einkaufspreis
<=
verkaufspreis

2.2.2 Grafische Darstellungs- und Interaktionsformen

Zur übersichtlichen Präsentation von Beziehungen zwischen Klassen oder Objekten werden häufig grafische Darstellungen gewählt. Im OMD können neben der Generalisierungshierarchie Beziehungen auf Objektebene grafisch visualisiert werden. Eine Klasse wird dabei durch ein Maus-sensitives Icon repräsentiert, das mit dem jeweiligen Klassennamen beschriftet ist. Ursprünglich war vorgesehen, Generalisierungshierarchien oder auch Beziehungsnetze interaktiv vom Benutzer erstellen zu lassen. Nachdem dieser Ansatz, der immer noch verfügbar ist, implementiert war, stellte sich jedoch heraus, daß es sinnvoll ist, alternativ dazu die grafischen Darstellungen aus den textuellen Angaben generieren zu lassen. Damit wird dem Umstand Rechnung getragen, daß der Benutzer in der Regel - das gilt vor allem für Beziehungsgeflechte - nicht in der Lage ist, eine übersichtliche Darstellung zu produzieren.

Die Generalisierungshierarchie wird bei größeren Modellen so umfangreich, daß sie nicht mehr vollständig in einem Fenster sichtbar ist - selbst wenn das Fenster auf die volle Bildschirmgröße expandiert wurde. Es ist deshalb möglich, eine Teilhierarchie erzeugen zu lassen. Dazu wird eine bestimmte Klasse ausgewählt, deren Darstellung dann mit denen ihrer Ober- und Unterklassen verknüpft wird. Eine solche Teilhierarchie kann interaktiv ausgeweitet werden, indem ein ausgewählter Knoten expandiert wird, das heißt, es werden seine Unterklassen dargestellt.

Abb. 63: Grafische Symbole zur Darstellung von Beziehungstypen (jeweils für von links nach rechts gerichtete Beziehungen)

Beziehungen werden in Anlehnung an semantische Netze visualisiert. Dazu werden die Klassensymbole mit gerichteten Kanten verknüpft, die mit einem Symbol für den jeweiligen Beziehungstyp (Interaktions- oder Aggregationstyp) versehen sind (vgl. Abb. 63). Außerdem werden an die Kanten die Kardinalitäten angetragen. Die Symbole für die Beziehungstypen sind mit dem Namen der Beziehung beschriftet. Sie können mit der Maus selektiert werden. Eine selektierte Beziehung kann invertiert werden. Dazu wird die Richtung der Kante umgedreht und die Beschriftung durch die Bezeichnung der inversen Beziehung (beispielsweise "beinhaltet" durch "ist Teil von"). Angesichts der möglichen Komplexität von

Beziehungsgeflechten und der damit verbundenen Gefährdung der Übersichtlichkeit beschränkt sich die Darstellung auf jeweils eine Klasse und die für sie (genauer: für die Objekte, die sie repräsentiert) eingetragenen Beziehungen. Die grafischen Darstellungen von Klassenhierarchie und Beziehungsnetzen können auch verwendet werden, um durch das Modell zu navigieren.

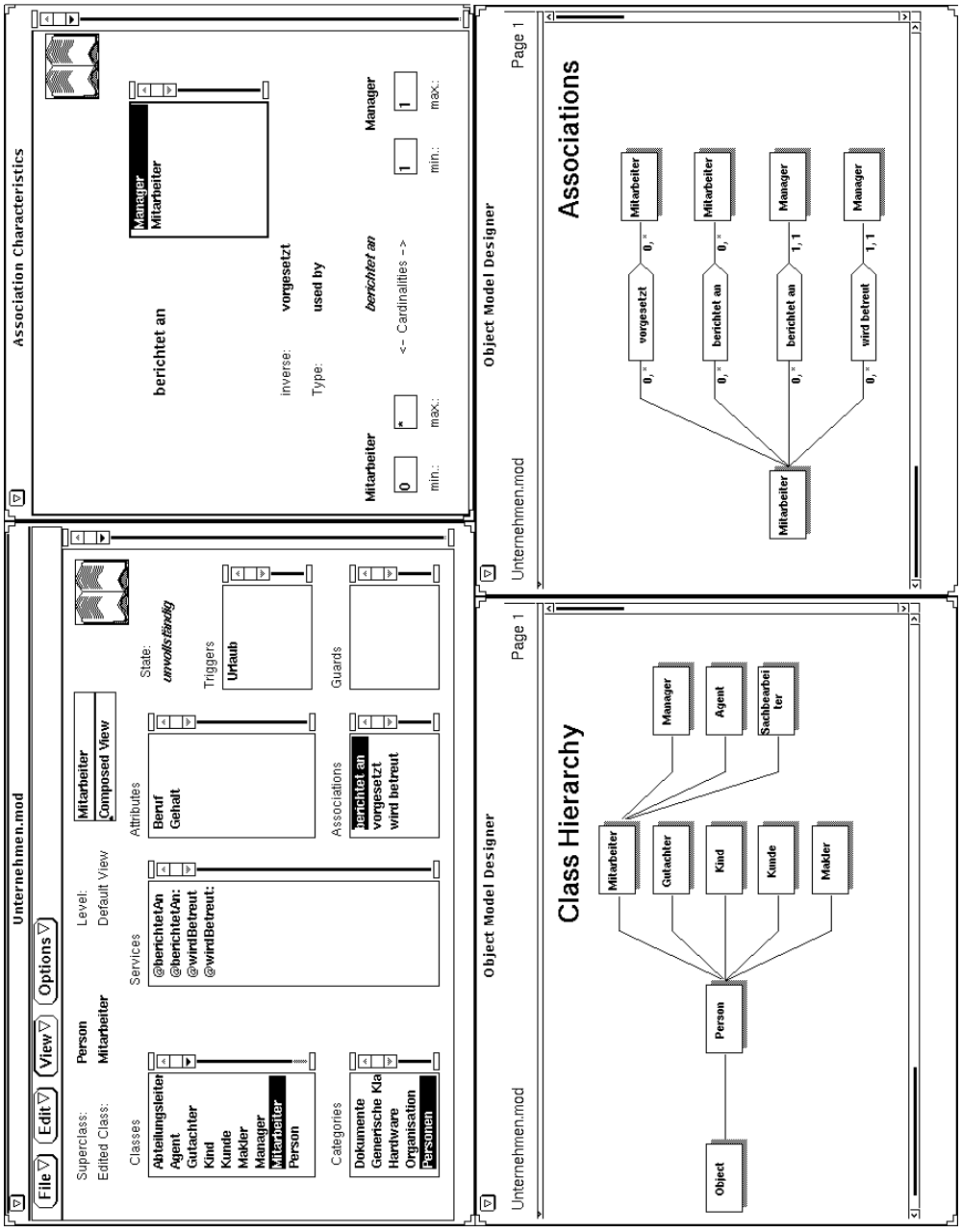


Abb. 64: Die gemeinsame Darstellung der verschiedenen Sichten des OMD

Dazu sind die einzelnen Grafik-Editoren mit dem zentralen Browser verbunden. Wann immer in einem der Editoren eine Klasse mit der Maus aktiviert wird, wird im Browser die entsprechende Klasse ausgewählt. Man kann also über die grafische Darstellung in die Klassen "zoomen". Abbildung 64 zeigt, wie die verschiedenen Darstellungsformen des OMD gemeinsam präsentiert werden können - wobei die Fenster für die Eigenschaften einer Klasse übereinanderliegen.

2.3 Prototypische Implementierung und Instanzierung

Der OMD ist in der gegenwärtigen Version vor allem für den Entwurf von Objektmodellen gedacht. Es ist allerdings auch eine rudimentäre Transformation des Modells in ausführbaren Code vorgesehen - schließlich ist eine solche Transformation von erheblicher Bedeutung. Dies gilt natürlich vor allem für die wünschenswerte Überwindung der Friktionen zwischen den verschiedenen Phasen der Software-Entwicklung. Daneben hat eine Implementierung des Modells oder von Teilen des Modells auch unmittelbare Bedeutung für den Modellentwurf selbst: Bei der Erörterung des Modells mit Anwendern kann es sich als hilfreich erweisen, an der einen oder anderen Stelle von der konzeptuellen Ebene auf eine Ebene zu wechseln, auf der konkrete Instanzen dargestellt werden.

Idealtypisch würde eine solche Transformation vorsehen, aus einem Objektmodell mit Hilfe einer geeigneten Daten-Definitionssprache (besser: Objekt-Definitionssprache) ein Schema für ein Objektverwaltungssystem zu generieren. Ein solcher Ansatz ist allerdings zur Zeit mit Nachteilen verbunden. So sind die von objektorientierten Datenbankmanagement-Systemen angebotenen Klassenschemata (auf die bei der Implementierung zu referieren wäre) noch nicht standardisiert. Man wäre also auf ein System beschränkt. Dieser Umstand wiegt umso schwerer als die Implementierung einer weitgehend automatisierten Transformation mit einem erheblichen Aufwand verbunden wäre. Zudem müßte man dabei Verbindung zwischen Smalltalk und dem jeweiligen Datenbankmanagement-System schaffen - und damit die mit einem Smalltalk-Image verbundenen Integrationsvorteile aufgeben. Eine unmittelbare Umsetzung in Smalltalk ist aus bereits genannten Gründen (vgl. V.1) allerdings auch nicht angeraten.

Im OMD wurde deshalb ein anderer Ansatz gewählt. Diese Entscheidung ist im wesentlichen auf eine glückliche Fügung zurückzuführen: Bei internen Recherchen anläßlich einiger Probleme mit Smalltalk stellte sich heraus, daß Kollegen in einem anderen Institut der GMD ein System entwickelt hatten, das sich bei näherer Hinsicht als sehr gut geeignet erwies. Es handelt sich dabei um ein Frame-basiertes System (SFK, vgl. V.1), das in Smalltalk implementiert ist. Fischer/Rostek (1991 b) haben es für den Entwurf und die Pflege von Thesauri konzipiert. Dem Anwender wird dazu eine Objekt-Definitionssprache angeboten, mit der er die zwischen den Begriffen eines Thesaurus bestehenden Beziehungen

und die daran anknüpfenden Integritätsbedingungen beschreiben kann. Zur Beschreibung werden Frames verwendet, deren Slots zur Aufnahme von Attributen oder Diensten dienen. Attribute werden, wie im OMD, durch eine Klasse typisiert. Entscheidend ist nun, daß die konzeptuelle Beschreibung der Frames in ausführbaren Code kompiliert werden kann. Dabei werden aktive Objekte, sogenannte "Demons" erzeugt, die die Überwachung der Integritätsbedingungen garantieren.

```
initializeSlotDescriptions
   "Kasko-Vertrag"

(self slot: #Schadenfreiheitsrabatt)
   range: Bonus;
   maxCardinality: 1;
   beforeAdd: [:vertrag :bonus :actBonus :transact|
   (actBonus < vertrag maxBonus)].
(self slot: #Abschlußdatum)
   range: Datum;
   minCardinality: 1;
   maxCardinality: 1.
(self slot: #VereinbarteZahlung)
   range: Zahlungsart.
(self slot: #Jahresprämie)
      range: Geldbetrag;
      minCardinality: 1.
                  •
                  •
                  •
```

Abb. 65: Beispiel für die Transformation in SFK

Die Anbindung von SFK wurde zunächst nur versuchsweise durchgeführt. Dabei wurde hingenommen, daß auch SFK eine vollständige Transformation des Objektmodells nicht erlaubt. Das liegt natürlich vor allem daran, daß die Semantik des Objektmodells dazu nicht hinreichend formalisiert ist - dabei ist vor allem an die Beschreibung von Diensten zu denken. Daneben war einschränkend zu berücksichtigen, daß SFK auf einem zwar ähnlichen, aber eben doch nicht identischen Metamodell basiert. So sind für die grundsätzlich mögliche Spezifikation bestimmter Teile des Modells (wie Triggers oder Guards) keine speziellen Konstrukte vorhanden. Die automatische Transformation in SFK beschränkt sich denn auch auf Attribute und Beziehungen, sowie die mit ihnen verbundenen Zugriffsdienste. Auf der Instanzenebene überprüft das von SFK erzeugte Lauf-

zeitsystem die Klassen von Attributen und assoziierten Objekten sowie die zugehörigen Kardinalitäten. Abbildung 65 zeigt vom OMD generierten SFK-Code, dabei ist die kursiv gesetzte spezielle Vorbedingung manuell ergänzt. Sie besagt, daß der Schadenfreiheitsrabatt nur dann erhöht werden darf, wenn er kleiner als der maximal zulässige ist. Der reservierte Bezeichner "range:" markiert die Klasse eines Attributs. Nach dem Kompilieren eines solchen Frames kann mit Hilfe von Lese- und Schreibdiensten auf die Attribute zugegriffen werden.

SFK bietet nur eine rudimentäre Benutzerschnittstelle für den Zugriff auf Instanzen. Um dem Betrachter eine anschaulichere Darstellung zu bieten und gleichzeitig die im Metamodell vorgesehene Beschreibung der Präsentation eines Objekts zu verdeutlichen, werden die Default Views des jeweiligen Objekts genutzt. Dazu werden für alle Attribute die zugehörigen Default Views in einem Fenster dargestellt. Die Default Views werden mit dem jeweils zugeordneten Etikett kommentiert. Jedem Widget der Views werden dabei zwei Dienste zum Lese- und Schreibzugriff auf das jeweilige Attribut zugeordnet. Die Anordnung der Views am Bildschirm erfolgt unabhängig von Benutzerbedürfnissen: Die Widgets werden sequentiell von oben nach unten in einem entsprechend dimensionierten Fenster gelistet - dabei werden zunächst geerbte Attribute gezeigt. Die Zahl der Widgets kann beliebig groß sein, da der Inhalt eines erzeugten Fensters gerollt werden kann. Abbildung 66 zeigt die für die Objekte einer Klasse generierte Benutzerschnittstelle und Einträge für eine Instanz.

Die Instanzen können (temporär) in einer Listenstruktur abgelegt werden. Zudem ist es möglich, die Anordnung der Widgets im Fenster interaktiv zu verändern. Dazu wird auf den in ObjectForms® enthaltenen Interface-Builder zurückgegriffen. Dabei ist allerdings zu berücksichtigen, daß es weniger darum geht, ein konkretes Fenster des letztlich zu implementierenden Systems zu beschreiben, als vielmehr darum, die Darstellung der einzelnen Attribute zu veranschaulichen. Schließlich ist in einem konkreten Anwendungskontext in der Regel nur ein Teil eines Objekts, gegebenenfalls zusammen mit Teilen anderer Objekte, von Interesse. Im Unterschied zum OMD wird der Verarbeitungskontext im Office Procedure Designer berücksichtigt, so daß dort kontextspezifische Benutzerschnittstellen erzeugt werden können (vgl. V.3.2).

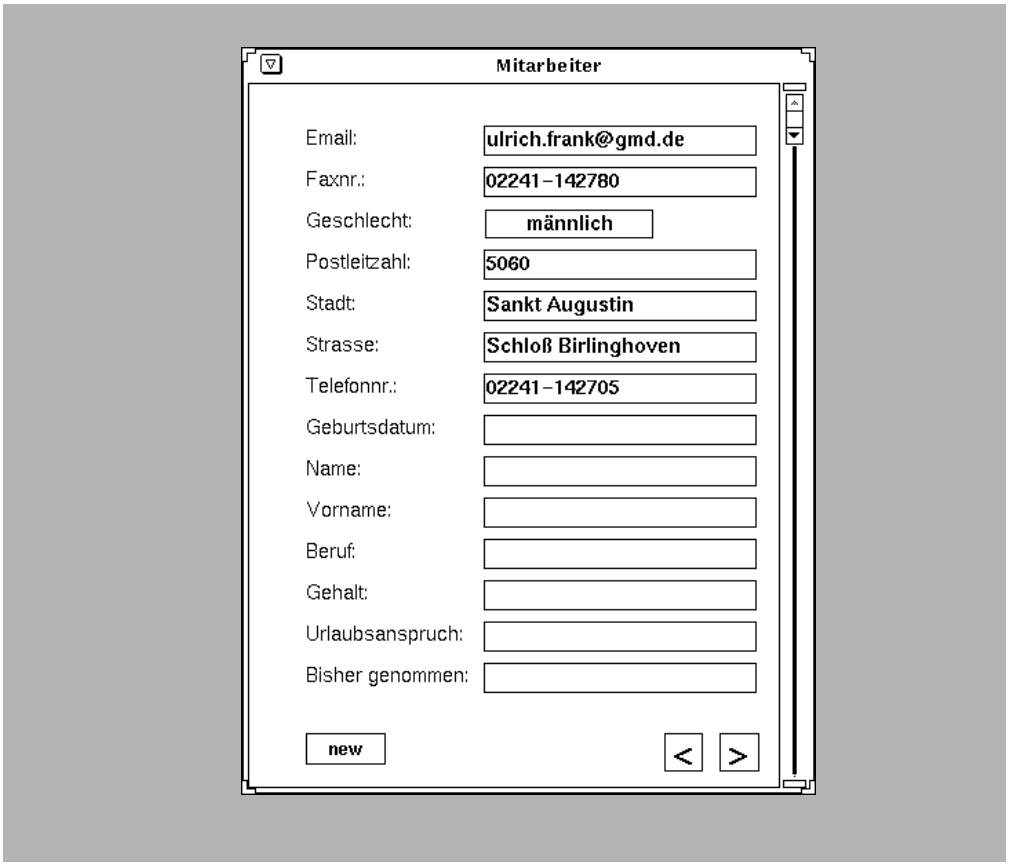


Abb. 66: Vom OMD generierte Benutzerschnittstelle zur prototypischen Instanzierung der Objekte einer Klasse.

3. Der Office Procedure Designer

Wie der Name es andeutet beschränkt sich die Nutzung des OMD auf den Entwurf von Objektmodellen, von dynamischen Aspekten der Nutzung von Objekten wird abstrahiert. Im Metamodell wurde dazu ein Ansatz zur Konzeptualisierung von Vorgängen (oder langen Transaktionen) vorgestellt (vgl. IV.2.1.2). Daneben spielen Vorgänge in der organisatorischen Perspektive eine Rolle. Dort werden sie allerdings auf einem anderen Abstraktionsniveau betrachtet: Im Vordergrund steht nicht die formale Betrachtung automatisierter Vorgänge, sondern die am Unternehmensgegenstand orientierte Erfassung und Analyse von Geschäftsprozessen. Im Office Procedure Designer (OPD) sollen diese beiden Blickwinkel auf Vorgänge oder Prozesse vereint werden. Einerseits werden dabei die im Metamodell dargestellten Anforderungen an die Integrität von Vorgängen

berücksichtigt, andererseits wird dem Umstand Rechnung getragen, daß Geschäftsprozesse für die Analyse und Neugestaltung der Organisation eines Unternehmens von zentraler Bedeutung sind.

Bei der Gestaltung des OPD standen die folgenden Anforderungen im Vordergrund:

- *Integration mit dem Objektmodell*. Wenn in einem Vorgang Objekte des Informationssystems benötigt werden, sollte dies - um die Konsistenz des Gesamtmodells zu fördern - durch eine Referenz auf das Objektmodell ausgedrückt werden. Umgekehrt sollte der Entwurf (automatisierbarer) Vorgänge objektorientiert erfolgen (wie es im Metamodell bereits beschrieben wurde), um so die entsprechenden Objekte im Rahmen des Objektmodells abbilden und verwalten zu können.
- *Analyse betriebswirtschaftlicher Zusammenhänge*. Die betriebswirtschaftliche Beurteilung von Geschäftsprozessen sieht sich einer Reihe von Kriterien gegenüber, die zum Teil in komplexer Weise verknüpft sind. Das Werkzeug sollte dazu beitragen, die bedeutsamen Zusammenhänge transparent zu machen und Schwachstellen aufzudecken. Dazu gehört auch eine Unterstützung der Simulation von Alternativen.
- *Anschaulichkeit*. Geschäftsprozesse sind in der Regel sehr viel besser geeignet, um mit Mitarbeitern über ein Unternehmen zu sprechen, als die isolierte Betrachtung von Objekten. Mehr als der OMD richtet sich deshalb der OPD auch an Domänenexperten ohne spezifische Modellierungskenntnisse. Für diese Gruppe sind allerdings gängige Ansätze zur Präsentation von Petri-Netzen wenig geeignet. Im OPD sollte deshalb eine verständlichere Darstellung von Netzen realisiert werden.

3.1 Die Konzeptualisierung von Geschäftsprozessen

Im Metamodell und auch in der organisatorischen Perspektive werden Vorgänge oder Geschäftsprozesse[1] als Petri-Netze modelliert. Dabei wird ein Vorgang in Teilvorgänge unterteilt, die wiederum als Vorgänge modelliert sein mögen. Außerdem ist der Informationsfluß zwischen den Teilvorgängen zu modellieren. Diese Grundstruktur wird im OPD übernommen, ist im Detail allerdings so anzupassen, daß sowohl die Anforderungen der Informationssystem-Perspektive als auch die einer organisatorischen Betrachtung berücksichtigt werden. Falls ein Geschäftsprozeß in Form eines Vorgangs (im Sinne einer langen Transaktion) unter Rückgriff auf das in IV.2.1.1 dargestellte Objektmodell beschrieben ist, kann diese Beziehung im OPD festgehalten werden.

1. Beide Begriffe werden im folgenden als synomym betrachtet.

3.1.1 Virtuelle Vorgangsmappen

Im Metamodell wird zur Beschreibung der zwischen den Teilvorgängen ausgetauschten Informationen ein sogenanntes Vorgangsdokument eingeführt. Ein solches Objekt stellt gleichsam einen Container für die von Teilvorgang zu Teilvorgang zu transportierenden Objekte dar. Für den OPD ist dies eine zu enge Sichtweise, da er auch die Modellierung von Geschäftsprozessen erlauben soll, in denen nicht nur solche Informationen verarbeitet werden, die in einem DV-System abliegen. Für die Modellierung wird deshalb eine *virtuelle Vorgangsmappe* eingeführt. Sie entspricht weitgehend dem genannten Vorgangsdokument, kann aber neben Objekten auch Verweise auf andere Informationsträger beinhalten.

Für jeden der in einem Vorgangsnetz verzeichneten Zustände einer Vorgangsmappe ist dann anzugeben, welche Voraussetzungen ihr Inhalt erfüllen muß, um dem jeweiligen Zustand zu entsprechen. Im OPD werden zur Zeit drei Arten von Informationsträgern unterschieden: *Objekte*, (Papier-) *Formulare* und *Anlagen*. Um einen bestimmten Zustand ("Antrag formal korrekt") zu definieren, sind die Zustände der benötigten Objekte und Formulare zu beschreiben. Für die Anlagen wird unterstellt, daß sie nicht verändert werden - es ist also lediglich sicherzustellen, daß sie vorhanden sind. Um dem Benutzer eine komfortable und konsistente Beschreibung der erforderlichen Zustände der zu berücksichtigenden Objekte zu ermöglichen, werden ihm unter Rückgriff auf das Objektmodell die Dienste der Objekte präsentiert. Ein Zustand kann dann beschrieben werden, indem mit Hilfe eines ausgewählten Dienstes eine Bedingung formuliert wird, die erfüllt sein muß. In der gegenwärtigen Implementierung beschränkt sich dies auf die Auswahl solcher Dienste, die einen booleschen Wert zurückliefern. Falls ein solcher Dienst für die Klasse noch nicht definiert ist, ist das Objektmodell entsprechend zu erweitern. Auf diese Weise können für jedes Objekt beliebig viele Bedingungen spezifiziert werden, die dann durch logische Konjunktion verknüpft werden. Es muß allerdings nicht für jedes Objekt ein Zustand definiert werden. Objekte können auch nur als notwendige Anlage dienen - wie beispielsweise ein Objekt, das einen mit Textverarbeitung erstellten Brief enthält.

Um den Zustand eines Formulars zu definieren, wird dem Benutzer eine Liste mit möglichen Zuständen präsentiert ("vollständig ausgefüllt", "abgezeichnet", "zur Kenntnis genommen" etc.). Durch einfache Auswahl solcher Teilzustände kann dann der benötigte Gesamtzustand definiert werden. Dabei ist es allerdings wünschenswert, daß Formulare auch als Objekte, nämlich in Form von Referenzdokumenten (vgl. IV.3.4) beschrieben sind. Auf diese Weise ist es möglich, Eigenschaften von Formularen (wie ihren Aufbau und Inhalt, den Verwendungszweck, die Zustände etc.) im Objektmodell zu verwalten. Falls es für die Kennzeichnung eines Zustands wesentlich ist, wer ihn herbeigeführt hat (wie etwa

"zur Kenntnis genommen") kann der Benutzer eine entsprechende Rolle aus einer Liste auswählen. Die möglichen Anlagen (wie Briefe, Berichte etc.) können ebenfalls aus einer Liste selektiert werden. Auch sie sollten - in dem oben mit Informationsmanagement gekennzeichneten Teil der organisatorischen Perspektive - im Objektmodell abgebildet sein.

Abbildung 67 zeigt die Angaben, die im OPD zur Beschreibung des Zustands einer Vorgangsmappe erfaßt werden. Die dargestellten Kardinalitäten sind durch die - vom OPD überwachte - Randbedingung zu ergänzen, daß insgesamt mindestens ein Informationsgegenstand als Inhalt angegeben werden muß.

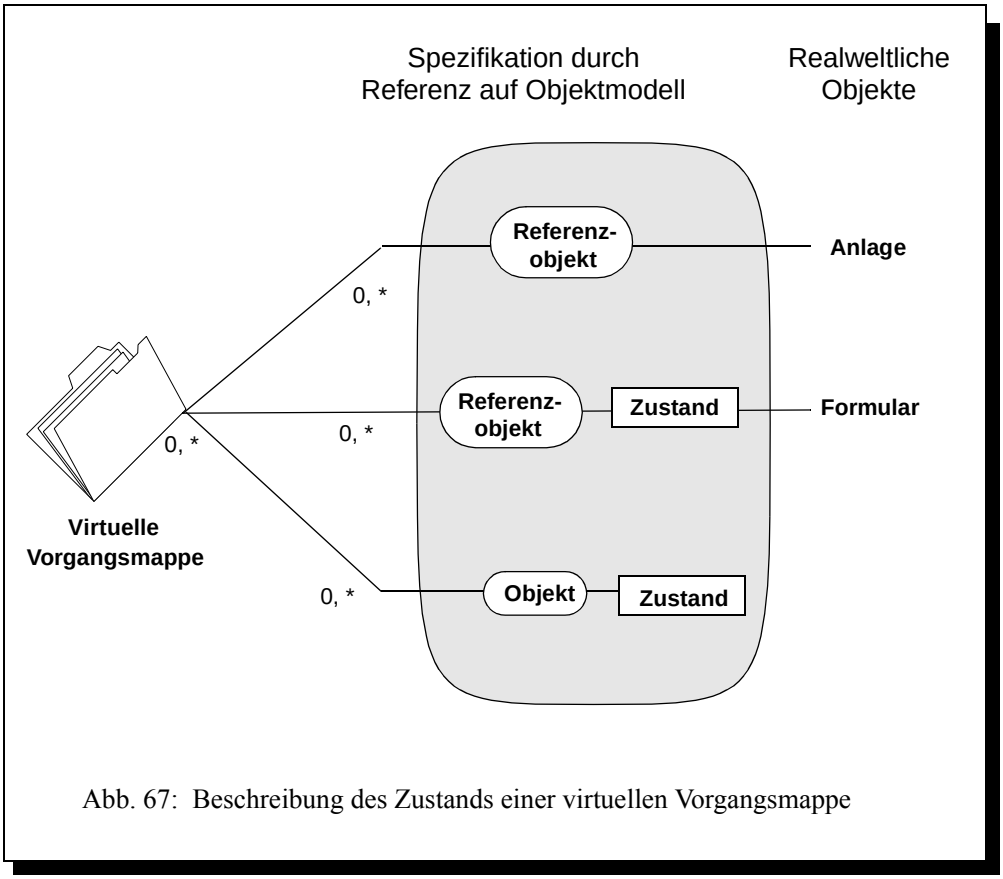


Abb. 67: Beschreibung des Zustands einer virtuellen Vorgangsmappe

Ein einfaches Beispiel für den Zustand einer virtuellen Vorgangsmappe ist der Zustand "formal korrekt" im Rahmen eines Vorgangs zur Bearbeitung von Urlaubsanträgen. Er könnte gekennzeichnet sein durch die folgenden Zustände, die jeweils durch Rückgriff auf die kursiv gesetzten Dienste spezifiziert werden:

- Formular: Urlaubsantrag (Referenzobjekt) *ist_vollständig_ausgefüllt*
- Objekt: Mitarbeiter *restUrlaub* > 0

An dieser Stelle wird deutlich, daß der Entwurf eines Geschäftsprozesses und der eines Objektmodells interagieren: Bei der Beschreibung eines Zustands der Vorgangsmappe kann sich herausstellen, daß eine benötigte Klasse im Objektmodell noch nicht oder noch nicht vollständig erfaßt ist. Um das gesamte Modell konsistent zu halten, wird der Benutzer in diesem Fall gezwungen, zunächst das Objektmodell zu erweitern, um anschließend auf die ergänzten Elemente referieren zu können.

3.1.2 Die Beschreibung von Teilvorgängen

Ein Teilvorgang wird in Anlehnung an die Darstellung in Abbildung 30 modelliert. Dazu können folgende Angaben zugeordnet werden - zum Teil unter Rückgriff auf das gegebenenfalls entsprechend zu erweiternde Objektmodell:

- *Aufbauorganisatorischer Rahmen*. Dazu können jeweils eine ausführende und eine verantwortliche Stelle (die identisch sein können) sowie eine übergeordnete organsatorische Einheit ausgewählt werden. Wenn diese Angaben für alle Teilvorgänge identisch sind, können sie auch einmal für den gesamten Vorgang gemacht werden. Sie werden dann an die Teilvorgänge propagiert - und können dort anschließend überschrieben werden.
- *Benötigte Ressourcen*. Hier können beliebig viele Klassen aus der organisatorischen Perspektive und dem Teilaspekt "Systemverwaltung" der Informationssystem-Ebene selektiert werden. Für jede der gewählten Ressourcen ist anzugeben, ob sie möglicherweise oder immer benötigt wird.
- *Ausnahmen*. An dieser Stelle ist allgemein an mögliche Störfälle in Arbeitsabläufen und ihre Behandlung zu denken. Dazu können auch technische Störfälle - wie etwa der Ausfall peripherer Geräte - gehören. Das gilt aber nur dann, wenn sie ablauforganisatorische Konsequenzen haben. Ihre generelle Behandlung (wie etwa die Benachrichtung des Benutzers oder das Ausweichen auf alternative Peripherie) ist an anderer Stelle zu beschreiben. Um eine einheitliche Ausnahmebehandlung für das gesamte Modell zu ermöglichen, sind alle Ausnahmen in einem zentralen Verzeichnis anzulegen. Dort sind dann unter anderem die Bezeichnung und eine Beschreibung der zu ergreifenden Maßnahmen festgelegt.
- *Informations- und Kommunikationsbedarf*. Zur Durchführung eines Teilvorgangs ist es unter Umständen nötig, weitere Informationen heranzuziehen. Dazu wird im OPD folgende Unterteilung vorgenommen:
  - *Objekte*. Die in der Vorgangsmappe enthaltenen Objekte (genauer: die sie repräsentierenden Klassen) werden übernommen, weitere können aus dem Objektmodell ausgewählt werden. Für jedes Objekt können dann noch die benötigten Dienste ausgewählt werden - also beispielsweise von einem

Objekt der Klasse "Versicherungsnehmer" die Dienste "bisherigeSchäden", "alter" und "beruf". Damit ist zweierlei intendiert: Zum einen dient die Frage nach den benötigten Objekteigenschaften der Vervollständigung des Objektmodells. Andererseits wird dadurch das Erzeugen einer prototypischen Benutzerschnittstelle ermöglicht (vgl. V.3.2). Wenn mit einem Dienst ein schreibender Zugriff verbunden ist, kann eingetragen werden, woher die dazu nötige Information stammt. Dazu kann als Quelle ein Objekt, ein sonstiger Informationsträger oder eine Person/Institution ausgewählt werden. Falls es sich um einen lesenden Dienst handelt, sollte angegeben werden, wozu die damit verbundene Information benötigt wird. Dabei kann zwischen drei Möglichkeiten gewählt werden: ein sonstiger Informationsträger, eine Person oder die Verwendung als Entscheidungshilfe für den Bearbeiter.

- *Sonstige Informationsträger*. Dazu gehören die Formulare und Anlagen der vom Teilvorgang übernommenen Vorgangsmappe. Darüber hinaus können weitere Informationsträger, die in der organisatorischen Perspektive des Objektmodells (unter dem Aspekt "Informationsmanagement") erfaßt sind, zugeordnet werden. Dabei ist an Berichte, Akten, Fachzeitschriften, aber auch an externe Datenbanken zu denken. Eigenschaften der Informationsträger wie Standort, Kosten und anderes sind im Objektmodell verzeichnet. Für jeden Informationsträger kann angegeben werden, welche Teilinformationen benötigt werden (beispielsweise für eine Tageszeitung: "Devisenkurse") oder welche Operationen auszuführen sind (wie etwa "Abzeichnen"). Wenn es sich dabei um eine Schreib-Operation handelt, ist - wie auch bei Objekten - die Quelle auszuwählen. Auch dazu könnte auf entsprechende Einträge im Objektmodell verwiesen werden. Gegenwärtig ist im OPD allerdings die Eingabe eines strukturierten Textes vorgesehen.
- *Personen, Institutionen*. An dieser Stelle sind die Personen oder Institutionen auszuwählen, mit denen im Verlauf der Bearbeitung kommuniziert werden muß oder könnte. Dazu kann dann jeweils - unter Rückgriff auf eine gegebenenfalls zu erweiternde Liste - angegeben werden, welches Medium üblicherweise genutzt wird und was der Anlaß und der Gegenstand der Kommunikation sind.

• *Durchführungsregeln*. Die in einem Teilvorgang vorzunehmenden Operationen erfolgen nach gewissen Kriterien oder Regeln. Sofern sie sich auf die zuvor ausgewählten Objekte beziehen, könnten sie unter Rückgriff auf deren Dienste beschrieben werden. In der gegenwärtigen Version des OPD werden solche Regeln allerdings als freier Text eingegeben, wobei gegebenenfalls die Namen der referenzierten Klassen verwendet werden sollten. Dabei wird dem Benutzer eine Schablone für Produktionsregeln präsentiert, in denen jeweils die von dem Teilvorgang zu erzeugenden Zustände der Vorgangsmappe den Aktionsteil bilden.

- *Ausführungszeiten*. Für jeden Teilvorgang kann eine durchschnittliche und eine maximale Bearbeitungszeit angegeben werden. Das Überschreiten der maximalen Zeit sollte in der Liste der Ausnahmen verzeichnet sein.

Für die einzelnen Elemente des Informationsbedarfs kann jeweils angegeben werden, ob sie für die Vorgangsbearbeitung optional oder obligatorisch sind.

3.1.3 Prototypische Instanzen

In IV.2.1.2 wurde darauf hingewiesen, daß die Beschreibung eines Vorgangs es mitunter erfordert, die Identität eines Objekts zu kennzeichnen. Das gilt immer dann, wenn in einem Vorgang mehr als ein Objekt einer Klasse zu berücksichtigen sind. Im OPD ist diese Anforderung nur in Teilen umgesetzt. So können die Inhaber der den Teilvorgängen zugeordneten Stellen durch eine Kennzeichnung (von 1 bis n) differenziert werden. Auf diese Weise ist es möglich auszudrücken, daß zwei verschiedene Teilvorgänge von ein und demselben Mitarbeiter zu betreuen oder zu verantworten sind. Daneben kann auf bestimmte Instanzen mit Hilfe von Diensten, die assoziierte Objekte zurückliefern, rekurriert werden. Auf diese Weise kann etwa für ein Objekt der Klasse "Versicherungsnehmer" ausgedrückt werden, daß die ihm zugeordneten Verträge gemeint sind. Wenn es darüber hinaus bedeutsam ist, zwischen verschiedenen Instanzen einer Klasse zu unterscheiden, kann dies im Kommentar beschrieben werden.

Neben der mitunter erforderlichen Identifikation von Instanzen gibt es - vor allem im Hinblick auf die im folgenden zu beschreibenden Analyse- und Simulationsverfahren - die Notwendigkeit, Angaben über Instanzen bestimmter Klassen zu machen. So können etwa die in einem Teilvorgang benötigten Ressourcen durch Beschaffungs- oder Nutzungskosten gekennzeichnet werden, für involvierte Mitarbeiter können Lohn- und Gehaltskosten sowie Ausfallzeiten angegeben werden. Dabei wird gewiß nicht eine bestimmte Instanz (etwa ein bestimmter Mitarbeiter) betrachtet. Andererseits handelt es sich hier auch nicht um generelle Klasseneigenschaften. Vielmehr werden hier Eigenschaften zugeordnet, die für eine bestimmte Grundgesamtheit (z.B. die Sachbearbeiter eines Unternehmens) als repräsentativ angesehen werden. Zur Kennzeichnung dieser besonderen Objekte verwenden wir hier den Begriff "prototypische Instanzen". Die Eigenschaften prototypischer Instanzen können zur Zeit nur manuell eingegeben werden. Sobald allerdings ein mit der Werkzeugumgebung integriertes Informationssystem mit tatsächlichen Instanzen besteht, können die prototypischen Instanzen mit Hilfe geeigneter statistischer Verfahren erzeugt werden. Auf diese Weise ergibt sich die Möglichkeit, organisatorische Analysen in regelmäßigen Abständen durchzuführen, ohne daß damit zwangsläufig der übliche hohe Erhebungsaufwand verbunden wäre.

3.2 Die interne Abbildung von Vorgangsmodellen und ihrer Integration mit Objektmodellen

Jedem Objektmodell können mehrere Vorgangsmodelle zugeordnet werden. Umgekehrt ist jedes Vorgangsmodell genau einem Objektmodell zugeordnet. Beim Anlegen eines neuen Vorgangsmodells wird vom OMD eine neue Vorgangsidentifikation erzeugt. Sie dient der Referenzierung eines Vorgangsmodells aus einem Objektmodell. Eine solche Referenzierung ist vor allem aus zwei Gründen nötig. Zum einen sollte die Identifikation der zu einem Objektmodell gehörigen Vorgangsmodelle nicht auf die im Zeitverlauf veränderlichen Vorgangsnamen rekurrieren, zum anderen ist vorgesehen, in den betroffenen Referenzverzeichnissen des Objektmodells die Stellen der zugehörigen Vorgangsmodelle abzulegen, von denen aus bestimmte Teile des Objektmodells referenziert werden (vgl. Anhang, Teil I). So kann der OMD vor dem Löschen einer Klasse oder eines Dienstes auf effiziente Weise prüfen, ob auf diese Elemente in einem Vorgangsmodell verwiesen wird.

Die Abbildung eines Vorgangsmodells selbst erfolgt einerseits mit einem Graphen, in dem Informationen über die räumliche Anordnung der Knoten eines Vorgangsnetzes sowie Angaben über Vorgänger und Nachfolger abgelegt sind. Jeder Knoten (also entweder ein Zustand des virtuellen Vorgangsdokuments oder ein Teilvorgang) wird dabei mit einer eindeutigen Identifikation versehen. Diese Identifikation dient dann als Schlüssel für den Eintrag in ein komplexes Verzeichnis, das eine umfassende Beschreibung des Vorgangsmodells enthält. Abbildung 68 zeigt einen Teil dieses Verzeichnisses.

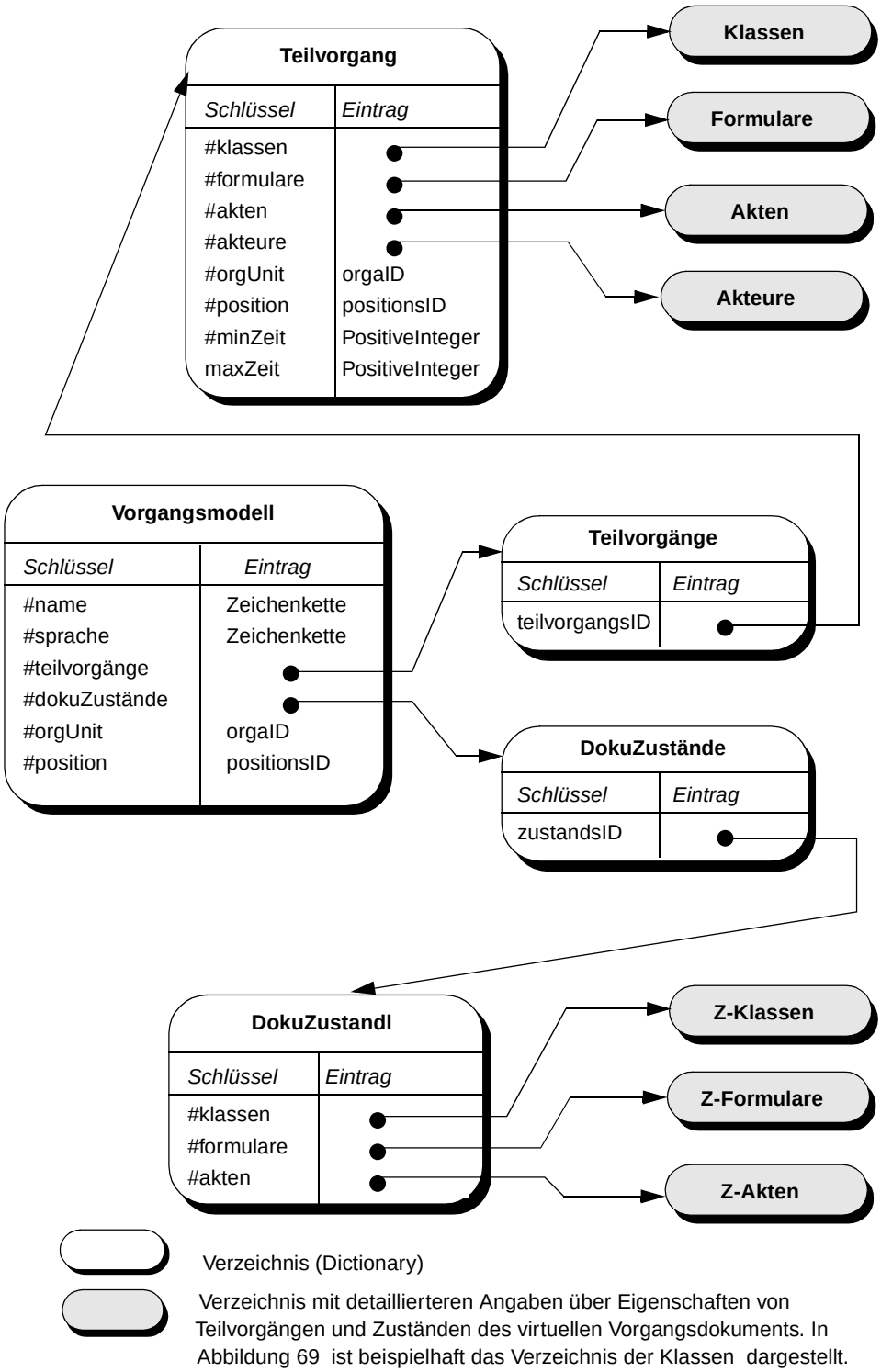


Abb. 68: Einträge in ein Vorgangsmodell-Verzeichnis. Die Angaben zum organisatorischen Kontext (Organisationseinheit, Position) werden vom Vorgangsmodellverzeichnis in die Verzeichnisse der Teilvorgänge propagiert. Dort können sie überschrieben werden.

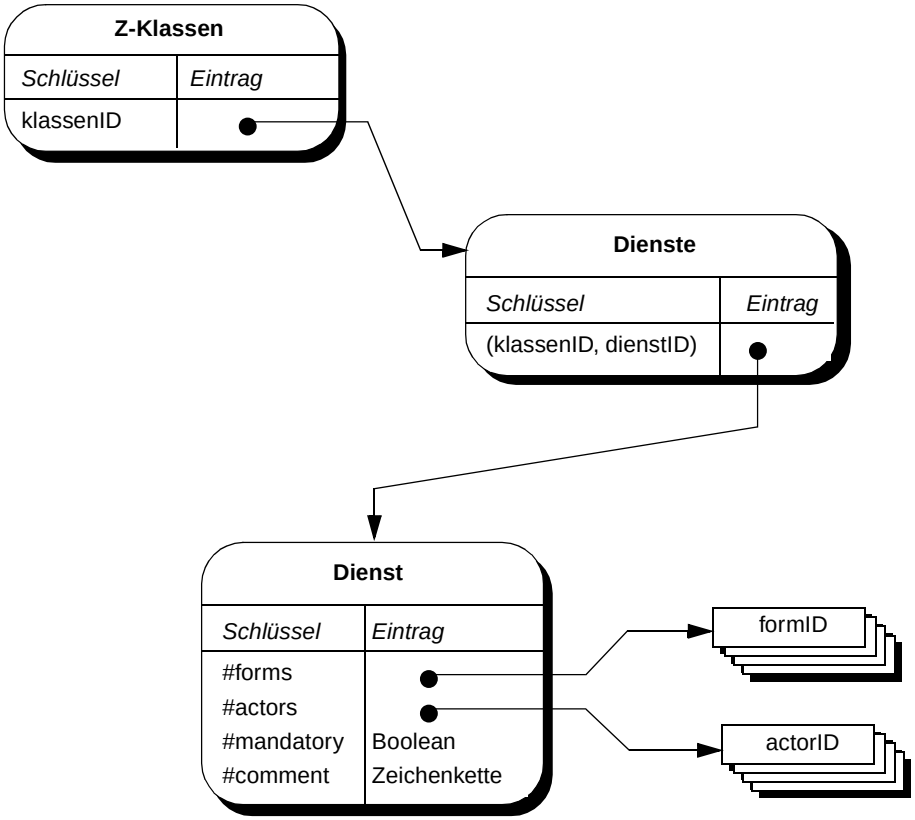


Abb. 69: Die einem Teilvorgang zugeordneten Dienste können nicht allein mit der Dienst-Identifikation der zuvor zugeordneten Klasse identifiziert werden, da sie u.U. von einer Oberklasse stammen. Deshalb wird zur Identifikation eines Dienstes ein Array aus der Dienst-Identifikation und der Identifikation derjenigen Klasse, in der der Dienst spezifiziert ist, verwendet.

3.3 Analyse- und Simulationsverfahren

Das Modell eines Geschäftsprozesses macht nicht unmittelbar deutlich, wie eine entsprechende organisatorische Implementierung zu bewerten ist. Daneben ist es im Hinblick auf die Systementwicklung wünschenswert, dem zukünftigen Benutzer eine anschauliche Vorstellung von der Funktionsweise des Systems zu bieten. Der OPD bietet eine Reihe von Verfahren, die an diese Anforderungen anknüpfen.

Zum Zweck einer betriebswirtschaftlich/organisatorischen Analyse kann auf der Grundlage der im Modell gemachten Angaben eine Kommunikationsanalyse

durchgeführt werden. Dazu wird - wahlweise für einen Teilvorgang oder für den gesamten Vorgang - ein Kommunikationsdiagramm in Sternstruktur generiert. Darin sind die genannten Stellen und Institutionen/Personen als Kommunikanden mit Hilfe anschaulicher Symbole dargestellt. Die Symbole unterscheiden sich für unternehmensinterne und -externe Kommunikanden. Sie werden durch Kanten verbunden, deren Dicke die Häufigkeit der Nennung der jeweiligen Kommunikation widerspiegelt. An die Kanten sind zudem Symbole für die jeweils genutzten Medien angetragen (vgl. Abb. 70).

Bei der Spezifikation der Teilvorgänge kann für die Veränderungen an Objekten oder Formularen angegeben werden, woher die dazu nötigen Informationen stammen. Auf der Grundlage dieser Angaben erzeugt der OPD eine Liste, in der Medienbrüche verzeichnet sind. Dabei wird ein Medienbruch als der Übergang von einem Informationsträger auf einen anderen angesehen. Jede Zeile der Liste besteht aus einem Text, in dem die transferierte Information und die beiden beteiligten Informationsträger dargestellt sind. Beispiel:

Objekt Kunde schadenfreiheitsrabatt --> *Informationsträger* Antragsformular

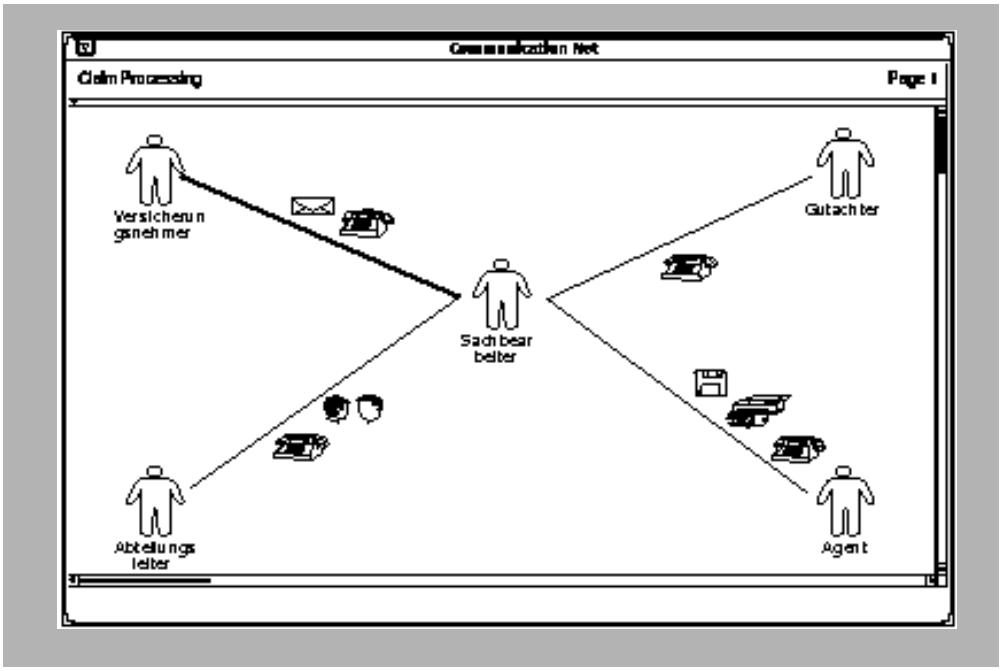


Abb. 70: Vom OPD erzeugter Kommunikationsstern eines Geschäftsprozesses

Neben den genannten Analyseverfahren bietet der OPD zwei Simulationsansätze. Unter Simulation wird dabei ein Verfahren verstanden, das darauf zielt, eine anschauliche Darstellungen möglicher Wirklichkeiten zu erzeugen. Typi-

scherweise werden dazu vereinfachende Annahmen gemacht. Im ersten Fall geht es um die Simulation der Benutzerschnittstelle des in der Vorgangsbearbeitung eingesetzten Informationssystems. Dazu wird für jeden Teilvorgang aus der Liste der zugeordneten Objekte und der jeweils ausgewählten Dienste eine prototypische Benutzerschnittstelle erzeugt. Das jeweils für einen Dienst benötigte Widget wird dabei durch den Zugriff auf die entsprechende Stelle des Objektmodells ermittelt.

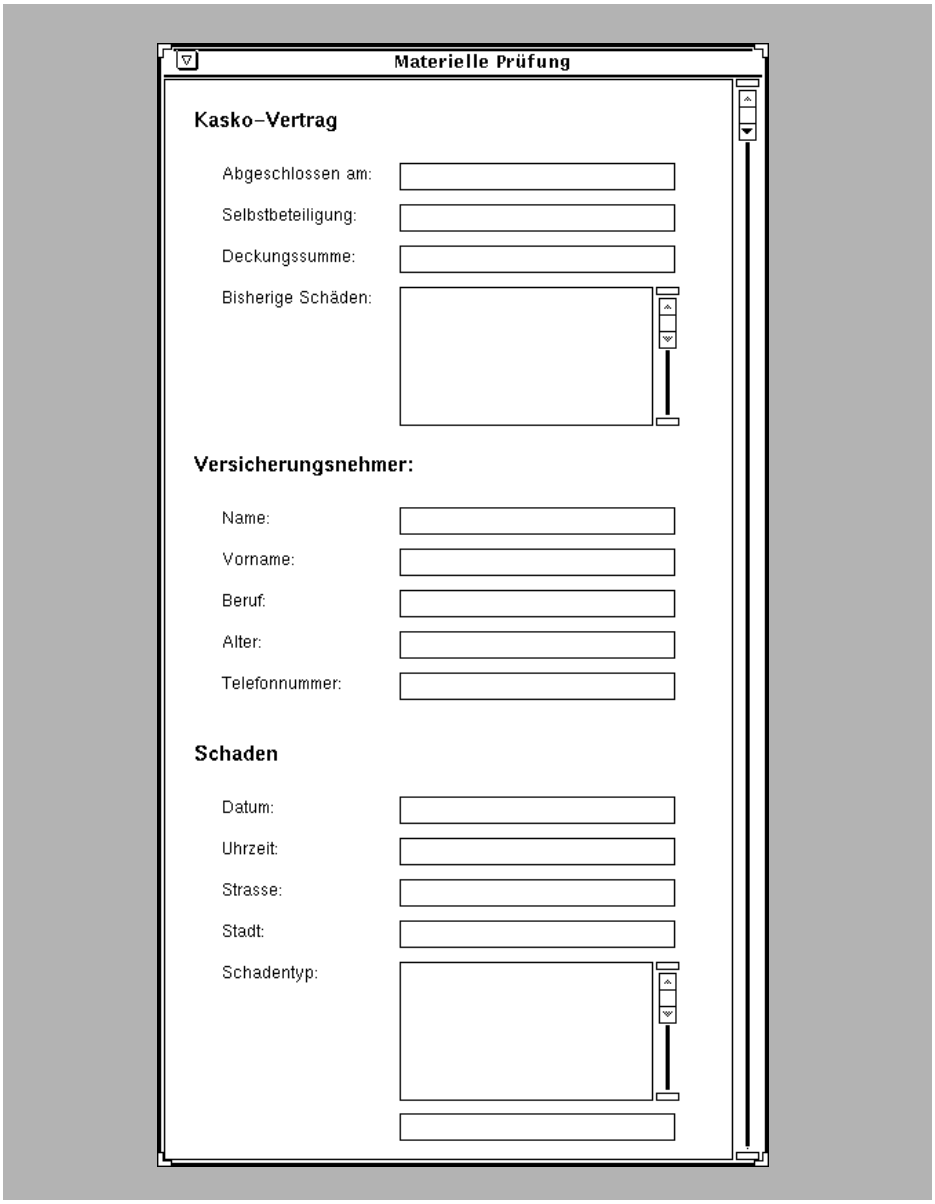


Abb. 71: Für einen Teilvorgang generierte prototypische Benutzerschnittstelle.

Die Anordnung der Widgets erfolgt ähnlich wie in IV.2.1.1.1.4 beschrieben: Die einzelnen Dienste werden sequentiell abgearbeitet, die zugeordneten Widgets von oben nach unten in ein Fenster plaziert. Die auf diese Weise erzeugte Benutzerschnittstelle ist in der Regel nicht zufriedenstellend (vgl. Abb. 71).Sie bietet allerdings eine günstige Voraussetzung, um gehaltvolle Stellungnahmen von zukünftigen Anwendern zu erhalten. So können die Abmessung des Fensters und die Anordnung der Widgets interaktiv verändert werden. Dabei sind auch Hinweise auf die benötigten Kontrollstrukturen zu erwarten. Zudem dient die prototypische Benutzerschnittstelle dazu, die Beschreibung des Teilvorgangs auf Vollständigkeit zu überprüfen.

Im zweiten Ansatz kann der Ablauf von Geschäftsprozessen simuliert werden. Da die im Modell erfaßten Angaben für eine vollständige Automatisierung des jeweiligen Vorgangs nicht hinreichen, sind einschränkende Annahmen nötig. Dazu ist für jedes der von einem Teilvorgang produzierten Ergebnisse eine geschätzte Wahrscheinlichkeiten anzugeben - sinnvollerweise nur dann, wenn mehr als ein Ergebnis produziert wird. Der OPD überprüft dabei, ob sich die Wahrscheinlichkeiten zu 100 summieren. Abbildung 70 zeigt ein Beispiel für die Zuordnung von Wahrscheinlichkeiten zu den Ergebnissen von Teilvorgängen.

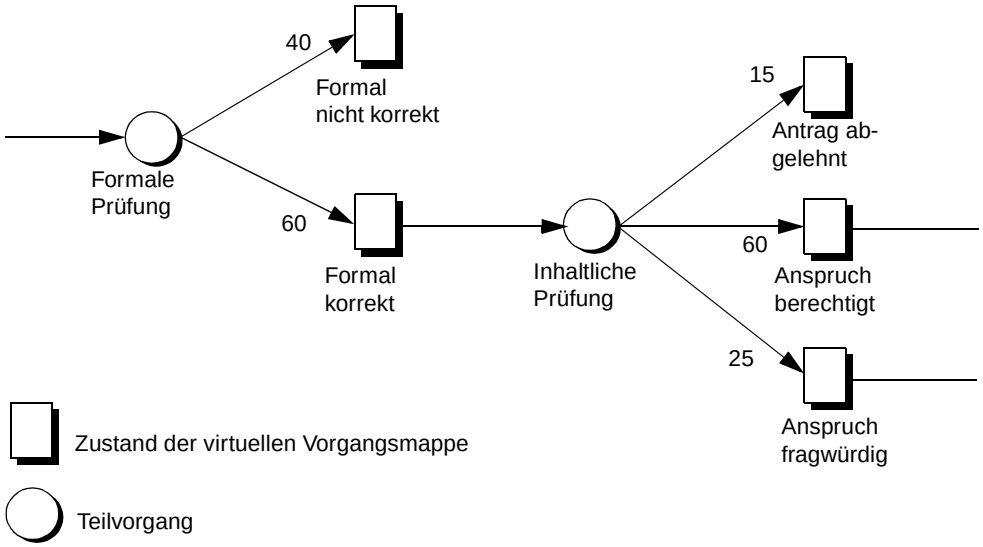


Abb. 72: Beispiel für Wahrscheinlichkeiten der möglichen Ergebnisse von Teilvorgängen.

Für einen Simulationslauf ist dann eine Anzahl von Vorgängen anzugeben (beispielsweise die Zahl der an einem Tag zu erwartenden Vorgänge). Der OPD berechnet die benötigte Ausführungszeit. Die Ausführung eines Vorgangs kann auch in verschiedenen Verzögerungsstufen am Bildschirm dargestellt werden.

Dazu werden Symbole für Vorgangsmappen durch die grafische Darstellung des Netzes bewegt. Teilvorgänge, die Engpässe darstellen, werden auf diese Weise deutlich herausgestellt. Der eigentliche Nutzen der Simulation liegt darin, daß der Benutzer einzelne Teile des Modells modifizieren kann, um sich dann die Auswirkungen dieser Veränderung darstellen zu lassen. Dazu kann er einerseits die Angaben über Ausführungszeiten und Wahrscheinlichkeiten ändern. Daneben - und dies ist sicher der wichtigere Fall - kann die Kapazität eines Vorgangs verändert werden. Dazu wird die Kapazität eines Knotens, der einen Teilvorgang repräsentiert, vervielfacht. Diese Veränderung kann durch eine Vervielfachung des Knotens dargestellt werden. Eine solche Veränderung des Netzes ist für Betrachter, die mit Petri-Netzen nicht vertraut sind, mitunter schwer nachvollziehbar ist. Die Übersichtlichkeit wird zudem grundsätzlich durch eine unter Umständen erhebliche Vergrößerung des Netzes (vgl. Abb. 12) gefährdet. Es gibt deshalb auch die Möglichkeit, die Darstellung des Netzes unverändert zu lassen. Die anschließende Simulation berechnet den Effekt der Kapazitätseränderung und stellt sie auf Wunsch in einem verlangsamten Ablauf dar.

Darüber hinaus entsteht eine Fülle weiterer Analyse- und Simulationsmöglichkeiten, wenn auch Interdependenzen mit anderen Vorgängen berücksichtigt werden. Im Grenzfall kann dann die gesamte Ablauforganisation eines Unternehmen Gegenstand einer ganzheitlichen Analyse und Simulation werden. Verfahren zur Berücksichtigung mehrerer Vorgänge sind gegenwärtig nicht implementiert.

3.4 Die Benutzerschnittstelle

Ähnlich wie beim OMD sind auch im OPD verschiedene Abstraktionsebenen vorgesehen. Um einen Geschäftsprozeß in seiner Gesamtheit zu entwerfen, um also die zeitlichen Zusammenhänge zwischen den Teilvorgängen auszudrücken, steht dem Benutzer ein dedizierter Grafik-Editor (eine Modifikation des im OMD verwendeten Editors) zur Verfügung. Dabei wird eine - im Vergleich zu gängigen Visualisierungen von Petri-Netzen - anschaulichere Darstellung ermöglicht: Die Zustände der virtuellen Vorgangsmappe sowie die einzelnen Teilvorgänge können mit Hilfe selbstsprechender Symbole (vgl. Abb. 73) abgebildet werden. Der Benutzer wählt die Symbole aus und ordnet sie in der gewünschten Reihenfolge. Um die gerichteten Kanten anlegen zu lassen, werden zwei Symbole nacheinander aktiviert. Die Verbindung wird allerdings nur realisiert, wenn sie zulässig ist.Eine solche grafische Darstellung von Vorgängen ist nicht nur für den Entwurf geeignet, sondern bietet auch eine attraktive Benutzerschnittstelle für die Betrachtung instanzierter Vorgänge: Der Benutzer kann zunächst aus einer Übersicht von Vorgängen einen Vorgang auswählen. Er erkennt dann an dem markierten Knoten den aktuellen Zustand des Vorgangs und kann sich zudem detailliertere Informationen über den bisherigen Verlauf und die aktuelle

Bearbeitungssituations anzeigen lassen.

Auf einer ähnlichen Abstraktionsebene, allerdings mit einem anderen Fokus, können in einem Fenster textuelle Angaben über den gesamten Vorgang gemacht werden (vgl. Abb. 75). Dazu zählen: auslösendes Ereignis, minimaler und maximaler Zeitbedarf und organisatorische Zuständigkeiten.

Ein Teilvorgang, der eine Interaktion des Bearbeiters mit dem DV-System vorsieht.

Ein Teilvorgang, der ohne jede Computerunterstützung abläuft.

Ein Teilvorgang, der selbst als Vorgang modelliert ist.

Ein vollständig automatisierter Teilvorgang.

Der Zustand einer virtuellen Vorgangsmappe.

Abb. 73: Symbole zur grafischen Darstellung von Geschäftsprozessen.

Um die Zustände der Vorgangsmappe und die Teilvorgänge detaillierter zu beschreiben, stehen zwei weitere Fenster zur Verfügung. In ihnen werden die Bezeichner der Zustände und Teilvorgänge gelistet. Für Teilvorgänge können Angaben wie organisatorische Zuständigkeit und geschätzter Zeitbedarf gemacht werden. Außerdem kann für die drei Kategorien "Objekt", "Informationsträger" und "Person/Institution" jeweils eine Liste angelegt werden. In ähnlicher Weise können für ausgewählte Zustände des virtuellen Vorgangsdokuments Objekte, Formulare und Anlagen in Listen verzeichnet werden. In beiden Fällen wird durch die Selektion eines Listenelements ein Fenster in den Vordergrund geholt, in dem die oben skizzierten Angaben (Auswahl von Dienste, Medien etc.) gemacht werden können. Die einem Zustand zugeordneten Informationen werden auch in dem durch ihn ausgelösten Teilvorgang dargestellt[1] - können dort allerdings nicht gelöscht werden. Abbildung 74 zeigt die Fenster, die auf den verschiedenen Abstraktionsebenen für textuelle Angaben zur Verfügung stehen, im Zusammenhang. Abbildung 75 veranschaulicht das Zusammenwirken der Textfenster mit dem Grafik-Editor.

1. Dabei werden Formulare und Anlagen zur Kategorie "Informationsträger" zusammengefaßt.

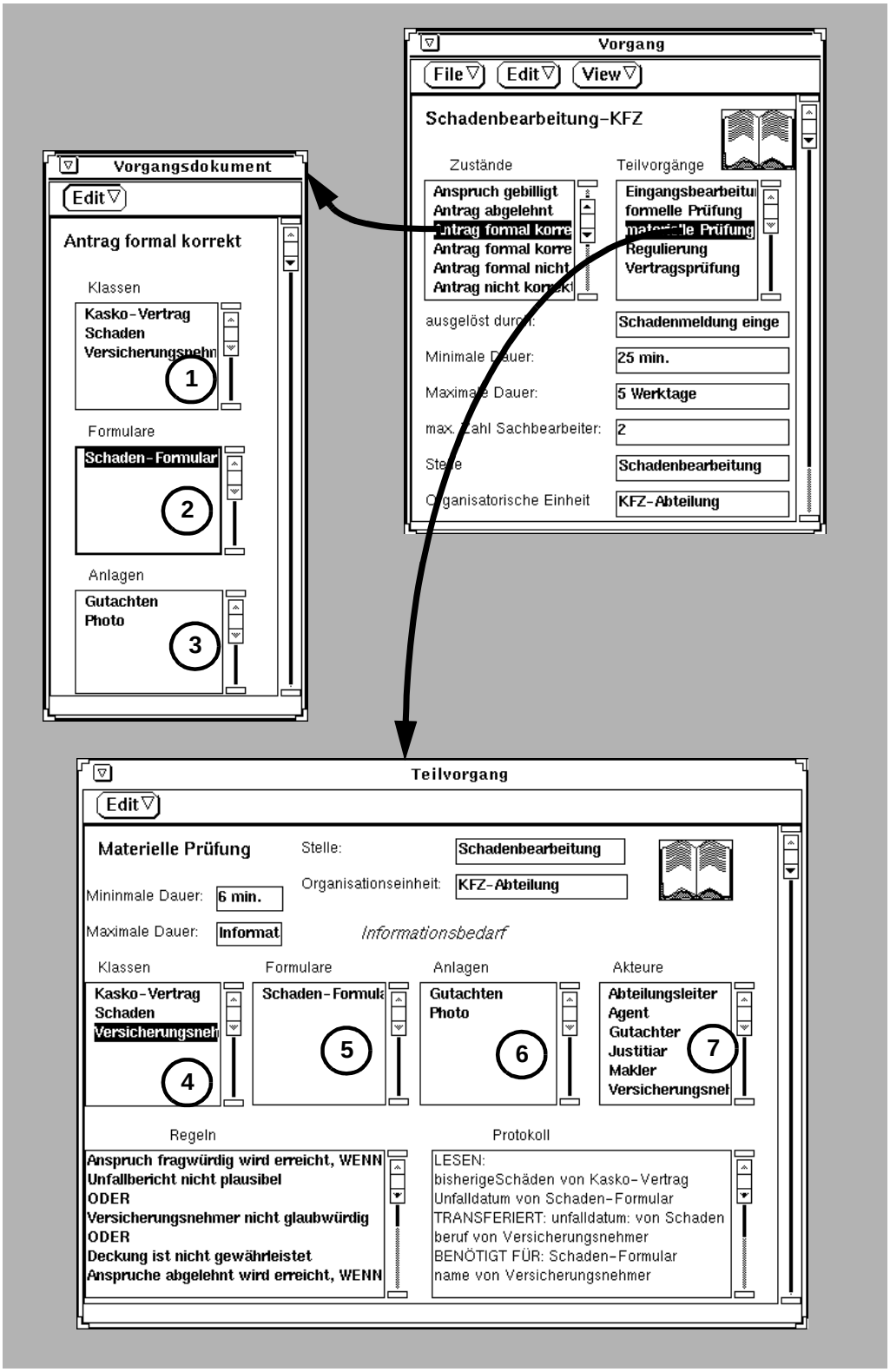
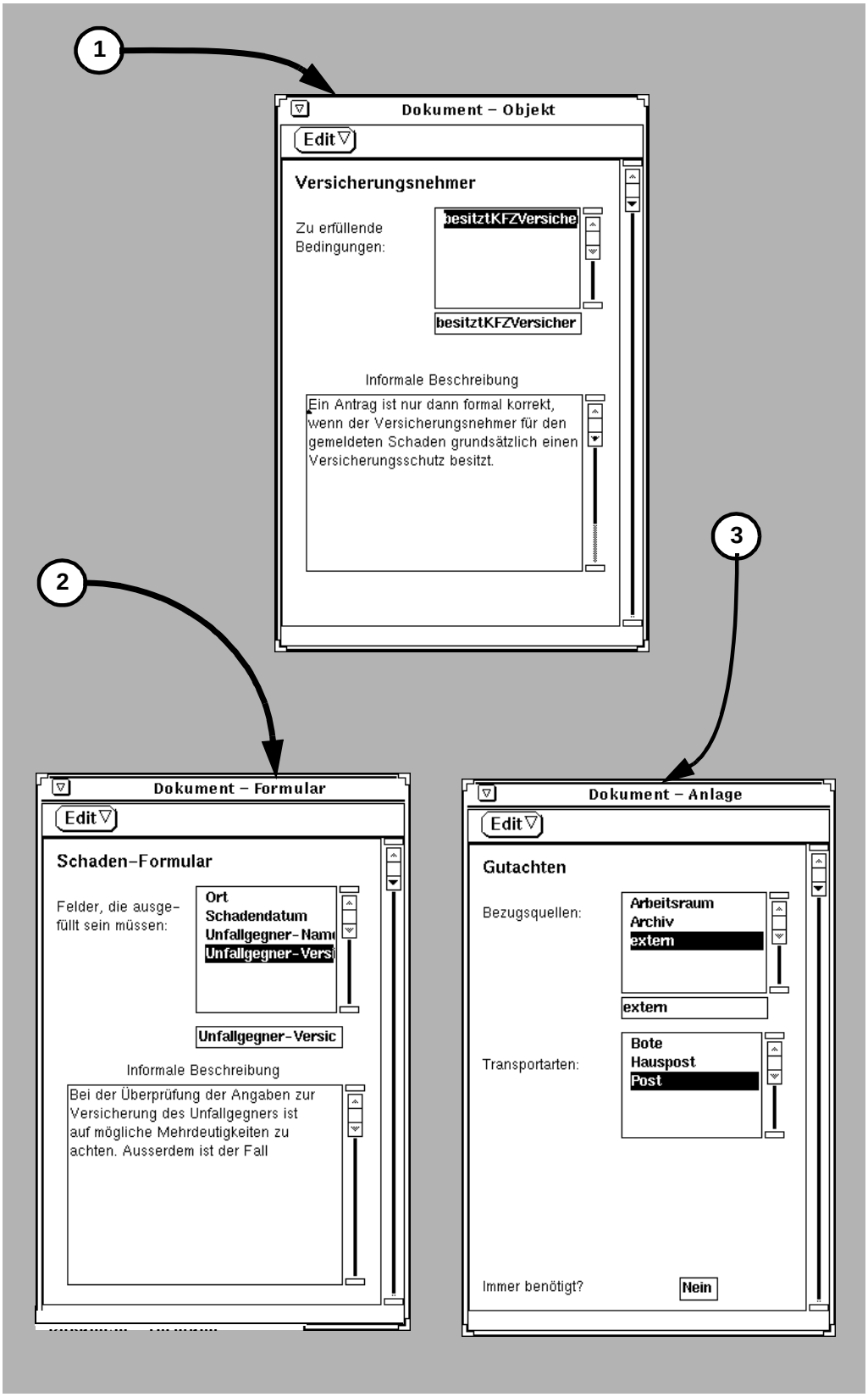
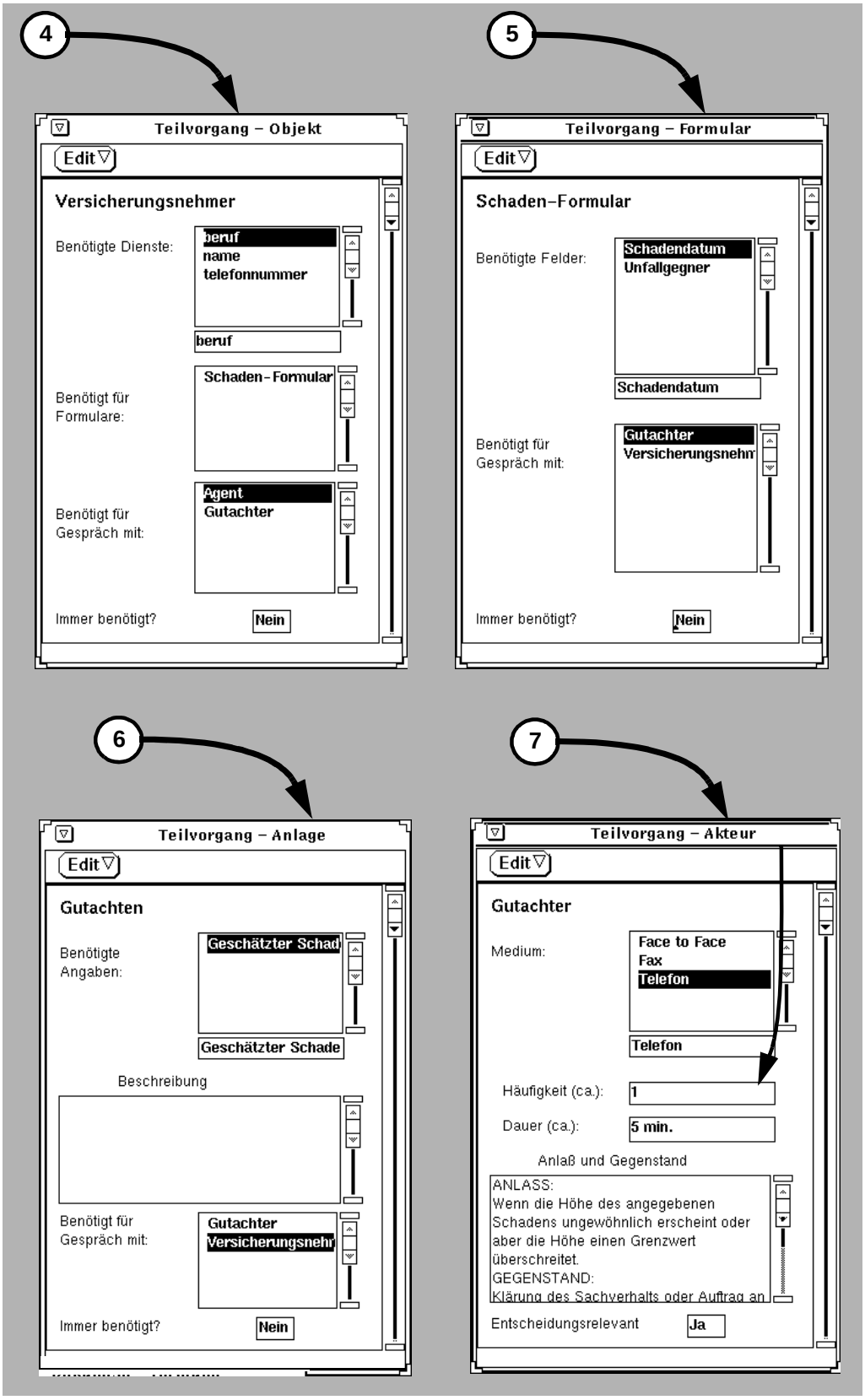


Abb. 74: Die verschiedenen Abstraktionsebenen der textuellen Beschreibung (Fortsetzung auf den folgenden Seiten)


1
Dokument – Objekt
Edit
Versicherungsnehmer
Zu erfüllende Bedingungen:
besitztKFZVersiche
besitztKFZVersicher
Informale Beschreibung
Ein Antrag ist nur dann formal korrekt, wenn der Versicherungsnehmer für den gemeldeten Schaden grundsätzlich einen Versicherungsschutz besitzt.
2
Dokument – Formular
Edit
Schaden-Formular
Felder, die ausgefüllt sein müssen:
Ort
Schadendatum
Unfallgegner-Name
Unfallgegner-Versi
Unfallgegner-Versic
Informale Beschreibung
Bei der Überprüfung der Angaben zur Versicherung des Unfallgegners ist auf mögliche Mehrdeutigkeiten zu achten. Ausserdem ist der Fall
3
Dokument – Anlage
Edit
Gutachten
Bezugsquellen:
Arbeitsraum
Archiv
extern
extern
Transportarten:
Bote
Hauspost
Post
Immer benötigt?
Nein


4
Teilvorgang – Objekt
Edit
Versicherungsnehmer
Benötigte Dienste:
beruf
name
telefonnummer
beruf
Benötigt für Formulare:
Schaden-Formular
Benötigt für Gespräch mit:
Agent
Gutachter
Immer benötigt?
Nein
5
Teilvorgang – Formular
Edit
Schaden-Formular
Benötigte Felder:
Schadendatum
Unfallgegner
Schadendatum
Benötigt für Gespräch mit:
Gutachter
Versicherungsnehm
Immer benötigt?
Nein
6
Teilvorgang – Anlage
Edit
Gutachten
Benötigte Angaben:
Geschätzter Schad
Geschätzter Schade
Beschreibung
Benötigt für Gespräch mit:
Gutachter
Versicherungsnehr
Immer benötigt?
Nein
7
Teilvorgang – Akteur
Edit
Gutachter
Medium:
Face to Face
Fax
Telefon
Telefon
Häufigkeit (ca.):
1
Dauer (ca.):
5 min.
Anlaß und Gegenstand
ANLASS:
Wenn die Höhe des angegebenen Schadens ungewöhnlich erscheint oder aber die Höhe einen Grenzwert überschreitet.
GEGENSTAND:
Klärung des Sachverhalts oder Auftrag an
Entscheidungsrelevant
Ja

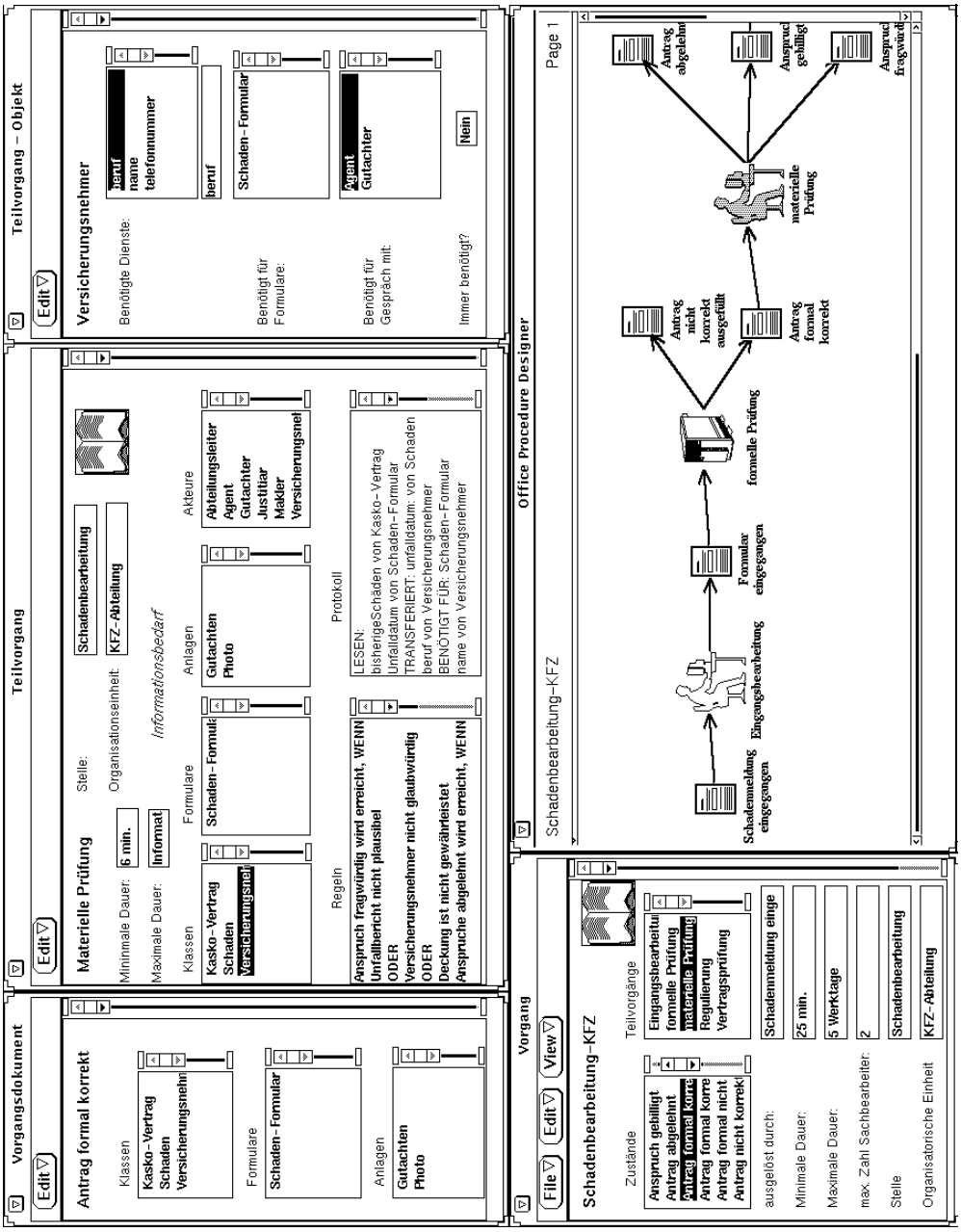


Abb. 75: Die gemeinsame Darstellung verschiedener Sichten des OPD

4. Der Value Chain Designer

Die strategische Perspektive ist einerseits durch ein höheres Abstraktions- oder Aggregationsniveau als die beiden anderen Hauptperspektiven gekennzeichnet: Nicht einzelne Objekte der Realität stehen in Vordergrund, sondern aggregierte Größen. Das Interesse ist weniger auf Details als vielmehr auf den Blick für das Ganze gerichtet. Andererseits - und dieser Umstand markiert die wesentliche Herausforderung - sind die für die Planung zu berücksichtigenden Wirkungszusammenhänge nur rudimentär bekannt und können durchaus mehrdeutig oder gar widersprüchlich sein. In diesem Sinne unterstützt der von Porter vorgeschlagene Wertketten-Ansatz das Auffinden und die Ordnung wichtiger Aspekte. Er bietet jedoch keine konkrete Identifikation und Bewertung der in einem bestimmten Unternehmen zu beachtenden Zusammenhänge.

Vor diesem Hintergrund war der Entwurf des Value Chain Designers (VCD) vor allem an den folgenden Anforderungen orientiert:

- Das Werkzeug soll die Ordnung, Verwaltung und Analyse strategisch bedeutsamer Informationen über ein Unternehmen nach Maßgabe des Wertketten-Ansatzes von Porter unterstützen.
- Die von Porter vorgeschlagenen Beziehungen zwischen den relevanten Konzepten sind teilweise nur vage formuliert. Ihre Anwendung setzt eine unternehmensspezifische Differenzierung voraus. Der VCD sollte deshalb die Möglichkeit bieten, neue Wirkungszusammenhänge einzuführen und sie zu kategorisieren, um gegebenenfalls auch den Bezugsrahmen verfeinern zu können. Diese Anforderung ist nicht zuletzt motiviert durch die Hoffnung auf induktiven Erkenntnisgewinn: Zunächst wird der (grobe) Rahmen des Wertketten-Ansatzes zum Sammeln planungsrelevanter Informationen genutzt, um dann durch die Analyse dieser Informationen zu einer neuen, aussagekräftigeren Konzeptualisierung zu gelangen.
- Die Beschreibung der im Wertketten-Ansatz berücksichtigten Konzepte rekurriert häufig auf Konzepte der operativen Ebene - in unserem Modell also auf Konzepte der organisatorischen und der Informationssystem-Perspektive. Dadurch wird nicht nur die Integration der drei Hauptperspektiven gefördert. Darüber hinaus ergibt sich dadurch die Chance, den Aufwand zur Erstellung des strategischen Modells erheblich zu reduzieren. Das Werkzeug sollte deshalb einen komfortablen Zugriff auf die im OMD und im OPD verwalteten Elemente bieten.
- Da die Theorie der strategischen Planung im allgemeinen, der Wertketten-Ansatz im besonderen nicht hinreichen, den Entwurf und die Rekonfiguration der Wertkette eines Unternehmens im Detail anzuleiten, sollten dem Benutzer in Abhängigkeit vom Betrachtungskontext Interpretations- und Planungshilfen

in Form von Heuristiken und Beispielen geboten werden.

4.1 Die verwalteten Konzepte und ihre Verbindung zu den Elementen der anderen Komponenten

Der Bezugsrahmen für den Entwurf des VCD teilt sich in zwei Bereiche. Im ersten Bereich ist eine Abbildung des Wertketten-Ansatzes realisiert. Sie basiert im wesentlichen auf den Konzepten und Beziehungen, die in Abbildung 46 dargestellt sind. Daraus ergibt sich eine Hierarchie von Aggregationsbeziehungen (Abb. 76), durch die mit Hilfe eines Browser-Systems navigiert werden kann.

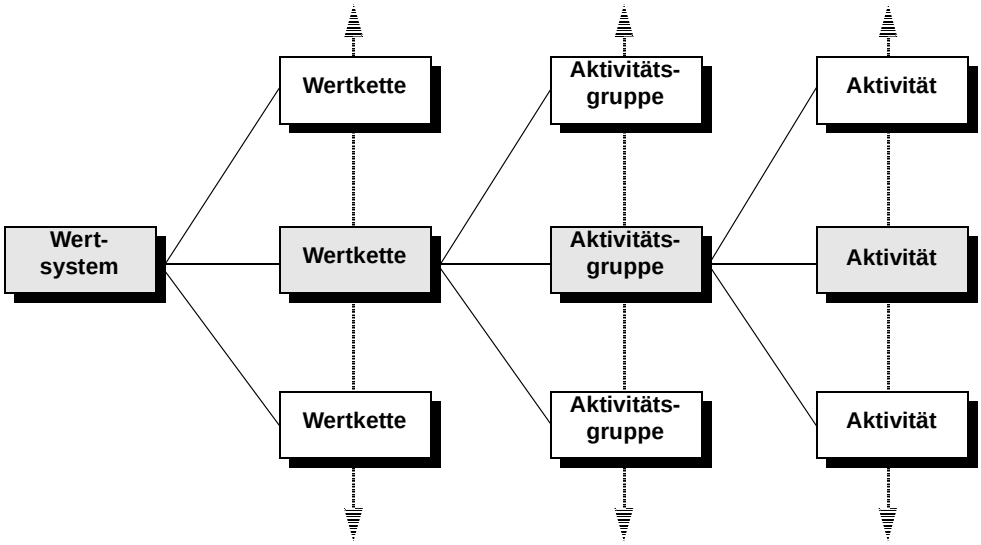


Abb. 76: Die Konzept-Hierarchie des Wertketten-Ansatzes

Im VCD kann jeweils das Wertsystem eines Unternehmens verwaltet werden. Es umfaßt die Wertkette (oder, falls es sich um ein diversifiziertes Unternehmen handelt: die Wertketten) des Unternehmens sowie die Wertketten seiner Lieferanten, seiner Distributionspartner und seiner Kunden. Angesichts des Erfassungsaufwands ist es dem Benutzer zu überlassen, wie viele Wertketten in welcher Detaillierung berücksichtigt werden. In der Regel dürfte der Fokus allerdings vor allem auf die Wertkette des eigenen Unternehmens gerichtet sein.

Für eine konkrete Instanz der in Abbildung 76 verzeichneten Konzepte kann jeweils eine Reihe von Angaben gemacht werden, wie etwa Bezeichnung, zeitliche Gültigkeit und dergleichen. Dabei wird der Benutzer angehalten, für die Bezeichnung der Aktivitätsgruppen die von Porter vorgeschlagenen Namen (in der je gewählten Landessprache) zu verwenden: Die Aktivitätsgruppen sind entsprechend vorbezeichnet und der Benutzer muß mehrfach bestätigen, daß er eine

andere Bezeichnung vorzieht. Während Aktivitäten durch die Aktivitätsgruppen, denen sie zugeordnet sind, ex ante entweder als primäre oder unterstützende Aktivitäten gekennzeichnet sind, ist jeweils noch ihre Ausrichtung ("direkt", "indirekt" oder "qualitätssichernd") anzugeben.

Im Mittelpunkt des Wertketten-Ansatzes steht die Beschreibung von Aktivitäten und der Beziehungen, durch die sie untereinander und mit anderen Elementen eines Unternehmensmodells verknüpft werden können. Dabei sind zwei Charakteristika der strategischen Perspektive von besonderer Bedeutung. So sperren sich die zu berücksichtigenden Kriterien mehr oder weniger stark gegen eine Formalisierung. Im VCD wird diesem Umstand mit Hilfe von drei alternativen Ansätzen begegnet. Zum einen ist eine schwache Formalisierung möglich, indem dem Benutzer die Möglichkeit gegeben wird, Einträge einer Ordinalskala zu definieren (wie etwa "hoch", "mittel", "gering"). Solche Skalen können bei der Initialisierung des Modells als Default vorgegeben werden, sie sind allerdings grundsätzlich variabel. Daneben können Listen von Prädikaten angelegt werden, die nicht geordnet sind (beispielsweise "kapitalintensiv", "arbeitsintensiv"). Schließlich gibt es die Möglichkeit, zur Beschreibung eines Kriteriums einen freien Text einzugeben. Dabei können einzelne Teile markiert und mit anderen Elementen (wie etwa Beispielen oder Erläuterungen in anderen Kontexten) des Modells verknüpft werden. Auf diese Weise kann etwa für eine Aktivitätsgruppe beschrieben werden, welchen Zielen sie dient, welchen Herausforderungen sie gegenübersteht etc.

Das zweite Charakteristikum betrifft die besondere Bedeutung, die Beziehungen zukommt. Dies gilt einerseits für Beziehungen zwischen spezifischen Konzepten des Wertketten-Ansatzes - also vor allem für Beziehungen zwischen Aktivitäten. Für ihre Kennzeichnung steht ein zentrales Verzeichnis mit Beziehungsnamen zur Verfügung, das mit fünf Einträgen ("unterstützt", "wird unterstützt", "konkurriert mit", "beinhaltet", "gehört zu") initialisiert ist und vom Benutzer erweitert werden kann. Bei der Zuweisung dieser Beziehungen werden gewisse Integritätsbedingungen überwacht. Dazu gehört, daß eine Aktivität sich nicht selbst unterstützen oder beinhalten darf. Auf der anderen Seite gibt es die Möglichkeit, Beziehungen zu Konzepten anzulegen, die im OMD oder im OPD verwaltet werden. Dazu kann einerseits über einen entsprechenden Browser auf alle Klassen des Objektmodells zugegriffen werden, andererseits können Geschäftsprozesse ausgewählt werden. Die Namen der Beziehungen können vom Benutzer frei vergeben werden. Er wird durch den Umweg über das zentrale Verzeichnis lediglich darin unterstützt, unnötige Mehrfachbezeichnungen zu vermeiden. Um die Bedeutung der Beziehungen zu beschreiben, können sie einerseits - da auf dieser Abstraktionsebene Kardinalitäten keine große Akzeptanz erwarten lassen - als optional oder obligatorisch gekennzeichnet werden, andererseits kann ein Hyper-

text-Kommentar abgelegt werden.

Eine zentrale mit der Detaillierung eines Unternehmensmodells verbundene Herausforderung ist darin zu sehen, die Aggregation von Größen, die im OMD oder im OPD verwaltet werden, auf ein in der strategischen Perspektive präferiertes Abstraktionsniveau zu beschreiben. Dazu kann der Benutzer in einem zentralen Verzeichnis Kategorien von Gütern und Leistungen anlegen. Sie können dann entweder als Ressourcen oder als produzierte Leistungen interpretiert werden. Das Verzeichnis ist mit einer Reihe von Kategorien initialisiert. Für jede Kategorie können dann die Klassen im Objektmodell (wie etwa die verschiedenen Ressourcenarten, die in der organisatorischen Perspektive modelliert sind) ausgewählt werden, die ihr zugeordnet sind. Beispiel: "Human-Ressource" entspricht der Klasse "Mitarbeiter" und ihrer Unterklassen. In diesem Zusammenhang ist nicht zuletzt an die Verknüpfung von Konzepten, die in den Unterstützungsaktivitäten "Infrastruktur" und "Forschung und Entwicklung" ("technology development") verwendet werden, mit denen der Informationssystem-Ebene zu denken.

Indem korrespondierende Konzepte auf den verschiedenen Ebenen eines Unternehmensmodells in Beziehung gesetzt werden, ist natürlich noch nicht beschrieben, ob und in welcher Weise von den Instanzen einer Ebene auf die einer anderen (hier: der strategischen Ebene) aggregiert werden kann. Der VCD bietet dazu gegenwärtig die Möglichkeit, folgende Aggregationsbeziehungen zu beschreiben:

- Ein übergeordnetes Objekt kann dadurch gekennzeichnet werden, daß es aus allen Instanzen einer oder mehrerer ausgewählter Klassen (und gegebenenfalls deren Unterklassen) besteht. Beispiel: "Human-Ressource" besteht aus allen Instanzen der Klasse Mitarbeiter und ihrer Unterklassen.
- Die Menge der Instanzen ausgewählter Klassen kann auf solche mit bestimmten Eigenschaften reduziert werden (beispielsweise Mitarbeiter, die einen technischen Beruf haben). Dabei wird unterstellt, daß diese Eigenschaften mit Hilfe boolescher Dienste überprüft werden können.
- Der Wert eines Attributs eines übergeordneten Objekts kann als Summe oder Durchschnitt der Werte eines Attributs aller oder ausgewählter Instanzen einer oder mehrerer Klassen (sowie gegebenenfalls ihrer Unterklassen) definiert werden. Beispiel: der Wert des Attributs "Kosten pro Jahr" von "Human-Ressource" ergibt sich als Summe der Werte des Attributs "Gesamtkosten pro Jahr" aller Instanzen der Klasse "Mitarbeiter" und ihrer Unterklassen.

Bei der Aggregation sind gewöhnlich Integritätsbedingungen zu berücksichtigen, die der VCD nur rudimentär überwacht: Wenn ein Objekt zugeordnet wird, das bereits als Bestandteil eines Aggregats zugeordnet wurde, wird der Benutzer darauf aufmerksam gemacht. Ein anderer wichtiger Fall kann nicht überwacht werden: Wenn ein Geschäftsprozeß als Bestandteil einer Aktivität gekennzeichnet

ist, sollte es eigentlich nicht möglich sein, die durch ihn verursachten Kosten doppelt zuzuordnen - über die Kosten für den Geschäftsprozesses selbst und durch die Zuordnung von Ressourcen, die von ihm benutzt werden. Eine solche Verzerrung kann nicht ausgeschlossen werden, da eine Ressource einer Aktivität auch unabhängig von einem Geschäftsprozeß zugeordnet werden könnte.

Daneben ist eine Reihe komplexerer Aggregationsbeziehungen denkbar: durch eine Gewichtung der zu aggregierenden Größen oder durch Regeln für die Aggregation nicht intervall-skalierter Größen. Solche Beziehungen können im VCD zur Zeit nicht formal definiert werden, da die dazu letztlich notwendige Verwendung einer formalen Sprache für die strategische Perspektive nicht geeignet erscheint. Stattdessen können solche Zusammenhänge in den Kommentaren, die zu den Beziehungen angelegt werden, beschrieben werden.

Die Beziehungen zwischen den Elementen der strategischen und denen der beiden anderen Ebenen - das gilt vor allem für Aggregationsbeziehungen - machen deutlich, daß die Modellierung der einzelnen Ebenen in wechselseitiger Abstimmung erfolgen sollte. So werden bestimmte Größen, die auf der strategischen Ebene von Bedeutung sind, erst dazu führen, die für ihre Aggregation nötigen Eigenschaften auf anderen Ebenen einzuführen. Daneben ist es sinnvoll, die Zugehörigkeit von Objekten zu bestimmten strategisch bedeutsamen Kategorien durch die Einführung entsprechender Dienste (die dann jeweils einen booleschen Wert zurückliefern) auf Objektebene prüfen zu können. Falls der Aufwand zur Ermittlung von Aggregationsgrößen aus anderen Größen im Modell zu groß erscheint, erlaubt der VCD die Eingabe von Schätzwerten.

Zusammenfassend kann festgestellt werden, daß der VCD - dem theoretischen Gehalt des Wertketten-Ansatzes entsprechend - einen erweiterbaren Rahmen zur Verwaltung von Konzepten bietet, deren Bedeutung nur in Teilen formalisiert ist. Damit bietet er die Chance, durch die Analyse von auf diese Weise ausführlich beschriebenen Wertsystemen zu neuen Konzepten und Zusammenhängen und damit: zu restriktiveren Integritätsbedingungen zu gelangen und damit die Funktionalität des VCD anzureichern. Die weniger strukturierte und formalisierte Beschreibung von Konzepten im VCD wird deutlich, wenn man die Modellierung der Beziehungen von Aktivitäten zu Objekten bzw. Klassen (Abb. 77) mit der Modellierung von Beziehungen im Objektmodell vergleicht.

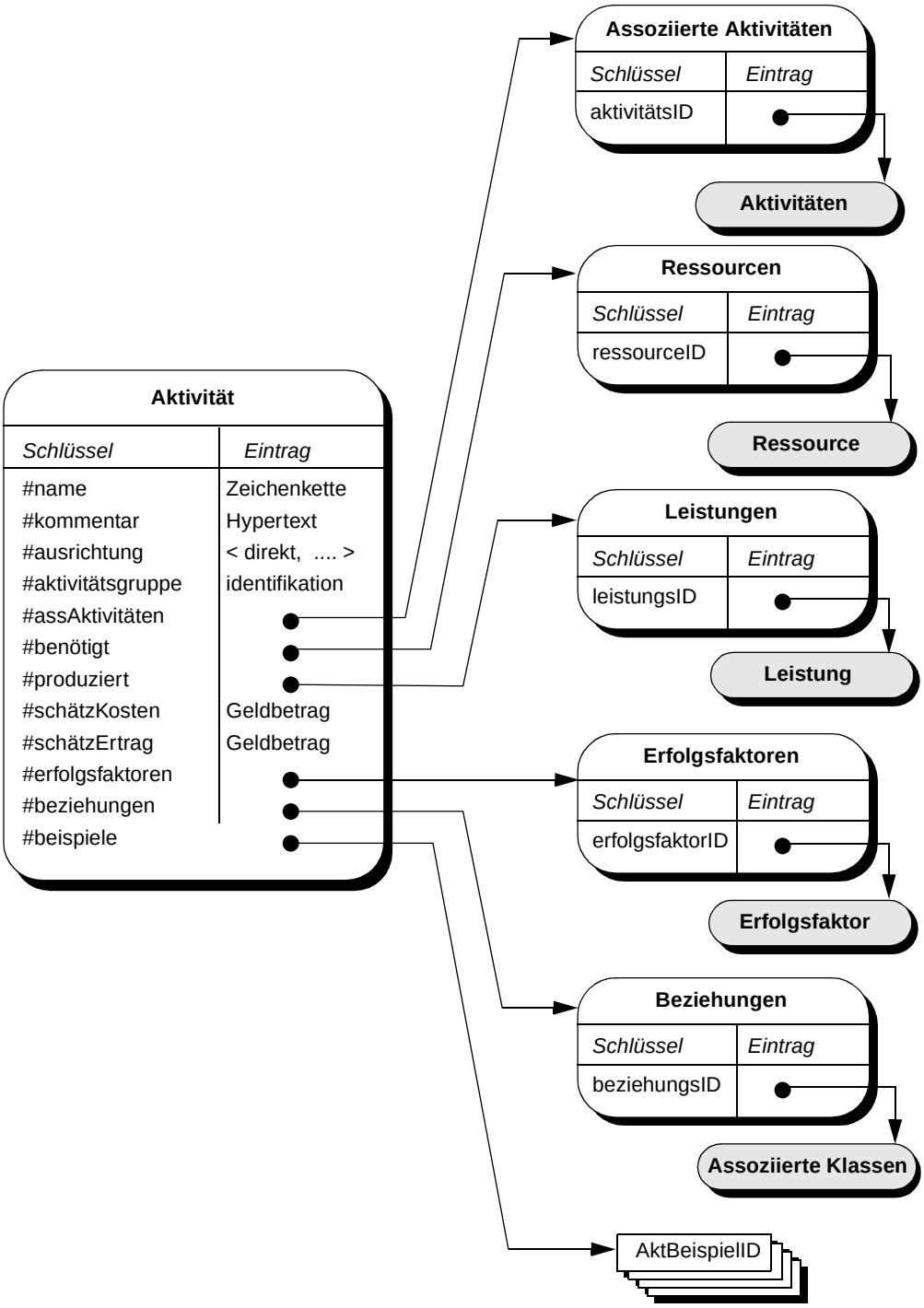


Abb. 77: Aufbau des Verzeichnisses zur Abbildung einer Aktivität. Die schraffierten Symbole repräsentieren weitere Verzeichnisse, so daß sich - ähnlich wie bei der Abbildung des Objektmodells - ein Geflecht von Verzeichnissen ergibt.

4.2 Unterstützung der Analyse eines Wertsystems

Neben der Verdeutlichung gegebener Zusammenhänge soll ein Wertsystem den Planungsprozeß dadurch unterstützen, daß die systematische Entwicklung von Alternativen und die Erfassung deren möglicher Auswirkungen erleichtert werden. Ein erster Ansatz dazu ist die teilweise Modifikation von Wertketten. Auch wenn der VCD in der gegenwärtigen Implementierung nur wenige Integritätsbedingungen überwacht, ist er der vielfältigen Zusammenhänge wegen für einen solchen Trial and Error-Prozeß sicher besser geeignet als einfache Präsentationsmedien wie Papier oder Wandtafel. So können beispielsweise die Ergebnisse von Aggregationen (sofern das zugehörige Objektmodell dazu in hinreichender Weise beschrieben ist) mit denen von Schätzungen verglichen werden. Aggregationen können dabei schrittweise verändert werden. Außerdem kann sehr schnell festgestellt werden, welche Aktivitäten vom Löschen einer Aktivität (beispielsweise wenn diese Aktivität ausgelagert werden soll) unmittelbar betroffen wären.

Daneben bietet der VCD dem Planer für die Analyse und Neugestaltung eines Wertsystems im wesentlichen drei Hilfen:

- Checklisten
- Beispiele
- Retrievalverfahren

Um die Analyse und gegebenenfalls die Modifikation einer Wertkette anzuleiten, ist in der gegenwärtigen Version eine an Porters Lehrbuch angelehnte Heuristik mit Hilfe von Browsern und Hypertext implementiert. Der Benutzer wählt dazu eine von drei generischen Strategien und navigiert dann durch eine hierarchische Checkliste. Die Einträge der einzelnen Listen können dabei auf weitere Listen verweisen und/oder auf Hypertext-Annotationen. Abbildung 78 zeigt ein Beispiel für eine solche Hierarchie - in Anlehnung an Porter (1985, S. 100 ff.). Die Listeneinträge können vom Benutzer verändert oder gelöscht werden. Zudem können neue Einträge vorgenommen werden.

Beispiele können einerseits unmittelbar in den Hypertext-Annotationen dargestellt werden, andererseits besteht die Möglichkeit, auf beispielhaft instanzierte Wertsysteme oder Wertketten zu verweisen. Diese Beispiele sind persistent abgelegt und können in beliebiger Zahl (solange keine Hardware-Restriktionen eintreten) in das Image geladen werden. Sie werden an der Benutzerschnittstelle in der gleichen Weise wie das jeweils aktuelle Wertsystem (oder die Wertkette) präsentiert und können in gleicher Weise exploriert werden. Der Benuter kann Beispiele verändern und neue Beispiele anlegen. Dabei ist zu berücksichtigen, daß für jedes Beispiel individuelle Namenskonventionen gelten. Wenn also in einem Beispiel die Bezeichnung für eine Beziehung geändert wird, ist davon nur das Beispiel selbst betroffen.

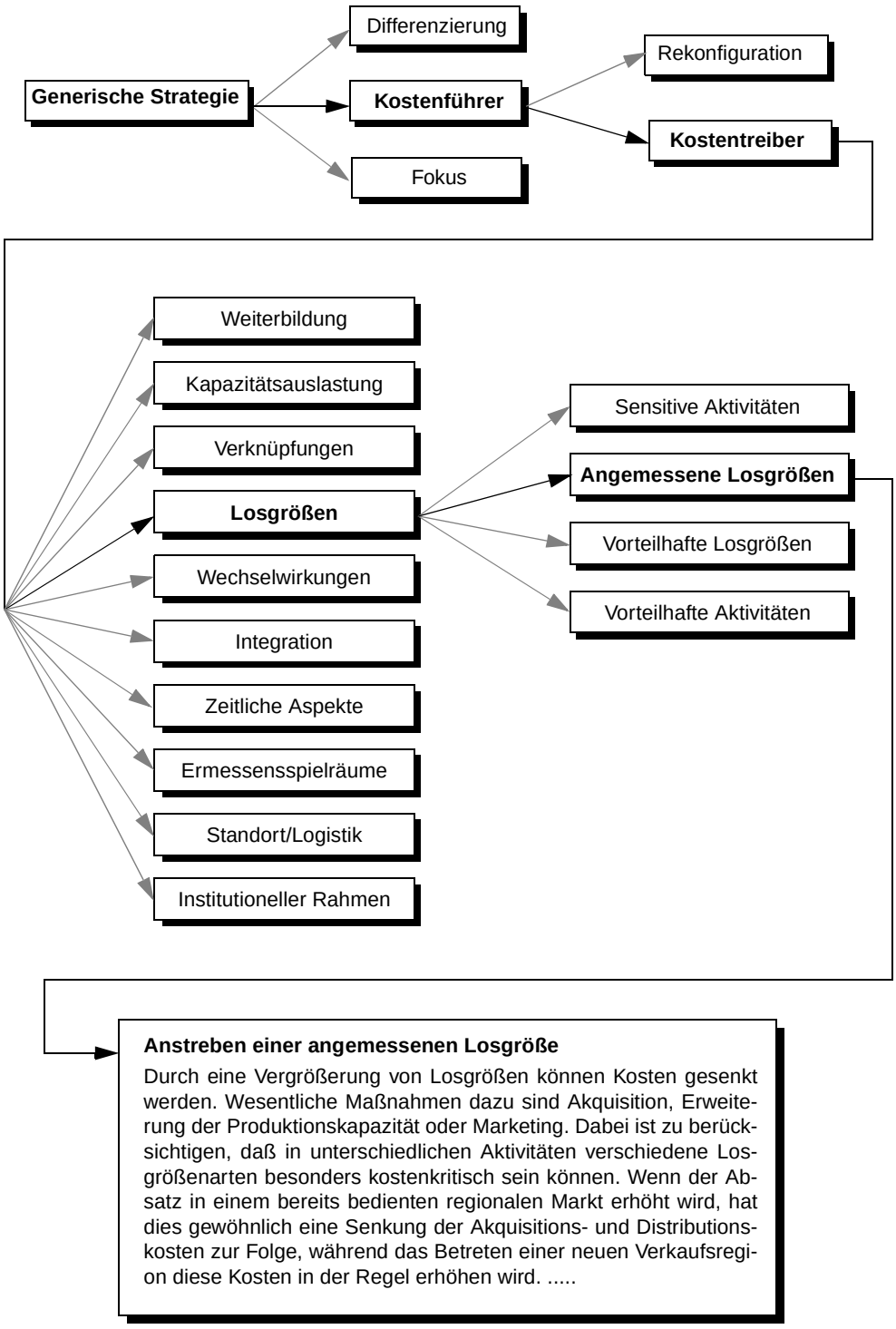


Abb. 78: Ausschnitt aus einer Hierarchie von Checklisten und Textannotationen, wie sie imValue Chain Designer instanziert ist.

Um das Navigieren durch ein Wertsystem und die Suche nach bestimmten Bestandteilen zu unterstützen, sind im VCD verschiedene Retrievalverfahren implementiert. Sie werden zum Teil als Browser verfügbar gemacht. Eine Gruppe von Retrievalverfahren setzt an den innerhalb eines Wertsystems definierten Beziehungen an. Sie erlauben unter anderem die Ermittlung der Aktivitäten, die mit einer vorgegebenen Aktivität in irgendeiner oder aber in einer bestimmten Weise assoziiert sind. Andere Verfahren zielen darauf, die Suche nach Elementen der organisatorischen und Informationssystem-Perspektive zu unterstützen. So kann für jede Aktivität ermittelt werden, welche Geschäftsprozesse und Organisationseinheiten ihr zugeordnet sind oder welche Bestandteile des Informationssystems von ihr unmittelbar benötigt werden.[1]

Angesichts des hohen Stellenwerts nicht formalisierter Information spielen Volltext-Retrievalverfahren eine besondere Rolle. Dazu werden alle Hypertexte vollständig invertiert (also ohne Stoppwörter). Suchausdrücke können durch logische Verknüpfung einzelner Zeichenketten (die rechts trunkiert sein können) gebildet werden. Durch die Verwaltung der Wörter in einem Binärbaum ist eine schnelle Suche gewährleistet. Von den spezifischen Problemen der Volltext-Suche in Hypertexten wurde allerdings abstrahiert. Da die Wörter des Suchbaums jeweils nur auf die Knoten verweisen, in denen sie explizit vorkommen, liefert das Verfahren eben nur solche Knoten, die selbst die Bedingungen des Suchausdrucks erfüllen. Nehmen wir beispielsweise an, es soll nach einem Text gesucht werden, in dem sowohl "Outsourcing" als auch "Informationsmanagement" vorkommen. Wenn der erste Begriff in einem Knoten enthalten ist, der einen Verweis auf einen anderen Knoten beinhaltet (beide Knoten also für eine bestimmte Sichtweise eine logische Einheit bilden), in dem der zweite Begriff vorkommt, wird das Suchverfahren hier zu keinem Treffer führen.

Da ein Wertsystem im Zeitverlauf eine Fülle von Texten beinhalten kann, mag es von Interesse sein, die dabei verwendete Begrifflichkeit näher zu betrachten. Der VCD bietet zur Zeit eine rudimentäre Unterstützung solcher Analysen: Der Benutzer kann sich den Binärbaum der invertierten Texte als Cross-Reference-Liste ausgeben lassen. Eine solche Liste gibt Aufschluß über die Häufigkeit der einzelnen Begriffe, die Verwendung von Synomyma und gegebenenfalls die Kontextabhängigkeit der Terminologie - sofern einzelne Hypertextknoten sich einem bestimmten Kontext zuordnen lassen.

1. Der Umfang solcher Retrievalmöglichkeiten hängt natürlich davon ab, ob in in welchem Maß entsprechende Beziehungen modelliert wurden.

4.3 Die Benutzerschnittstelle

Die Benutzerschnittstelle des VCD ist darauf ausgerichtet, die Konzepte des Wertketten-Ansatzes in anschaulicher Weise zu vermitteln. Dabei sind die Prinzipien zu beachten, die auch für den OMD und den OPD angewandt wurden - also grundsätzlich die Berücksichtigung verschiedener Abstraktionsebenen und - konkreter: einander ergänzende textuelle und grafische Präsentationen. Die höchste Abstraktionsstufe bildet die Abbildung eines Wertsystems als einer Liste von Wertketten. Die Auswahl einer Wertkette richtet den Fokus auf eine detailliertere Darstellung derselben. Die damit verbundene Abstraktionsebene, also die überblickartige Darstellung einer Wertkette, bildet - wie auch in Porters Lehrbuch - das Zentrum, den Ausgangspunkt für die Arbeit mit dem VCD.

In Anlehnung an Porter (vgl. Abb. 46) wird eine Wertkette als eine Menge von Aktivitätsgruppen dargestellt. Jede Gruppe wird durch die Liste der ihr zugeordneten Aktivitäten repräsentiert. Im Unterschied zu Porters Darstellung werden Unterstützungsaktivitäten nicht in Form von Balken repräsentiert, die in ihrer horizontalen Ausdehnung jeweils die Reihe aller primären Aktivitäten überspannen: Sie könnten sonst nicht in einer Listbox dargestellt werden.[1] Die Beziehungen zwischen Aktivitäten können zudem grafisch visualisiert werden (vgl. Abb. 79). Dazu wird auf eine leicht modifizierte Version des im OMD zur Darstellung von Beziehungen verwendeten Grafik-Editors zurückgegriffen. Mit Hilfe eines weiteren Grafik-Editors kann die Verteilung von Kosten, Erträgen und Dekkungsbeiträgen auf die einzelnen Aktivitäten dargestellt werden (soweit die entsprechenden Angaben zugeordnet wurden). Dabei kann zwischen der Darstellung absoluter und relativer Werte gewählt werden.

Durch die Auswahl einer Aktivität in einer Listbox wird ein Fenster in den Vordergrund gerückt, in dem generelle Angaben über die Aktivität verzeichnet sind. Dazu gehören geschätzte Kosten und Erträge sowie - gegebenenfalls - die gleichen durch Aggregation ermittelten Größen und die jeweils daraus abgeleiteten Deckungsbeiträge. Außerdem sind Listen mit den assoziierten Aktivitäten, Ressourcen, Leistungen und Geschäftsprozessen dargestellt. Die Auswahl eines Elements aus einer dieser Listen rückt ein Fenster zur näheren Beschreibung dieses Elements in den Vordergrund. Dabei spielen Ressourcen und Leistungen eine besondere Rolle, weil zu ihrer näheren Kennzeichnung auf eine weitere Detaillierungsebene verzweigt werden kann: Eine Ressource (wie auch eine Leistung) kann als Aggregation von Objekten (oder deren Eigenschaften), die im OMD verwaltet werden, beschrieben werden.

1. In einer früheren Version des VCD (vgl. Frank/Klein 1992 b, S. 27) ist die Darstellung stärker an Porter angelehnt. Dabei sind die einzelnen Aktivitätsgruppen allerdings nur als Buttons dargestellt, die Wertkette bietet also keinen Überblick über die in ihr enthaltenen Aktivitäten.

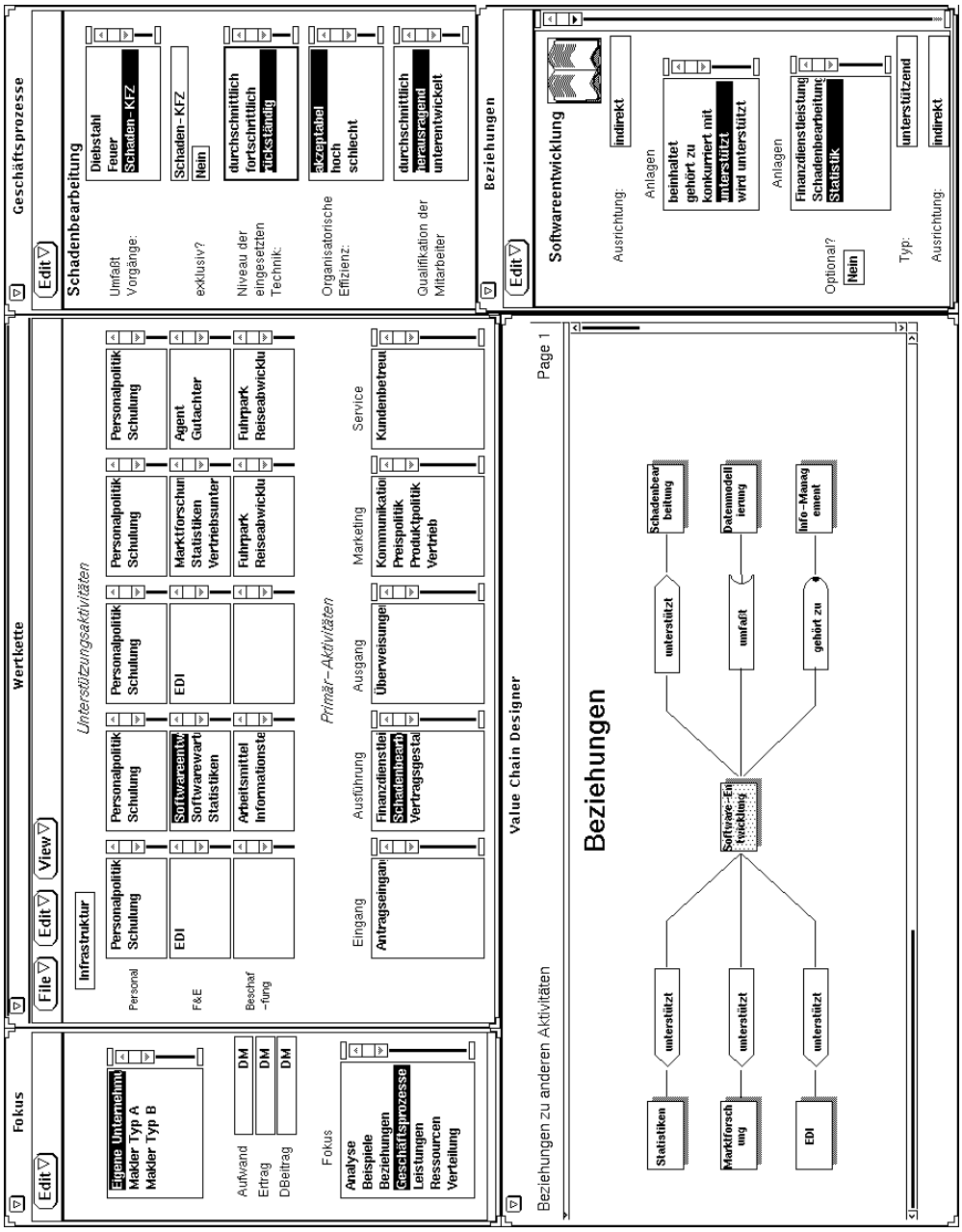


Abb. 79: Die gemeinsame Darstellung verschiedener Sichten des VCD

An einzelnen Stellen kann es für den Benutzer von Interesse sein, in den OMD oder den OPD zu verzweigen. Wenn beispielsweise die im VCD vorgesehene Beschreibung eines Geschäftsprozesses als unzureichend empfunden wird, kann die grafische Darstellung dieses Prozesses mit Hilfe des dedizierten Grafik-Editors des OPD veranlaßt werden. Die dazu nötige Kommunikation zwischen den

Komponenten wird dadurch realisiert, daß jede Komponente (genauer: das sie repräsentierende Objekt) Referenzen auf die anderen Komponenten enthält. Auf diese Weise können sich die Komponenten Nachrichten zuschicken und grundsätzlich den gesamten Vorrat an Diensten einer anderen Komponente nutzen.

Die gemeinsame Nutzung der drei Komponenten führt dazu, daß eine größere Zahl von Fenstern (die Zahl ist variabel, es kann allerdings in der Regel von ungefähr 25 Fenstern ausgegangen werden) zu verwalten ist. Um die Übersichtlichkeit der Benutzerschnittstelle zu fördern, sind deshalb Maßnahmen wichtig, die dazu beitragen, die in einem bestimmten Arbeitskontext relevanten Fenster in den Vordergrund zu holen. Im VCD sind dazu zwei Möglichkeiten vorgesehen. So werden einige Fenster durch bestimmte Dialogaktivitäten sichtbar gemacht (wenn beispielsweise ein Attributname in einer Liste ausgewählt wird, erscheint das Fenster zur Bearbeitung der Attributeigenschaften im Vordergrund). Daneben gibt es Funktionen, die es dem Benutzer erlauben, gezielt einzelne Fenster oder die Fenster einer bestimmten Komponente in den Vordergrund zu holen.

5. Die Verwendung der Werkzeuge im Lebenszyklus von Informationssystemen

Der Einsatz der dargestellten Entwicklungsumgebung ist mit einer Reihe von Zielen verknüpft. So soll sie unter anderm dazu beitragen, die Friktionen zwischen den verschiedenen Phasen im Lebenszyklus betrieblicher Informationssysteme zu überwunden. Daneben soll sie die Berücksichtigung möglichst aller relevanten Aspekte und deren Beziehungen unterstützen. Es liegt auf der Hand, daß der Einsatz der Werkzeuge allein nicht garantiert, solche ambitionierten Ziele zu erreichen. Vielmehr ist eine sachkundige Nutzung der Werkzeuge vor dem Hintergrund spezifischer Anforderungen und - natürlich - die Kompetenz der Beteiligten zur Analyse und Konzeptualisierung des jeweils relevanten Realitätsausschnitts gefordert.

Auch wenn zwischen den drei hier differenzierten Perspektiven deutliche Interdependenzen bestehen, steht die Betrachtung und Analyse der strategischen Optionen eines Unternehmens idealtypisch am Beginn der Gestaltung von Unternehmensmodellen. Erst wenn eine für die Ziele der organisatorischen Gestaltung hinreichende Detaillierung der Wertkette erreicht ist, empfiehlt sich die Modellierung der beiden anderen Ebenen - die dann wiederum rückkoppelnd Hinweise auf die weitere Konkretisierung der strategischen Perspektive liefern mag. Es wird allerdings nicht immer sinnvoll sein, eine umfangreiche und detaillierte Beschreibung und Analyse strategischer Orientierungen durchzuführen: Bestehende Strategien mögen als hinreichend und leicht vermittelbar gelten oder der Aufwand für eine detaillierte Erfassung im VCD als unzumutbar hoch erscheinen.

Auch für den Einsatz des OMD und des OPD ist zu berücksichtigen, in welchem Umfang Organisation und Informationssystem zur Disposition stehen. Hier ist eine Fülle verschiedenster Konstellationen denkbar. Um eine Vorstellung davon zu vermitteln, wie die Phasen des Lebenszyklus ausgefüllt werden können und wie sich deren Integration darstellt, unterscheiden wir im folgenden vereinfachend zwei Ausgangssituationen. Die erste sieht vor, daß ein Informationssystem vollständig neu entworfen und implementiert wird. Dabei kann allerdings auf existierende Referenzmodelle und bereits implementierte Klassen zurückgegriffen werden. In der zweiten - sicherlich realistischeren Situation - sind bestehende Anwendungen zu erhalten. Das Ziel der Modellierung ist also neben der Erweiterung eines bestehenden Informationssystems die Vorbereitung geeigneter Reengineering-Maßnahmen zur schrittweisen Migration in ein objektorientiertes System. Das Ausmaß der Organisationsanalyse kann in beiden Fällen in einer großen Bandbreite variieren.

Die Unterscheidung verschiedener Phasen dient der gedanklichen Strukturierung. Sie ist einerseits dadurch zu relativieren, daß durch die Verwendung gleicher Konstrukte die Übergänge so reibungslos verlaufen können, daß eine eindeutige Grenzziehung schwierig wird. Die Einteilung in sechs Phasen (wie auch deren Bezeichnung) reflektiert zudem ein subjektiv bestimmtes Detaillierungsniveau, das auch anders gewählt werden könnte. Andererseits ist daran zu denken, daß es vielfältige Rückkopplungen zwischen den Phasen geben kann. In Abbildung 80 sind die Beziehungen zwischen den Phasen und ihre Zuordnung zu den Komponenten der Entwicklungsumgebung dargestellt.

Phase 0: Erfassung und Analyse strategischer Rahmenbedingungen

Mit Hilfe von Interviews, der Auswertung von Dokumenten und sonstigen Erhebungstechniken sind die relevanten Wertketten sind zu erfassen. Anschließend sind sie in den VCD abzubilden. Das werkzeuggestützte Modell dient dann als Medium für die Diskussion der bis dahin erfaßten Zusammenhänge. Die Diskussion dient der Revision des Modells und der Auswahl einer generischen Strategie. Neben einer mehr oder weniger detaillierten Charakterisierung von Wertaktivitäten können zudem strategische Optionen entworfen und analysiert werden. Es wird also zunächst ausschließlich der VCD genutzt.

Phase 1: Domänenerfassung

In dieser Phase werden - wiederum unter Rückgriff auf geeignete Analyse- bzw. Erhebungstechniken - zunächst fachsprachliche Beschreibungen des jeweils relevanten Realitätsausschnitts gesammelt. Im Vordergrund stehen dabei Fragen nach den zu verrichtenden Aufgaben, den zwischen diesen Aufgaben bestehenden Beziehungen sowie nach den jeweil benötigten Ressourcen. Diese Sammlung ist anschließend unter Mitwirkung der Domänenexperten zu analysieren.

Dabei ist einerseits auf Vollständigkeit (werden alle wesentlichen Konstellationen berücksichtig?) und Konsistenz zu achten, andererseits sollten die verwendeten Begriffe systematisiert werden. Dazu gehört die Beseitigung von Homonymen, die Kennzeichnung oder Vermeidung von Synomyen sowie die Auszeichnung wichtiger Beziehungen (wie "Oberbegriff von", "bedingt" u.ä.). In den so aufbereiteten Unterlagen sind anschließend wesentliche realweltliche Objekte, Aufgaben und Geschäftsprozesse zu identifizieren und mit Hilfe des OMD und des OPD abzubilden. Dabei stehen weniger detaillierte Objekteigenschaften als vielmehr wichtige Beziehungen zwischen Objekten bzw. Klassen im Vordergrund.

Zusätzlich bei Berücksichtigung eines vorhandenen Informationssystems: Es ist eine Liste bestehender Anwendungen und der jeweils benötigten Systembasis (Hardware, Betriebssystem, Datenbankmanagement-System und dergleichen) zu erstellen. Außerdem sind Beziehungen (wie Interaktionen oder sonstige Abhängigkeiten) zwischen Anwendungen zu ermitteln.

Phase 2: Domänenanalyse

Die Beziehungen zwischen Objekten sind zu überprüfen und gegebenenfalls neu zu ordnen oder zu erweitern. Die Eigenschaften von Klassen sind zu erfassen ohne daß dabei die Eigenschaften (wie Attribute, Dienste, Trigger, Guards) im Detail zu beschreiben sind. Geschäftsprozesse bzw. Vorgänge sind in Teilvorgänge zu unterteilen; den Teilvorgängen sind benötigte Objekte und Objekteigenschaften zuzuordnen. Dabei ist gegebenenfalls eine Erweiterung von Objekteigenschaften angebracht. Falls wiederverwendbare Unternehmensmodelle verfügbar sind, kann an dieser Stelle überprüft werden, welches Modell eventuell auf die betrachtete Domäne angewandt werden kann.

Zusätzlich bei Berücksichtigung eines vorhandenen Informationssystems: Bestehende Anwendungen sind danach zu ordnen, wie wichtig und wie aufwendig ein objektorientiertes Reegineering ist. Dazu gehören Kriterien wie: von außen benötigte Dienste, Anwendung im Quellcode vorhanden, Aufwand zur Umstellung auf Server/Client-Betrieb etc. Außerdem kann hier eine Zuordnung von Anwendungen zu Vorgängen und bereits erfaßten (Daten-) Objekten erfolgen. Dabei ist nicht zuletzt ein gegebenenfalls bestehendes Datenmodell oder Data Dictionary zu untersuchen.

Phase 3: Entwurf und Reorganisation

Auf der Grundlage der bisherigen Beschreibung von Vorgängen kann eine Organisationsanalyse durchgeführt werden, in deren Verlauf Engpässe, Medienbrüche und dergleichen betrachtet werden. Zusätzlich kann - wiederum mit Unterstützung des OPD - eine Kommunikationsanalyse durchgeführt werden. Abschlie-

ßend sollte nach einer Betrachtung organisatorischer Alternativen eine (vorläufige) Festlegung der Organisation von Geschäftsprozessen erfolgen. Die Analyse erfolgt nicht zuletzt vor dem Hintergrund strategischer Vorgaben. Dabei kann sich ergeben, daß diese Vorgaben zu modifizieren sind. In jedem Fall können die im VCD erfaßten Angaben an dieser Stelle unter Verweis auf die im OMD und im OPD verwalteten Konzepte weiter detailliert werden.

Die Eigenschaften von Objekten bzw. Klassen sind nach Maßgabe der im OMD angebotenen Kriterien weiter zu detaillieren. Dazu gehört die Beschreibung von Attributen, Diensten, Triggers und Guards sowie die nähere Spezifikation von Beziehungen. Dabei ist gegebenenfalls eine vorhandene Menge wiederverwendbarer Klassen zu berücksichtigen. Im Einzelfall ist dann zu prüfen, ob die Verwendung solcher Klassen möglich ist und welche Änderungen sie voraussetzt. Für die Objekte dazu geeigneter Klassen ist zudem eine Präsentation in Form eines Default-Views festzulegen.

Zusätzlich bei Berücksichtigung eines vorhandenen Informationssystems: Es ist festzulegen, welche Anwendungen eine objektorientierte Schnittstelle erhalten sollen und wie diese Schnittstellen zu realisieren sind. Dabei ist neben einer softwaretechnischen Betrachtung vor allem eine ökonomische Bewertung gefordert. Anwendungen, die als Quasi-Objekte verkapselt werden sollen, sind als Klassen in das gesamte Objektmodell einzufügen. Dabei werden sie durch ihre Dienste und die Beziehungen zu anderen Klassen bzw. Objekten gekennzeichnet. Daneben ist zu beschreiben, wie bestehende Anwendungen auf Objekte zugreifen können. Dabei ist einerseits an den Fall zu denken, daß Anwendungen auf ihre Daten über das Dateiverwaltungssystem des Betriebssystems zugreifen. In diesem Fall sind geeignete Repräsentationen von Objektzuständen in Dateien vorzusehen. Andererseits ist an den Fall zu denken, daß eine Anwendung ein vorhandenes Datenbankmanagement-System nutzt. In diesem Fall ist eine Reihe von Anpassungen denkbar, auf die hier nicht im Detail eingegangen werden soll. Um einige zu nennen: Für jede Klasse können Dienste vorgesehen werden, die den Zustand der Objekte dieser Klasse in eine Datenbank schreiben oder sie aus derselben lesen. In diesem Fall verwaltet ein Objekt seinen Zustand nur scheinbar selbst. Daneben kann die Ablage in einer konventionellen Datenbank zusätzlich zu der objektinternen Verwaltung der Objektzustände erfolgen. Besonders ambitioniert wäre die Abbildung des Schemas der Datenbank auf das Objektmodell eines Objektverwaltungssystems. Dabei ist auch daran zu denken, daß es zumeist nötig sein wird, die in den Anwendungen enthaltenen Aufrufe in der konventionellen Data Manipulation Language beizubehalten. Dabei sind nicht zuletzt bestehende und zukünftig zu erwartende Standards zu berücksichtigen.

Phase 4: Prototypische Implementierung und Test

Abgesehen von den wiederverwendeten Klassen sind alle Klassen soweit zu implementieren, daß ein Testbetrieb möglich ist. Beim Test ist auch auf die Evaluation und Modifikation der generierten Benutzerschnittstellen zu achten. Der Test dient darüber hinaus der sorgfältigen Überprüfung wichtiger im Objektmodell enthaltener Integritätsbedingungen.

Zusätzlich bei Berücksichtigung eines vorhandenen Informationssystems: Hier steht die Implementierung der Schnittstellen zwischen den neu geschaffenen Klassen und den bisherigen Anwendungen im Vordergrund. Da ein solcher Ansatz für ein komplexes System mit einem erheblichen Aufwand verbunden ist, ist es naheliegend, sich zunächst auf die Anwendungen zu konzentrieren, die sich nach Maßgabe der Analyseergebnisse dazu besonders anbieten.

Phase 5: Implementierung und Testbetrieb

Die in der vorherlaufenden Phase entdeckten Fehler sind zu beseitigen. Wenn nötig, sind Verbesserungen oder Optimierungen im Hinblick auf Ziele wie Performance, Speicherbedarf und dergleichen durchzuführen. In diesem Zusammenhang ist die Infrastruktur, wie Hardware, Netzwerke, Kommunikationsprotokolle, verbindlich festzulegen - soweit dies noch möglich bzw. nötig ist. Die den einzelnen Klassen und Teilvorgängen zugeordneten Benutzerschnittstellen sind soweit wie dies sinnvoll erscheint zu vereinheitlichen. Dabei sind auch die Benutzerschnittstellen bisheriger Anwendungen zu berücksichtigen.

Zusätzlich bei Berücksichtigung eines vorhandenen Informationssystems: Falls bei der prototypischen Implementierung verschiedene Alternativen für die Gestaltung einzelner Schnittstellen vorgesehen waren, ist nun jeweils eine bestimmte Alternative auszuwählen und eventuell weiter zu modifizieren.

Auch wenn durch die Verwendung der Entwicklungsumgebung die Gestaltung von Unternehmensmodellen erheblich erleichtert wird, kann doch nicht darüber hinweggesehen werden, daß der Aufwand für den umfassenden und detaillierten Neuentwurf eines Unternehmensmodells die Möglichkeiten eines Unternehmens übersteigen mag. Wenn allerdings erst ein Modell entwickelt (oder ein generisches Modell angepaßt) ist, sind die mit Änderungen der Strategie, Organisation oder des Informationssystems einhergehenden Modifikationen des Modells erheblich geringer.

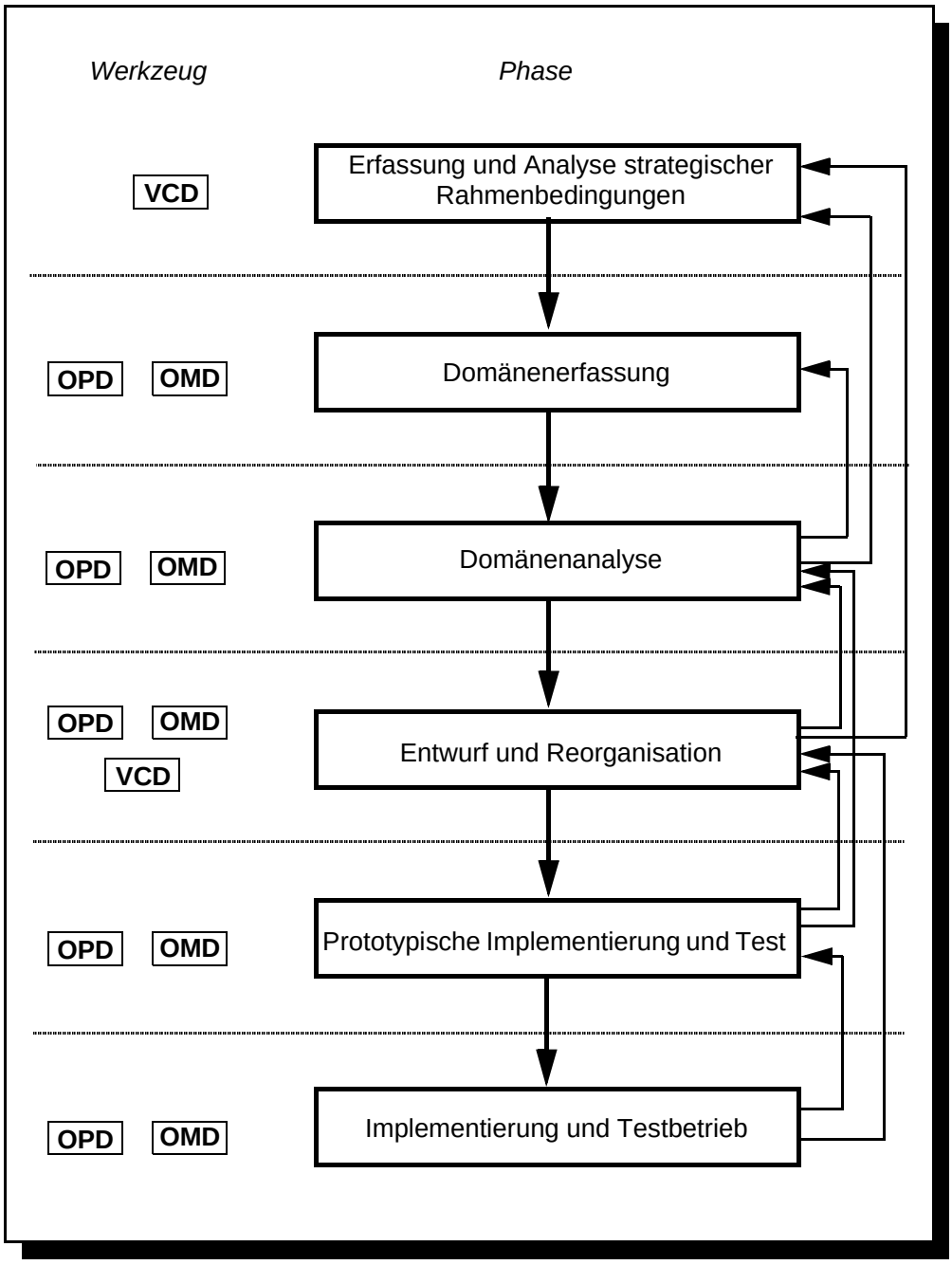


Abb. 80: Die Verwendung der Entwicklungsumgebung in den Phasen des Lebenszyklus und wichtige Beziehungen zwischen den Phasen

VI. Zukünftige Herausforderungen

"The transformation we are concerned with is not a technical one, but a continuing evolution of how we understand our surrounding and ourselves ... "

Terry Winograd und Fernando Flores

"... I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted."

Alan Turing

Schon in der Einleitung der vorliegenden Schrift wurde das eigentümliche Dilemma einer Untersuchung, die auf den Entwurf von Unternehmensmodellen gerichtet ist, deutlich: Einerseits verspricht eine ganzheitliche - multiperspektivische - Betrachtung von Unternehmen überzeugende Vorteile, andererseits ist der mit einer solchen Betrachtung verbundene Blickwinkel zwangsläufig so weit, daß die für wissenschaftliche Analysen in der Regel so wichtige Fokussierung auf einzelne Ausschnitte nicht eingehalten werden kann. Der vorgestellte Bezugsrahmen und die darauf basierende Entwicklungsumgebung ist denn auch vor allem als Vorschlag zur Betrachtung und Modellierung von Unternehmen im Hinblick auf den Einsatz von Informationssystemen gedacht. Auch wenn er durch die Vision generischer Referenzmodelle motiviert ist, kann (und will) er nicht mit einem - wie auch immer relativierten - Anspruch auf allgemeine Gültigkeit verbunden sein. Das gilt in mehrfacher Hinsicht: So ist der Bezugsrahmen nicht vorrangig darauf angelegt, unternehmensübergreifende Invarianzen zu beschreiben. Er reflektiert vielmehr eine *mögliche Betrachtungsweise*. Zudem ist er noch weiter zu detaillieren und vor allem mit domänenspezifischen Klassen zu füllen. Auch an den vorgestellten Werkzeugen kann noch eine Fülle von Feinarbeit vorgenommen werden (das gilt besonders für den Value Chain Designer). Vor allem aber wäre der Anspruch, ein Referenzmodell zu präsentieren, nicht vereinbar mit der Vorstellung eines diskursiven, evolutorischen Prozesses, an dem nicht zuletzt auch Betroffene beteiligt sind. Nur so ist darauf zu hoffen, daß die vielfältigen Bedürfnisse der Anwender in angemessener Weise berücksichtigt werden. Dabei ist auch daran zu denken, daß Referenzmodelle nicht verordnet werden können. Vielmehr setzen sie eine breite Akzeptanz voraus. Die Diskussion um Unternehmensmodelle hat gerade erst begonnnen - in der Forschung wie bei einigen Anwendern und Anbietern. Das hier entwickelte Gerüst einer Theorie multiperspektivischer Unternehmensmodellierung ist mit der Hoffnung verknüpft, diese Diskussion weiter anzuregen.

Im folgenden soll zunächst - gewissermaßen als selbstkritische Ergänzung des vorgestellten Modells - aufgezeigt werden, wo vorrangige Ansatzpunkte für weitere Forschungsarbeiten zu sehen sind. Anschließend soll die These verdeutlicht werden, daß die Verbesserung der Wirtschaftlichkeit betrieblicher Informationssysteme vor allem mit einer kulturellen Herausforderung verbunden ist. Den Abschluß bildet der Versuch, unter Rückgriff auf multiperspektivische Unternehmensmodelle eine Standortbestimmung der Wirtschaftsinformatik vorzunehmen.

1. Ansatzpunkte für die Weiterentwicklung des Bezugsrahmens und der Werkzeugumgebung

Das Metamodell zum Entwurf von Objektmodellen enthält eine Reihe von Kompromissen. So wurde Einfachvererbung der Mehrfachvererbung vorgezogen. Dafür gibt es gute Gründe (vgl. IV.2.1.1). Dennoch sind weitere Untersuchungen nötig, um eine differenziertere vergleichende Evaluation der beiden Alternativen zu ermöglichen. Dabei ist vor allem an den Entwurf und die Pflege komplexer Objektmodelle zu denken. Um beide alternativen Generalisierungsformen in anschaulicher Weise vergleichbar zu machen, wäre es wünschenswert, den OMD um die Möglichkeit der Mehrfachvererbung zu erweitern.

Die Einführung von Pre- und Postconditions sowie von Triggers und Guards trägt wesentlich zu einer semantischen Anreicherung von Objektmodellen bei. Gleichzeitig ist sie mit der Schwierigkeit verbunden, daß traditionelle Grenzen zwischen Modellierung und Programmierung aufgeweicht werden. Im OMD wurde ein Kompromiß gewählt, der die Spezifikation einfacher Bedingungen in einem Auswahldialog vorsieht. Er entlastet zwar von der Notwendigkeit, eine formale Sprache zu verwenden, erlaubt allerdings nicht, die Mächtigkeit der jeweiligen Konzepte voll auszuschöpfen. Es wäre deshalb sinnvoll, alternativ zur gegenwärtigen Lösung eine Spezifikationssprache zu entwerfen und dafür einen Parser zu implementieren. Eine solche Sprache könnte dann auch dazu verwendet werden, die Wertebereiche von Klassen formal zu definieren.

Im Metamodell ist vorgesehen, Beziehungen nicht als Objekte zu beschreiben. Diese Entscheidung kam durch die Erfahrungen mit dem zunächst favorisierten Ansatz, Beziehungen auch als Objekte zu modellieren[1], zustande. Dennoch sind weitere Untersuchungen (auch hier wieder durch den alternativen Entwurf großer Objektmodelle) wünschenswert, um zu gehaltvolleren Aussagen über die Vorteilhaftigkeit des einen oder des anderen Ansatzes zu gelangen. Im Hinblick auf die Vorbereitung der Implementierung ist daran zu denken, Verfahren zu entwicklen, die aus der Beschreibung von Triggers und Guards ein Gerüst für die Spezifikation von Diensten (in Form von Pre- und Postconditions) generieren. Dabei ist -

1. Vgl. dazu etwa Frank (1992 a) oder Frank/Klein (1992 b).

wie grundsätzlich beim Generieren - besonderes Augenmerk auf die Vermeidung von Irreversibilität zu richten.

Die Werkzeugumgebung bietet keine Unterstützung für die frühe Erhebungsphase. Im Phasenmodell in V.5 wird vielmehr auf eine Beschreibung mit Hilfe natürlichsprachlicher Texte verwiesen. Eine mögliche Werkzeugunterstützung in dieser Phase könnte in Form eines spezialisierten Hypertextsystems erfolgen: Auf diese Weise ließen sich zentrale Begriffe im Text (wie etwa Objektkandidaten) markieren. Sie könnten dann einerseits miteinander und später mit korrespondierenden Konstrukten auf anderen Abstraktionsebenen verknüpft werden. Aus markierten Begriffen könnten Glossare erstellt werden.[1]

Zur Unterstützung der Analyse von Objektmodellen kann eine Reihe möglicher Ansätze untersucht werden. Dabei ist vor allem an induktive Verfahren zu denken. So könnte beispielsweise ein Verfahren implementiert werden, das gegebene Klassen auf gemeinsame Muster untersucht und so Hinweise auf mögliche Generalisierungen liefern kann. In ähnlicher Weise könnten Dienste, die auf bestimmte Attribute zugreifen, auf gemeinsame Pre- oder Postconditions untersucht werden - ein Indiz für eine mögliche Generalisierung in Form von Triggers oder Guards. In diesem Zusammenhang wäre es wünschenswert, das Metamodell detaillierter zu formalisieren.[2]

Von wesentlicher Bedeutung für die Brauchbarkeit des Metamodells ist die Frage danach, ob es für die Anwendungspraxis nicht zu komplex ist. Zur Beantwortung dieser Frage sind empirische Untersuchungen mit repräsentativen Nutzern unumgänglich. Eine analytische Bewertung allein ist nicht hinreichend: So hat zwar die Regel "keep it simple" eine nicht zu unterschätzende Bedeutung für den Erfolg eines softwaretechnischen Ansatzes. Prominente Beispiele dafür sind das ERM und das relationale Datenmodell. Im Vergleich dazu ist das hier vorgestellte Metamodell von hoher Komplexität. Dennoch läßt sich aus diesem Vergleich kaum eine überzeugende Schlußfolgerung über die Angemessenheit dieser Komplexität ziehen: Einerseits ist der Entwurf eines Informationssystems mit einem ERM allein nicht getan. Die Integration mit einem funktionalen Modell ist tendenziell weniger übersichtlich als ein Objektmodell (auch wenn hier subjektive Prädispositionen natürlich streuen mögen). Außerdem ist das Metamodell seiner Komplexität zum Trotz durch das Bemühen um Vereinfachung gekennzeichnet. So wird von einzelnen Aspekten, die für die Software-Entwicklung im

1. In Jacobson (1992) ist ein Ansatz beschrieben, der eine systematische Erfassung von Begriffen aus einer Anwendungsdomäne ermöglicht. Dieser Ansatz bietet die Grundlage für ein spezielles - in Smalltalk implementiertes - Hypertextsystem.
2. In der Informatik gibt es bisher nur wenige Arbeiten, die auf eine formale Beschreibung von Klassen bzw. Objekten zielen. Ein gelungener Ansatz findet sich in Breu (1991).

engeren Sinn wichtig sind, abstrahiert, um die Betrachtung einer Domäne durch solche Spezifika nicht zu belasten. Dabei ist beispielsweise daran zu denken, Klassen als Objekte zu behandeln, also Metaklassen einzuführen.

Um einen möglichst friktionslosen Übergang zur Implementierung zu erreichen, ist eine weitgehend automatisierte Transformation eines Objektmodells in die Definitionssprache eines objektorientierten Datenbankmanagement-Systems (oder deutlicher: eines Objektverwaltungssystems) erstrebenswert. Eine entsprechende Transformation ist durch die partielle Überführung des Modells in SFK (vgl. V.2.3) bereits realisiert. SFK verfügt über sehr leistungsfähige Funktionen zur Objektverwaltung, ist aber mit zwei Einschränkungen verbunden: Es dient der Verwaltung residenter und nicht persistenter Objekte und es erlaubt keine Verteilung. Es wäre deshalb eine reizvolle Aufgabe, eine Anbindung an eines der am Markt angebotenen Objektverwaltungssysteme zu realisieren.[1]

Das an Petri-Netzen orientierte Konzept zur Modellierung von Handlungsabläufen in Unternehmen kann in mehrfacher Hinsicht weiterentwickelt werden. Dabei ist zunächst an Untersuchungen zu denken, wie weitere Integritätsbedingungen in das Modell eingeführt werden können. Zur Zeit ist die Nutzung des Modells im Rahmen des OPD vor allem auf die Analyse gerichtet. So dient es beispielsweise der Ermittlung des Informationsbedarfs in Teilvorgängen, der auch unter Verweis auf das Objektmodell konzeptuell exakt beschrieben werden kann. Wie in V.3.1 erwähnt sind gegenwärtig allerdings nur beschränkte Möglichkeiten vorgesehen, Beziehungen zwischen den konzeptuell beschriebenen Informationen auf Objektebene zu artikulieren. Hier sind Ansätze gefragt, die es erlauben, die Modelle in komfortabler und sicherer Weise mit Semantik anzureichern. Daneben wäre es reizvoll, zu untersuchen, inwieweit größere Bibliotheken implementierter Klassen (für Vorgangsdokumente und Teilvorgänge) ein schnelles Prototyping unterstützen. Das hieße, vor dem Hintergrund des hier vorgeschlagenen Bezugsrahmens die Machbarkeit der Vision des Implementierens von Vorgangssystemen mit Hilfe konfigurierbarer Komponenten[2] neu zu überprüfen.

In der gegenwärtigen Version unterstützt der OPD allein die Analyse einzelner Geschäftsprozesse. Vor allem in größeren Unternehmen kann es komplexe Verpflechtungen zwischen Geschäftsprozessen geben - in Form unmittelbarer Abhängigkeiten oder durch die Nutzung gemeinsamer Ressourcen. Eine entsprechende Weiterentwicklung des Werkzeugs verspräche Aufschlüsse über die Brauchbarkeit von Verfahren zur Unterstützung einer unternehmensweiten Optimierung der Ablauforganisation.

---

1. Dazu bietet sich vor allem ein System an, das über eine an Smalltalk angelehnte Object Definition Language verfügt - wie es beispielsweise in Butterworth et al. (1991) vorgestellt wird.
2. Vgl. dazu beispielhaft Nierstrasz et al. (1990).

Kooperatives Handeln in Unternehmen vollzieht sich nicht allein in Form reglementierter Vorgänge. Doch auch in solchen Fällen gelten gewisse - mehr oder weniger rigide - Randbedingungen, die mitunter von den Beteiligten selbst gesetzt und im Zeitverlauf modifiziert werden. Dabei ist beispielsweise an das Einrichten von Arbeitsgruppen zu denken, die mit variierender Autorisierung Informationen teilen und gemeinsam bearbeiten. Solche Arbeitskontexte werden häufig, wenn sie mit Computerunterstützung eingerichtet werden, mit dem Etikett "Computer Supported Cooperative Work" (CSCW) versehen. Ihre Modellierung weist gewisse Parallen zur Vorgangsmodellierung auf, gleichzeitig aber auch gewichtige Unterschiede. So ist der Informationsbedarf häufig nicht ex ante festzulegen. Zudem können neue Mitglieder aufgenommen werden, während der Arbeitskontext besteht. Die Bedingungen zur Regelung der gemeinsamen Arbeit sind gestreut in einem Kontinuum zwischen relativ streng und weitgehend offen.

Ein Werkzeug, das die Modellierung solcher CSCW-Kontexte unterstützt, muß eine besonders komfortable Benutzerschnittstelle bieten, da es vom Endanwender zu bedienen ist. Dazu gehört auch, den einzelnen Teilnehmern zu ermöglichen, individuelle aktive Informationsfilter zu definieren - ein Umstand, der seit einiger Zeit unter dem Stichwort "awareness"[1] diskutiert wird. Der OPD bietet günstige Voraussetzungen für eine entsprechende Erweiterung. So erlaubt der Grafik-Editor eine komfortable und anschauliche Modellierung zeitlicher Zusammenhänge (wenn eine solche gefordert ist). Vor allem aber verspricht der Zugriff auf das Objektmodell erhebliche Vorteile: Für gemeinsam genutzte Objekte können Zugriffsrechte definiert werden, es können Beziehungen zwischen Objekten etabliert werden (um referentielle Integrität zu unterstützen) und schließlich kann das Trigger-Konzept genutzt werden, um sich individuell über bestimmte Ereignisse informieren zu lassen.

Die strategische Perspektive ist von herausragender Komplexität. Es verwundert deshalb wenig, daß der Wertketten-Ansatz nur einen relativ offenen Bezugsrahmen darstellt. Es gibt deshalb einen Bedarf an Forschung zur weiteren Detaillierung des Bezugsrahmens und zur engeren Anbindung an die beiden anderen Komponenten. Solche Untersuchungen würden darin bestehen, die gegebenen Werkzeuge zur möglichst umfassenden Modellierung von Unternehmen zu nutzen, um auf diese Weise neue Integritätsbedingungen zu entdecken. Dabei ist allerdings einschränkend darauf hinzuweisen, daß die strategische Planung seit geraumer Zeit Gegenstand intensiver Forschung ist - ohne daß daraus eine gehaltvolle Theorie[2] entstanden ist. In diesem Zusammenhang ist auch der Wertketten-Ansatz selbst weiter kritisch zu prüfen.

1. Vgl. Prinz (1993).
2. Gehaltvoll impliziert im Hinblick auf die Implementierung nicht zuletzt: verläßliche und formalisierbare Aussagen über Invarianzen.

Der Entwurf von Unternehmensmodellen vollzieht sich in der Regel in einem evolutorischen Prozeß, in dessen Verlauf mehr oder weniger große Veränderungen des Modells zu erwarten sind. Außerdem ist davon auszugehen, daß meistens mehrere Personen beteiligt sind. Aus diesem Umstand erwachsen einige Anforderungen, die die Entwicklungsumgebung gegenwärtig nicht erfüllt. Die Implementierung einer einfachen Versionsverwaltung ist relativ schnell zu realisieren. Einfach soll dabei heißen: Die verschiedenen Versionen einzelner elementarer Teile des Modells werden in einer Liste abgelegt. Die jeweils jüngste Version ist dann für das gesamte Modell die gültige Version. Die gegenwärtig verwendeten Verzeichnisse zur Modellverwaltung sind dazu nur geringfügig zu ändern. Wenn mehrere Personen an einem Modell arbeiten, kann es wünschenswert sein, wenigstens temporär verschiedene Sichten auf das Modell zu erlauben. Wegen der vielfältigen Interdependenzen im Modell ist eine solche multipersonale Versionsverwaltung mit einem erheblichen Aufwand verbunden. Im einfachsten Fall würde man für jeden Benutzer eine Historie des Modells führen. Die Veränderungen anderer Benutzer blieben dann dem einzelnen verborgen. Solch ein Ansatz[1] wird aber nicht immer befriedigend sein. Um hier zu besseren Resultaten zu gelangen, ist es angeraten, die Interaktionen und Interdependenzen bei multipersonalem Modellentwurf zu untersuchen.

Wenn man an mehrere Benutzer denkt, liegt die Anforderung nahe, die Entwicklungsumgebung verteilt verfügbar zu machen. Die Werkzeuge sind grundsätzlich für eine Verteilung geeignet: Sie könnten als Server fungieren, die ihre Dienste auch nach außen (über die Grenzen der jeweiligen Maschine hinweg) anbieten. Dazu müßte lediglich ein Objekt implementiert werden, das die eingehenden Nachrichten synchronisiert und den Dispatch vornimmt. Der Wermutstropfen dabei: Um die Kommunikation zwischen Server und Clients auf verteilten Rechnern zu realisieren, sind entsprechende Protokolle zu nutzen. Solche Protokolle (wie etwa das auf TCP/IP aufsetzende RPC[2]) refererieren gegenwärtig auf einen kleinen Satz von Datentypen, in die die zu übertragenden Daten jeweils zu transformieren sind. Das bedeutet, daß die auf einem möglichst anwendungsnahen Niveau beschriebenen Klassen wieder auf hardwarenahe Datentypen heruntergebrochen und anschließend in die vom Protokol vorgesehene Repräsentation überführt werden müssen.

Dieser Umstand wird dadurch noch unangenehmer, daß entsprechende Transformationsdienste in Smalltalk zur Zeit nicht verfügbar sind.[3] Man muß also die Zustände der zu übertragenden Objekte in Datenstrukturen einer geeigneten Programmiersprache (wie etwa C) transformieren. Darüber hinaus wäre eine ver-

1. Entsprechende Implementierungen finden sich beispielsweise in einigen Smalltalk-Erweiterungen.
2. Vgl. dazu Santifaller (1990).

teilte Versionsverwaltung zu realisieren, da ja nur so gewährleistet werden kann, daß die in einem Objekt enthaltenen Referenzen in einheitlicher Weise befriedigt werden. Auch wenn diese Schwierigkeiten keine unüberwindbare Herausforderung darstellen, ist es gegenwärtig wenig reizvoll, eine entsprechende Entwicklung in Angriff zu nehmen. So sind die noch erforderlichen konzeptuellen Änderungen an den Werkzeugen leichter zu implementieren und zu testen, wenn man sich auf Smalltalk beschränken kann. Darüber hinaus sind in den nächsten Jahren Produkte zu erwarten, die die Verteilung unterstützen. Dabei ist einerseits an objektorientierte Kommunikationsprotokolle (wie beispielsweise von der OMG mit dem "Common Object Request Broker"[1] vorgesehen) zu denken, andererseits an Objektverwaltungssysteme.

2. Der Fortschritt betrieblicher Informationssysteme als kulturelle Herausforderung

Wenn man der These zustimmt, daß unternehmensübergreifende Integration und Wiederverwendbarkeit für die Leistungsfähigkeit und Wirtschaftlichkeit betrieblicher Informationssysteme von besonderer Bedeutung sind, sollten sie für die weitere Entwicklung von Informationssystemen eine zentrale Orientierung darstellen. Wenn man sich darüber hinaus der hier entwickelten Argumentation anschließt, wonach multiperspektivische Unternehmensmodelle ein probates Mittel zur Förderung von Integration und Wiederverwendbarkeit darstellen, bleibt die Frage nach konkreten Strategien oder abstrakter: nach den besonderen Herausforderungen, die mit einer solchen Orientierung verbunden sind. Das gilt sowohl für die dedizierte Forschung als auch für die Praxis.

Wenn die an der Erforschung betrieblicher Informationssysteme beteiligten Disziplinen die Forderung nach Integration und Wiederverwendbarkeit überzeugend vertreten wollen, ist es angeraten, sie auch auf die eigene Arbeit anzuwenden: In der einschlägigen Forschung gibt es eine immer noch wachsende Flut von Publikationen, die mit exzessiven Überschneidungen verbunden ist - wenngleich diese auch nicht immer auf den ersten Blick als solche zu erkennen sind. Ähnliches gilt für die häufig mit viel Kompetenz und großer Sorgfalt entwickelten Prototypen. Die mit größtenteils isoliert betriebenen Einzelprojekten verbundene Vernachlässigung von Synergiepotentialen ist umso bedenklicher, als die Entwicklung auch prototypischer Informationssysteme mit einem stetig wachsenden Aufwand verbunden ist - trotz produktivitätsfördernder Entwicklungsumgebungen. Das Feh-

3. Es gibt allerdings Ankündigungen entsprechender Implementierungen, die eine Transformation in standardisierte Sprachen für die Beschreibung von Objektschnittstellen erlauben.

1. Vgl. Soley (1992) und Atwood (1991).

len gemeinsamer Referenzsysteme ist auch auf Fachkonferenzen in bedauerlicher Weise spürbar: Die kurze Zeit für Diskussionen reicht häufig nicht einmal aus, um zu einem gemeinsamen Verständnis zentraler Begriffe zu gelangen. Darüber hinaus müssen sich die Disziplinen, die an der Erforschung betrieblicher Informationssysteme beteiligt sind, eine Kritik gefallen lassen, die häufig als Diagnose auf die Misere der Anwendungspraxis appliziert wird: Sprachbarrieren gibt es nicht nur zwischen den Vertretern verschiedener Disziplinen[1], sondern auch innerhalb einer Disziplin.

Nun sind die skizzierten Anforderungen nicht nur - mit der für sich allein schon immensen - Herausforderung verbunden, den Prozeß wissenschaftlicher Forschung in anderer Weise zu organisieren. Es geht vielmehr auch darum, tradierte Wissenschaftskultur zu überdenken. Das gilt allgemein für gewisse Grundwerte wissenschaftlichen Arbeitens. So scheint das Bemühen um die gegenseitige Anpassung an gemeinsame Referenzmodelle im Konflikt mit wichtigen Voraussetzungen erfolgreicher Forschung - wie Kreativität, Individualität und Ideenwettbewerb - zu stehen. Um es überzeichnend zu formulieren: Droht hier nicht die Gefährdung freier Wissenschaft durch eine Standardisierungsbürokratie? Für Disziplinen, die sich dem Entwurf (software-) technischer Systeme verpflichtet fühlen (hier ist vor allem an die Informatik und - mit Abstrichen - an die Wirtschaftsinformatik zu denken), ist ein weiterer, subtiler Konflikt zu berücksichtigen: Der Erfolg solcher Forschung wird gern mit Vokabeln wie Entdeckung und Erfindung assoziiert. Wenn der Fokus der Forschung aber vor allem auf die Schaffung generischer Referenzsysteme - auch wenn dazu Entdecken und Erfinden sehr hilfreich sein mögen - gerichtet ist, wird die Sinn-stiftende Vermittlung der Ziele wissenschaftlicher Arbeit zu einem schwierigen Unterfangen. Es ist allemal leichter, Legitimation und damit: Finanzierung zu sichern, wenn die traditionellen, naturwissenschaftlich-technizistisch geprägten Forschungsziele betont werden. Die beachtlichen Erfolge der KI-Forschung beim Akquirieren von Forschungsgeldern liefern dafür ein beeindruckendes Beispiel[2] - obwohl schon Turing auf einen Umstand hingewiesen hat, der durch die Geschichte der Datenverarbeitung nachdrücklich bestätigt wird: Wenn man die Problemlösungskapazität von Digitalrechnern erhöhen will, ist es erfolgversprechender, die zu verarbeitenden Informationen so aufzubereiten, daß darauf zuverlässige formale Interpretationsverfahren angewandt werden können, als sich um die formale Rekonstruktion menschlicher Intelligenz zu bemühen.[3]

---

1. Ich möchte an dieser Stelle allerdings nicht verschweigen, daß es (natürlich) auch sehr angenehme Ausnahmen gibt, wie ich in den zurückliegenden Jahren vor allem in der GMD von Zeit zu Zeit feststellen durfte.

2. Gleichzeitig weckt dieses Beispiel Zweifel an der Angemessenheit einer solchen Orientierung: Wenn Versprechen nicht eingelöst werden, ist langfristig eine Gefährdung der Reputation zu erwarten.

Die dargestellten Einwände machen deutlich, daß eine stärkere Hinwendung zur Entwicklung von Referenzmodellen mit erheblichen Auswirkungen auf die Praxis gegenwärtiger Forschung verbunden ist. Bei näherer Betrachtung handelt es sich dabei aber nicht nur um eine Neuorientierung, sondern auch um eine Neubesinnung: Schließlich ist die Förderung und Pflege eines freien, kritischen und fruchtbaren Diskurses eine der vornehmsten Aufgaben von Wissenschaft. Die Arbeit an gemeinsamen Referenzsystemen schafft dafür günstige Voraussetzungen. Genau hier liegt m.E. die Herausforderung: Der Umgang mit solchen Modellen darf nicht verordnet werden, es müssen vielmehr auch für den einzelnen Forscher motivierende Anreize dafür geschaffen werden, sich einzubringen.[1] Es geht also nicht darum, bestimmte Sichtweisen vorzugeben, sondern eher darum, ein gemeinsames Forum zu entwickeln, das dem einzelnen Wissenschaftler motivationsfördernde Resonanz verspricht. In ähnlicher Absicht - wenn auch mit engerem Fokus - sprechen Tsichritzis und Gibbs (1990, S. 1) von "software communities":

> "... we define a *software community* as a group of persons with similar software culture. By "similar software culture" we mean a common set of terms for communication, a common understanding of the issues of software development and a common approach for discussing and resolving these issues."

In der Praxis - sowohl bei Anwendern als auch bei Anbietern - ist man auf den ersten Blick flexibler. Vor dem Hintergrund einfacher Wirtschaftlichkeitsüberlegungen breitet sich allmählich die Erkenntnis aus, daß verbindliche Konventionen, also Standards, auch in Anwendungsnähe erhebliche Vorteile versprechen. Indizien dafür sind unter anderem Vereinbarungen für den elektronischen Austausch von Geschäftsdokumenten und auch erste Ansätze zum Entwurf branchenspezifischer Unternehmensmodelle (IBM 1990). Die nähere Betrachtung dieser Beispiele ist allerdings ernüchternd: Während einfache, softwaretechnisch fragwürdige Vereinbarungen wie EDIFACT[2] längst als Standard etabliert sind, sind die wenigen Ansätze zur Schaffung generischer Unternehmensmodelle und zur Implementierung zugehöriger Klassenbibliotheken von einem solchen Status noch weit entfernt. Auch wenn es sich dabei um ein ungleich komplexeres Unterfangen handelt - schließlich geht es nicht nur um den Entwurf unternehmensweiter Informationssysteme, sondern auch um die Erarbeitung überzeugender Migrationsstrategien, ist zu vermuten, daß auch Hindernisse auf einer politisch/ökonomischen Ebene eine Rolle spielen. Eine solche gemeinsame Anstrengung ist eben mit schwer kalkulierbaren Risiken verbunden: Der Einigungsprozeß kann scheitern, das gesamte Vorhaben durch die Marktdominanz alternativer

3. Eine ausführliche Darstellung dieser Position findet sich in Frank (1988)

1. Das läßt bei einigem Optimismus auf einen sehr viel wirksameren Wettbewerb hoffen - und für diejenigen, die ihm gewachsen sind, auf konstruktive Rückkopplungen.

2. Zur Kritik vgl. Frank (1991).

Ansätze gefährdet werden.

Jenseits solcher konkreten Schwierigkeiten ist an ein grundsätzliches Problem zu denken, das die Eckpfeiler der Marktwirtschaft betrifft. So droht durch eine Standardisierung auch von Anwendungssystemen eine Gefährdung des Wettbewerbs. Das gilt vor allem für die Anbieter, aber auch für die Nutzer, sofern diese sich durch den Einsatz von Informationssystemen einen Vorteil gegenüber ihren Konkurrenten erhoffen. Nun kann man sich vorstellen, daß Unternehmensmodelle einen gemeinsamen Rahmen darstellen, innerhalb dessen sich der Wettbewerb in einzelnen Bereichen vollzieht. Für Software-Anbieter hieße dies, daß sie sich auf eine für bestimmte Marktsegmente besonders attraktive Implementierung von Klassenbibliotheken spezialisieren könnten. Wenn ein Anbieter die Nutzung bestimmter Klassen möglich macht, können diese von anderen Anbietern verwendet werden. Auch wenn dies einerseits ein erwünschter Effekt ist, ergibt sich daraus allerdings die Frage danach, wie die Offenheit solcher Systeme mit dem notwendigen Urheberschutz der Anbieter vereinbar ist.

Die skizzierten Schwierigkeiten sind ernstzunehmen, sie sind allerdings nicht geeignet, generische Unternehmensmodelle als eine sinnvolle Orientierung zu diskreditieren. Dafür gibt es eine Reihe von Indizien. So sind bereits heute verschiedene Bestrebungen zu verzeichnen, die die gegenseitige Nutzung einzelner Dienste von Anwendungssystemen zum Ziel haben.[1] Damit deutet sich zunächst eine zunehmende Verflechtung von Anwendungen und langfristig die Überwindung des gegenwärtigen Anwendungsbegriffs an (wie sie im Szenario in IV.2.2.1 skizziert ist). Vor allem aber gibt es überzeugende Belege dafür, daß Unternehmen durchaus bereit sind, tradiertes Konkurrenzbewußtsein zu überdenken. Hier ist beispielsweise an strategische Allianzen selbst in solchen Bereichen zu denken, die traditionell durch einen starken Wettbewerb gekennzeichnet sind - Differenzierung wird dann eben in anderen Bereichen angestrebt.

Die dargestellten Perspektiven markieren nicht nur Chancen und Herausforderungen für die betroffenen Unternehmen, sie führen auch zu einer Reihe neuartiger Forschungsfragestellungen. Auf einer technischen Ebene ist hier an den Entwurf angemessener Abrechnungssysteme für die Software-Nutzung zu denken. Dazu gibt es bereits erste Ansätze, wie etwa "superdistribution" (Mori 1990, vgl. auch II.2.1). Die Untersuchung solcher neuen Allokations- und Distributionsformen stellt nicht zuletzt für die Wirtschaftswissenschaften ein reizvolles Forschungsfeld dar.

---

1. Beispielsweise sehen die von der OMG vorgeschlagenen Architekturen vor, das Anwendungen eine Menge von Diensten verfügbar machen. So kann beispielsweise ein Texteditor von einer Tabellenkalkulation den Inhalt einer bestimmten Zelle anfordern. Erste Implementierungen der dazu nötigen Kommunikationsdienste sind in Kürze zu erwarten - wie etwa das "Distributed Objects Everywhere" von Sun.

3. Zur Rolle der Wirtschaftsinformatik

Es ist unstrittig, daß der Entwurf generischer Unternehmensmodelle nur als evolutorischer Prozeß, der von verschiedenen wissenschaftlichen Disziplinen begleitet werden sollte, gelingen kann. Hier ist an die Informatik, die Betriebswirtschaftslehre und vor allem an die Wirtschaftsinformatik zu denken. Erkenntnisse aus diesen Disziplinen sind in den gemeinsamen Bezugsrahmen einzuordnen. Gleichzeitig ist zu erwarten, daß der interdisziplinäre Austausch neue Untersuchungen motiviert - Unternehmensmodelle als Katalysator für neue Forschungsarbeiten. Darüber hinaus ist es nötig, eine möglichst große Zahl von Unternehmen in die Betrachtung einzubeziehen, um durch vergleichende Analysen und wechselseitige Anpassungen zu konfigurierbaren generischen Modellen zu gelangen, die für viele Unternehmen eines bestimmten Typs verwendbar sind. Abbildung 81 zeigt den vereinfachten Ablauf eines solchen Ansatzes.

Die skizzierte Evolution von Unternehmensmodellen weist offensichtliche Ähnlichkeiten zu gängigen Erkenntnismodellen in den Wissenschaften - vor allem in empirischen, anwendungsorientierten Sozialwissenschaften - auf. Gleichzeitig gibt es deutliche Unterschiede. Dabei ist zunächst an die bereits erwähnte Notwendigkeit einer interdisziplinären Kooperation zu denken. Weder überzeugende Begründungen dieser Notwendigkeit noch pathetisch vorgetragene Appelle reichen hin, eine wirkliche Zusammenarbeit anzuregen. Vielmehr muß den verschiedenen Fachwissenschaftlern deutlich werden, daß sie in die Kooperation nicht nur ihre eigenen Interessen einbringen können, sondern darüber hinaus eine gute Chance besteht, ihre Forschungsresultate durch die synergetische Verknüpfung mit anderen Ergebnissen anzureichern. Um diese Anreize aufzuzeigen, ist es nach meiner Erfahrung hilfreich, prototypische Modelle und Werkzeuge zu präsentieren, weil deren kritische Diskussion dazu beiträgt, konkrete Ansatzpunkte für eine Zusammenarbeit zu erkennen. Darüber hinaus ist es notwendig, den Prozeß interdisziplinärer Forschung zu moderieren und zu motivieren. Es wäre naiv, die damit verbundenen Schwierigkeiten zu unterschätzen. Gleichzeitig wäre es falsch, deswegen die Möglichkeit einer solchen Kooperation zu verwerfen.

Zukünftige Informationssysteme erfordern Modelle von Anwendungsbereichen, die einerseits hohe softwaretechnische Anforderungen erfüllen, andererseits eine angemessene Rekonstruktion fachlicher Konzepte darstellen. Beide Aspekte sind interdependent: So steht einerseits die strategische Planung und die organisatorische Gestaltung eines Unternehmens in Wechselwirkung mit den Optionen der Informationstechnologie, andererseits sind bei der Beschreibung fachlicher Konzepte die besonderen Restriktionen softwaretechnischer Modelle zu berücksichtigen.

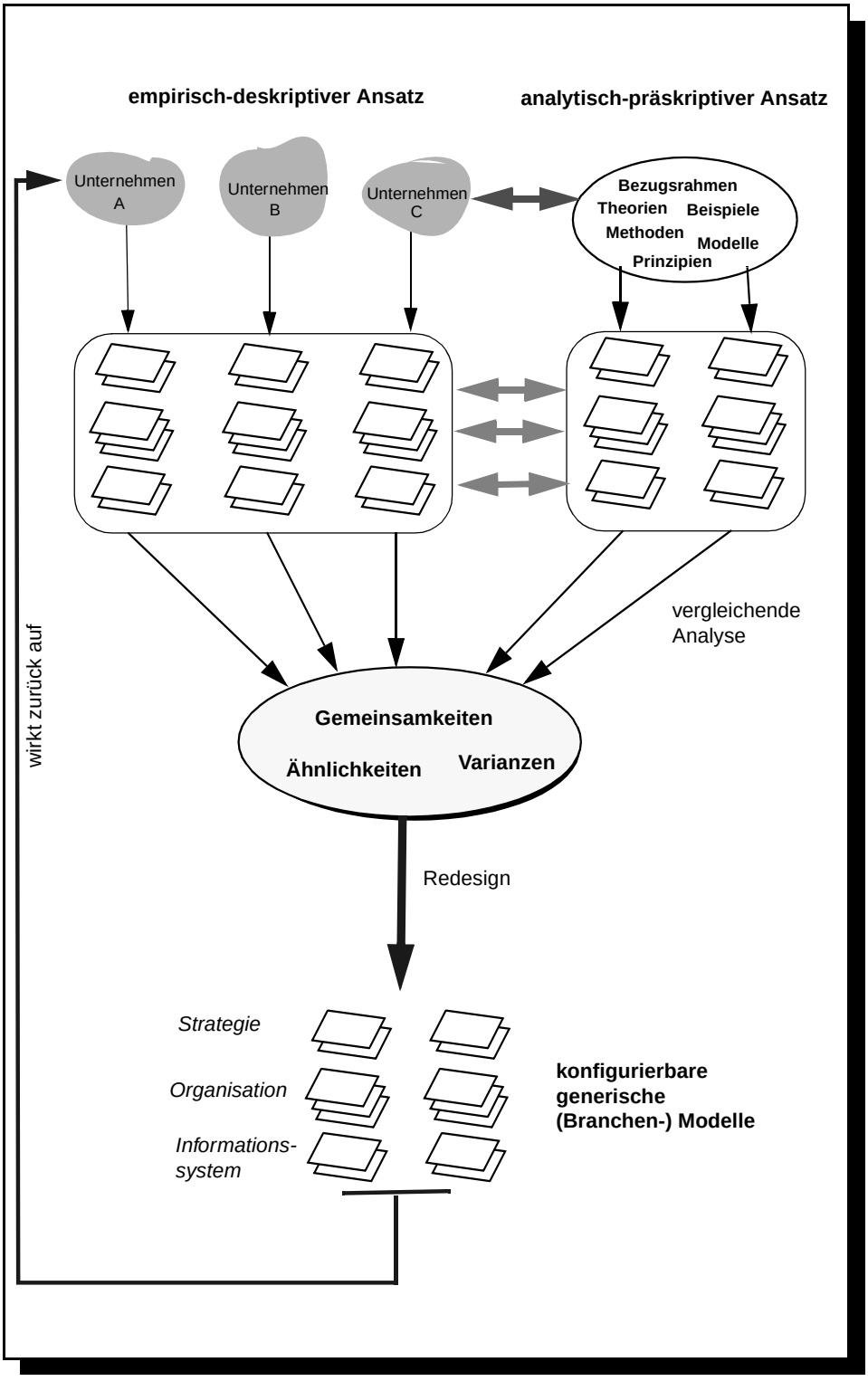


Abb. 81: Idealtypische Evolution konfigurierbarer generischer Unternehmensmodelle.

Ortner (1983, S. 7) spricht in diesem Zusammenhang treffend von einer "Rekonstruktion oder gar eine(m) Neuaufbau von Sprache", für die "Sprachgemeinschaft" der am Entwurf und der Nutzung von Informationssystemen beteiligten Personen. Isoliert betriebene Forschungen der Betriebswirtschaftslehre und der Informatik können kaum dazu beitragen, eine solche gemeinsame sprachliche Basis zu schaffen. Die dazu notwendige interdisziplinäre Zusammenarbeit ist für die Informatik wie auch für die Betriebswirtschaftslehre mit gravierenden Umstellungen verbunden. Dennoch - oder besser: gerade deshalb - ist hier vor allem die Wirtschaftsinformatik gefordert. So ist sie jenseits ihres wirtschaftswissenschaftlichen Fundaments ohnehin mehr oder weniger interdisziplinär orientiert. Daneben bietet sich ihr durch eine *aktive Moderation* die Chance, ihre Bedeutung für die Nachbardisziplinen zu betonen und damit dem Dilemma entgegenzuwirken, das sie zeitweilig plagt: Von der Betriebswirtschaftslehre nicht als vollwertige Teilsdisziplin anerkannt zu sein und von der Informatik als "Bindestrich-Informatik" abgetan zu werden.

Mit der Entwicklung von Unternehmensmodellen ist nicht allein das Ziel verbunden, gegebene Unternehmensrealität abzubilden. Darüber hinaus kommt der Frage, wie Realität in Wechselwirkung mit einer gegebenfalls noch zu entwikkelnden Informationstechnologie *gestaltet* werden kann, eine besondere Bedeutung zu. Der Erfolg von Unternehmensmodellen mißt sich also auch an der Akzeptanz in der Praxis - gleichwohl man ihn nicht darauf reduzieren sollte. Akzeptanz setzt Relevanz für und Transfer in die Unternehmen voraus. Damit ist ein Problem angesprochen, dessen Diskussion in der Betriebswirschaftslehre eine lange Tradition hat und das in neuerer Zeit auch manchen Informatiker beschäftigt.

Missionarischer Eifer hilft hier wenig. Unternehmensmodelle bieten allerdings in zweifacher Hinsicht gute Aussichten dafür, den Austausch zwischen der Praxis und den beteiligten Forschungsdisziplinen zu fördern. So bietet eine werkzeuggestützte Präsentation von Modellen, die zudem durch multimediale Annotationen ergänzt ist, ein anschauliches Kompendium. Der besondere Wert eines solchen Kompendiums dürfte weniger darin liegen, daß es einen unmittelbaren Beitrag zur Lösung aktueller Probleme bietet. Es könnte vielmehr gehaltvolle Orientierungen für geplanten Wandel und für die Bewertung der vielfältigen Angebote im Kontext betrieblicher Informationssysteme liefern. Ein solcher Ansatz ist für die Profilierung der Wirtschaftsinformatik gegenüber den einschlägigen Beratungsunternehmen Chance und Herausforderung zugleich. Im Unterschied zu den Beratern kann die Wirtschaftsinformatik in ihren Entwürfen von störenden und in der Praxis in vielfältiger Weise streuenden Restriktionen abstrahieren - also beispielsweise vom "Devil of Eternal Compatibility with the Horrors of the Past" (Meyer 1990, S. 83).[1] Eine derartige Ausrichtung, die letztlich den traditio-

nellen Theoriebegriff ("Ausschau") reflektiert, sollte allerdings nicht dazu führen, daß die Wirtschaftsinformatik den Kontakt zu ihrer Praxis verliert - Abstraktion also mit Vernachlässigung verwechselt wird.

Von besonderer Bedeutung für den Transfer wissenschaftlicher Erkenntnisse ist die Lehre. Werkzeuggestützt präsentierte Unternehmensmodelle bieten hier die Chance, den Studierenden das Lehrgebäude der Wirschaftsinformatik im interaktiven Umgang zu vermitteln. Im Hinblick auf die beruflichen Anforderungen des Wirtschaftsinformatikers ist dabei auch an die Berücksichtigung solcher Teile eines Unternehmensmodells zu denken, deren Konkretisierung von anderern Disziplinen stammt. Daneben schaffen Unternehmensmodelle eine günstige Voraussetzung dafür, die Zielsetzung von Diplomarbeiten und Dissertationen zu verdeutlichen und ihre Ergebnisse gegebenenfalls in einen Gesamtzusammenhang einzuordnen. Darüber hinaus bieten die so erfaßten Konzeptualisierungen wie auch deren Instanzierungen vielfältige Möglichkeiten für die (kritische) Anwendung und Erforschung automatisierter Analyseverfahren (vgl. VI.1) - das Modell als Labor. Damit ergibt sich die Chance, die Forderung nach Einheit von Forschung und Lehre in eindrucksvoller Weise einzulösen. Multiperspektivischen Unternehmensmodellen könnte dabei eine tragende Rolle zukommen - als Objekt und Objektivierung der Wirtschaftsinformatik.

1. Was gewiß nicht ausschließen sollte, daß solche Restriktionen in dedizierten Forschungsarbeiten eingehend untersucht werden.

Verzeichnis der Abbildungen

Literaturverzeichnis

Abkürzungen in Bezeichnungen von Zeitschriften und Sammelbänden

ACM	Association of Computing Machinery
AI	Artificial Intelligence
COMPCON	Computer Society International Conference
COMPSAC	Computer Software and Application Conference
HBR	Harvard Business Review
HICSS	Hawaii International Conference on System Sciences
HMD	Handbuch der modernen Datenverarbeitung
IEEE	Institute of Electrical and Electronical Engineers
IFIP	International Federation for Information Processing
IT	Information Technology
JOOP	Journal of Object-Oriented Programming
MIS	Management Information Systems
OOPSLA	Object Oriented Programming. Systems, Languages and Applications
PIK	Praxis der Informationsverarbeitung und Kommunikation
SIGOA	Special Interest Group on Office Automation
SIGPLAN	Special Interest Group on Programming Languages
TODS	Transactions on Database Systems
ZfB	Zeitschrift für Betriebswirtschaft
ZfbF	Zeitschrift für betriebswirtschaftliche Forschung
ZfhF	Zeitschrift für handelswissenschaftliche Forschung

Ackroyd, M.; Daum, D. (1991): Graphical notation for object-oriented design and programming. In: JOOP, Vol. 3, No. 5, S. 18-28

Ader, M.; Nierstrasz, O.; McMahon, S.; Müller, G.; Pröfrock, A.C. (1990): Gesamtheitliche objektorientierte Entwicklungsumgebungen. ITHACA-Forschungsbericht, ohne Ort

Ahmed, S.; Wong, A.; Sriram, D.; Logcher, R. (1992): Object-oriented database management systems for engineering: A comparison. In: JOOP, Vol. 5, No. 3, S. 27-43

Alabisco, B. (1988): Transformation of data flow analysis models to object-oriented design. In: Meyrowitz, N. (Hg.): OOPSLA ’88 - conference proceedings. New York, S. 335-353

Alasuvanto, J.H. (1990): Multi-Agent Hypermedia Systems in CIM. In: Proceedings of the Twenty-Third HICSS, Los Alamitos/Ca., S. 436-444

Albert, H. (1960): Wissenschaft und Politik. Zum Problem der Anwendbarkeit einer wertfreien Sozialwissenschaft. In: Topitsch, E. (Hg.): Probleme der Wissenschaftstheorie. Wien, S. 201-232

Arango, G. (1989): Domain Analysis - From Art Form to Engineering Discipline. In: Kent, W. (Hg.): Fifth International Workshop on Software Specification and Design - May 1989 Pittsburgh. Washington/D.C., S. 152-159

Atwood, T. (1991): Why the OMG Object Request Broker should mean good news for object databases. In: JOOP, Vol. 4, No. 4, S. 8-11

Badaracco, J. (1991): The Knowledge Link: How Firms Compete Through Strategic Alliances. Cambridge/Mass.

Baethge, M.; Overbeck, H. (1986): Zukunft der Angestellten. Neue Technologien und berufliche Perspektiven in Büro und Verwaltung. Frankfurt/New York

Bailin, S.C. (1989): An object-oriented requirements specification method. In: Communications of the ACM, Vol. 32, No. 5, S. 608-623

Balzer, R. (1984): Evolution as a new Basis for Reusability. In: COMPSAC 84, The IEEE Computer Society´s Eigth International Computer Software & Applications Conference. Nov. 7-9, Chicago/Ill., Silver Spring, S. 471-473

Balzert, H. (1990): CASE: Systeme und Werkzeuge. Mannheim

Barbier, F. (1992): Object-oriented analysis of systems through their dynamic aspects. In: JOOP, Vol. 5, No. 2, S. 45-51

Barnard, C.H. (1938): The Functions of the Executive. Cambridge

Barthelmess, W. (1991): IBM-FAS 90 - Financial Application Solution for the 90's. Raleigh

Batini, C.; Ceri, S.; Navathe, S.B. (1992): Conceptual database design- an entity-relationship approach. Redwood City/Ca.

Becker, L.; Guay, T. (1992): Object-oriented diagnosis. In: JOOP, Vol. 5, No. 6, S. 43-52

Behrends, H.; Jasper, H. (1993): Spezifikation und Verarbeitung von Ereignissen in aktiven Datenbanken. In: Reichel, H. (Hg.): Informatik, Wirtschaft, Gesellschaft. 23. GI-Jahrestagung. Berlin, Heidelberg u.a., S. 470-475

Benjamin, R.I. (1990): Electronic Data Interchange: How Much Competitive Advantage? In: Long Range Planning, Vol. 23, No. 1, S. 29-40

Benson, R.J.; Parker, M.M. (1985): Enterprise-wide Information Management: An Introduction to the concepts. IBM Los Angeles Scientific Center Working Paper GC320-2768, May. Los Angeles

Berard, E. (1986): An Object-Oriented Design Handbook. Rockville/M.D.

Berger, P.L.; Luckmann, T. (1980): Die gesellschaftliche Konstruktion der Wirklichkeit. Eine Theorie der Wissenssoziologie. Frankfurt/M.

Bergmann, M.; Kohlert, R.; Mildt, H. (1987): MOSAIK - Modulares und rechnergestütztes Methodenpaket zur effizienten Gestaltung der Büroarbeit. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 183 - 206

Biggerstaff, T. (1984): A Radical Hypothesis: Reusability is the Essence of Design. In: COMPSAC 84, The IEEE Computer Society´s Eigth International Computer Software & Applications Conference. Nov. 7-9, Chicago/ Ill. Silver Spring

Biggerstaff, T.; Richter, C. (1987): Reusability Framework, Assessment and Direction. In: IEEE Software, Vol. 4, No. 2, S. 50-64

Biggerstaff, T.J.; Perlis, A.J. (1989): Software reusability. Vol. 1: Concepts and models. Reading/Mass. u.a.

Biggerstaff, T.J.; Perlis, A.J. (1989): Software reusability. Vol. 2: Applications and experience. Reading/Mass. u.a.

Blumenthal, S.C. (1969): Management Information Systems. A Framework for Planning and Development. Englewood Cliffs/N.J.

Bode, T.; Cremers, A.B.; Freitag, J. (1992): OMS - Ein erweiterbares Objektmanagementsystem. In: Bayer, R.; Härder, T.; Lockemann, P. (Hg.): Objektbanken für Experten. Berlin, Heidelberg, S. 29-54

Boehm, B. (1981): An experiment in small-scale software engineering. In: IEEE Transactions on Software Engineering SE-7(5), S. 482 ff.

Booch, G. (1982): Object-Oriented Design. In: Ada-Letters, Vol. 1, No. 3, S. 64-76

Booch, G. (1986 a): Object-Oriented Development. In: IEEE-Transactions on Software-Engineering. Vol. 12, No. 2, S. 211-221

Booch, G. (1986 b): Software Engineering with Ada. Menlo Park/Ca.

Booch, G. (1987): Object-Oriented Development. In: Peterson, G.E. (Hg.): Object-Oriented Computing - Tutorial. Vol. 2: Implementations. Washington/DC., S. 5-15

Booch, G. (1990): Object-oriented design with applications. Redwood City

Bourbaki, N. (1991): Toward a definition of object-oriented languages, part 1. In: JOOP, S. 62-65

Boynton, A.C.; Zmud, R.W. (1987): Information Technology Planning in the 1990's: Directions for Practice and Research. In: MIS Quarterly, No. 1, S. 59-71

Bracchi, G.; Pernici, G. (1984): The Design Requirements of Office Systems. In: ACM Transactions on Office Information Systems, Vol. 2, No. 2, S. 151-170

Braun, U.; Schmid, H.U. (1988): Wiederverwendbare abstrakte Datentypen und deren Auswahl durch ein Expertensystem. In: Informatik - Forschung und Entwicklung, Nr. 3, S. 164-181

Bretzke, W.R. (1980): Der Problembezug von Entscheidungsmodellen. Tübingen

Breu, R. (1991): Algebraic specification techniques in object oriented programming environments. Berlin. u.a.

Brodie, M.L. (1984): On the Development of Data Models. In: Borgida, A.; Mylopoulos, J.; Schmidt, J. (Hg.): On Conceptual Modelling. Perspectives from Artificial Intelligence, Databses and Programming. Berlin, Heidelberg u.a., S. 19-47

Brodie,M.L.; Mylopoulos,J.; Schmidt,J. (1984): On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming. Berlin, Heidelberg u.a.

Bruni, G.; Cardigo, C.; Damiani, M.; Seminati, G. (1990): Final Report on Insurance Domain Requirement Analysis. ITHACA.Datamont.89.D.7.3., ohne Ort

Brzoska, C. (1990): Temporal logic programming and its relation to constraint logic programming. Karlsruhe

Budde, R.; Schnupp, P.; Schwald, A. (1980): Untersuchungen über Maßnahmen zur Verbesserung der Software-Produktion. Teil 1: Theoretische Ansätze auf dem Gebiet der Software-Technologie. GMD-Bericht 130. München

Budde, R.; Züllighoven, H. (1990): Software-Werkzeuge in einer Programmierwerkstatt - Ansätze eines hermeneutisch fundierten Werkzeug- und Ma-

schinenbegriffs. München u.a.

Buhr, R. (1984): System Design with Ada. Englewood Cliffs/N.J.

Bullinger, H.J.; Niemeier, J. (1989): Integrationsmanagement auf dem Weg zu CIB. In: Office Management Nr. 10, S. 6-21

Burton, B.A.; Aragon, R.W.; Bailey, S.A.; Koehler, K.D.; Mayes, L.A. (1987): The Reusable Software Library. In: IEEE Software, July, S. 25-33

Butterworth, P. (1991): ODBMSs as database managers. In: JOOP, Vol. 4, No. 2, S. 47-50

Butterworth, P.; Otis, A.; Stein, J. (1991): The Gemstone Object Database Management System. In: Communications of the ACM, Vol. 34, No. 10, S. 64-77

Carley, K. (1992): Organizational Learning and Personnel Turnover. In: Organization Science. Vol. 3, S. 20-47

Carlyle, R. (1990): The Tomorrow Organization. In: Datamation, Feb. 1, S. 22-29

Checkland, P.B. (1984): Systems Thinking in Management: The Development of Soft Systems Methodology and Its Implications for Social Science. In: Ulrich, H.; Probst, G.J.B. (Hg.): Self-Organization and Management of Social Systems. Berlin, Heidelberg u.a., S. 94-104

Checkland, P.B. (1987): Weiches Systemdenken. In: Die Unternehmung, 41. Jg., Nr. 2, S. 117-133

Chen, P.P. (1976): The Entity-Relationship Model: Towards a Unified View of Data. In: ACM TODS, Vol. 1, No. 1, S. 9-36

Cherry, G. (1987): PAMELA 2: An Ada-Based Object-Oriented Design Method. Reston, Va.

Chrapary, H.J.; Rosenow-Schreiner, X.; Waldhör, K. (1991): Das Elektronische Organisationshandbuch. In: Lutze, R.; Kohl, A. (Hg.): Wissensbasierte Systeme im Büro. Ergebnisse aus dem WISDOM-Verbundprojekt. München, Wien, S. 295-312

Coad, P. (1991): OOA/OOD and OOP. In: JOOP, Vol. 4, No. 2, S. 74-81

Coad, P.; Yourdon, E. (1991): Object Oriented Design. Englewood Cliffs/N.J.

Coad, P.; Yourdon, E. (1990): Object-Oriented Analysis. Englewood Cliffs/N.J.

Cohen, M.D.; March, J.G.; Olsen, J.P. (1976): People, Problems, Solutions and the Ambiguity of Choice. In: March, James G.; Olsen, Johan P. (Hg.):

Ambiguity and Choice in Organizations. Bergen, S. 24-37

Coombs, C.H.; Raiffa, H.; Thrall, R.M. (1954): Some views on mathematical models and measurement theory. In: Psychological Review, S. 132-144

Cox, B.J. (1990): There is a Silver Bullet. In: BYTE, October, S. 209-218

Cox, B. (1992): What if there is a silver bullet. In: JOOP, Vol. 5, No. 3, S. 8 ff.

Croft, W.B. (1987): Representing Office Work with Goals and Constraints. In: Lochovsky, F. (Hg.): Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization. Toronto, S. 13-18

Cunningham, W.; Beck, K. (1986): A diagram for object-oriented programs. In: Meyrowitz, N. (Hg.): OOPSLA '86 - conference proceedings, Portland, Sept. New York, S. 39-43

Curtis, B.; Krasner, H.; Iscoe, N. (1988): A Field Study of the Software Design Process for Large Systems. In: Communication of the ACM, Vol. 31, No. 11, S. 1268-1287

Curtis, B. (1989): Cognitive Issues in Reusing Software Artifacts. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software Reusability. Vol.II: Applications and Experience. Reading/Mass., S. 269-287

Cyert, M.R.; March, J.G. (1963): A Behavioral Theory of the Firm. Englewood Cliffs/N.J.

Daft, R.L.; Weick, K.E. (1984): Toward a Model of Organizations as Interpretation Systems. In: Academy of Management Review. Vol. 9, S. 284-295

Daft, R.L. (1992): Organization Theory and Design. 4. Aufl., Saint Paul/Mass. u.a.

Davis, A.M.; Fairley, R.E.; Incorvaia, A.J. (1990): Case Studies in Software Reuse. In: Knafl, G.J. (Hg.): COMPSAC 90 - the fourteenth annual International Computer Software and Applications Conference; proceedings, Chicago/Ill., Oct. 31 - Nov. 2. Washington/D.C., S. 301-306

Davis, G.B. (1973): Management Information Systems: Conceptual Foundations, Structure and Development. New York

Dean, H.: (1991): Object-oriented design using message flow decomposition. In: JOOP, Vol. 4, No. 2, S. 21-31

De Champeaux, D.; Faure, P. (1992): A comparative study of object-oriented analysis methods. In: JOOP, No. 1, Vol. 5, S. 21-33

Desfray, P. (1990): A Method for Object Oriented Programming: The Class-Relationship Model. In: Bezivin, J.; Meyer, B.; Nerson, J.-M. (Hg.): TOOLS 2 - Technology of object-oriented languages and systems. Paris, S. 121-131

Deßaules, D. (1992): Konzeptionelle Datenmodellierung für ein EDV-gestütztes operatives Absatzführungssystem. München u.a.

Devlin, B.A.; Murphy, P.T. (1988): An architecture for a business and information system. In: IBM System Journal No. 1, S. 60-80

Dicke, W.; Funkel, H. (1987): SOPHO-PLAN - Planungsverfahren für die Bürokommunikation. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 243 - 254

Dittrich, K.R. (1990): Object-Oriented Database Systems: The next Miles of the Marathon. In: Information Systems, Vol. 15, No. 1, S. 161-167

Dittrich, K.R. (1990): Objektorientierte Datenmodelle als Basis komplexer Anwendungssysteme - Stand der Entwicklung und Einsatzperspektiven. In: Wirtschaftsinformatik, Heft 3, S. 228-237

DosSantos, C.S.; Neuhold, E.J.; Furtado, A.L. (1980): A data type approach to the entity relationship model. In: Chen, P.P. (Hg.): Entity-relationship approach to system analysis and design. Amsterdam u.a., S. 103-119

Dunn, M.F.; Knight, J.C. (1991): Software Reuse in an Industrial Setting: a Case Study. In: IEEE. (Hg.): Proceedings of the 13th International Conference on Software Engineering - Austin, Texas, May 13-15, 1991. Washington/ D.C. u.a., S. 329-338

ESPRIT Consortium AMICE (1989): Open System Architecture for CIM. Berlin, Heidelberg u.a.

Eckert, H. (1991): CIM-"Unternehmensmodellierung" mit ADW. In: IT News, Ernst&Young Case Services, Heft 4, S. 7-9

Eddins, W.R.; Sutherland, D.E.; Crosslin, R.L. (1991): Using Modeling and Simulation in the Analysis and Design of Information Systems. In: Sol, H.G.; Van Hee, K.M. (Hg.): Dynamic Modelling of Information Systems. Amsterdam, New York u.a., S. 61-88

Edwards, J. (1989): Lessons learned in more than ten years of practical application of the Object-Oriented Paradigm. London

Eichler, R. (1987): KOM-System - Planung der Einführung neuer Technologien. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung

der Büroarbeit. München, S. 125 - 142

Elgass, P.; Krcmar, H. (1993): Computergestützte Geschäftsprozeßplanung. In: Information Management, Heft 1, S. 42-49

Ellis, C.A.; Bernal, M. (1982): OFFICETALK-D: An experimental office information system. In: SIGOA Newsletter 3, 1 and 2, S. 131-140

Embley, D.W.; Kurtz, B.D.; Woodfield, S.N. (1992): Object-oriented systems analysis - a model-driven aproach. Englewood-Cliffs, N.Y.

Emery, J.E. (1979): Small-scale software components. In: ACM SIGSOFT Software Engineering Notes. Vol. 4, No. 4, S. 18

Emond, J.C. (1988): CIM-OSA concepts demonstrated with an object-oriented language. In: Puente, E.; MacConaill, P. (Hg.): Computer Integrated Manufacturing. Proceedings of the 4th CIM Europe Conference. Berlin, Heidelberg u.a., S. 265-281

Endres, A. (1988): Software-Wiederverwendung: Ziele, Wege und Erfahrungen. In: Informatik Spektrum, 11. Jg., Nr. 2, S. 85-95

Endres, A.; Uhl, J. (1992): Objektorientierte Software-Entwicklung. Eine Herausforderung für die Projektführung. In: Informatik Spektrum, 15. Jg., Nr. 5, S. 254-264

Engel, H. (1990): SAA - Standard der Zukunft. In: Wirtschaftsinformatik, Heft 5, S. 422-428

Esprit Project 56 (1987): Functional Analysis of Office Requirements. Köln

European Space Agency (ESA) (1989): HOOD Reference Manual. Issue 3.0. Nordwijk

Evans, M.W. (1989): The software-factory - a fourth generation software engineering environment. New York

Eversheim, W.K. (1989): The European Approach to an ’Open System Architecture’ for CIM Esprit Project 688 ’CIM-OSA’. In: N.N. (Hg.): CADCAM yearbook 1989/90. London, S. 5-10

Feiten, L.; Hoyer, R.; Kölzer, G. (1987): KSA - Analyse von Kommunikationsstrukturen im Büro. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 143 - 164

Ferstl, O.K.; Sinz, E.J. (1990): Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell. (SOM). In: Wirtschaftsinformatik, Heft 6, S. 566-581

Ferstl, O.K.; Sinz, E.J. (1991): Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik, Heft 6, S. 477-491

Firesmith, D. (1992): Object-oriented requirements analysis and logical design. Chichester

Fischbacher, A. (1986): Strategisches Management der Informationsverarbeitung - Konzeption, Methodik und Instrumente. München

Fischer, G. (1987): Cognitive View of Reuse and Redesign. In: IEEE Software, Vol. 4, No. 4, S. 60-72

Fischer, G.; Henninger, S.; Redmiles, D. (1991): Cognitive Tools for Locating and Comprehending Software Objects for Reuse. In: IEEE. (Hg.): Proceedings of the 13th International Conference on Software Engineering - Austin, Texas, May 13-15, 1991. Washington/D.C. u.a., S. 318-328

Fischer, D.H.; Rostek, L. (1991 a): SFK: A Smalltalk Frame Kit. Concepts and Use. Darmstadt

Fischer, D.H.; Rostek, L. (1991 b): Consistency Rules and Triggers for Thesauri. In: International Classification, Vol. 18, No. 4, S. 212-225

Flavin, M. (1981): Fundamental concepts of information modeling. New York, N.Y.

Fliege, B.; Schmuckner, R. (1987): EPOS - Zur Bewältigung komplexer Bürokommunikationsprojekte. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 91 - 102

Floyd, C. (1989): Softwareentwicklung als Realitätskonstruktion. In: Lippe, W.-M. (Hg.): Software-Entwicklung. Konzepte, Erfahrungen, Perspektiven. Fachtagung, veranstaltet vom Fachausschuß 2.1 der GI, Marburg, 21.-23. Juni. Berlin, Heidelberg u.a., S. 1-20

Forster, M. (1987): Betriebswirtschaftliche Modelle als Antwort auf Probleme der betrieblichen Praxis. In: Schmidt, R.H.; Schor, G. (Hg.): Modelle in der Betriebswirtschaftslehre. Wiesbaden, S. 243-254

Frakes, W.B. (86/87): Software Reuse Through Information Retrieval. In: ACM SIGIR Forum, Vol. 21, No. 1/2, S. 30-36

Frakes, W.B.; Nejmeh, B.A. (1988): An Information System for Software Reuse. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 142-151

Frank, U. (1988): Expertensysteme: Neue Automatisierungspotentiale im Büro-

und Verwaltungsbereich. Wiesbaden

Frank, U. (1989): Zur Implementierung von Wissen über Organisationen. In: Kruse, H.-G.; Frank, U. (Hg.): Praxis der Expertensysteme. München, S. 183-198

Frank, U. (1991): Anwendungsnahe Standards der Datenverarbeitung: Anforderungen und Potentiale. Illustriert am Beispiel von ODA/ODIF undEDIFACT. In: Wirtschaftsinformatik, Heft 2, S. 100-111

Frank, U.; Kronen, J. (1991): Kommunikationsanalyseverfahren - Theoretische Konzepte, Anwendungspraxis und Perspektiven zur Gestaltung von Informationssystemen. Braunschweig

Frank, U. (1992 a): Designing Procedures within an object-oriented Enterprise Model. In: Sol. H. (Hg.): Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems. Delft, S. 365-387

Frank, U. (1992 b): Integrierte Informationssysteme: Konventionelle Modelle und Perspektiven objektorientierter Kommunikation. Illustriert durch Beispiele in C und Smalltalk. In: PIK, 15. Jg., Heft 1, S. 29-35

Frank, U. (1992 c): A Tool Supported Methodology for Designing Multi-Perspective Enterprise Models. In: McGregor, R. (Hg.): Proceedings of the Third Australian Conference on Information Systems, S. 675-690

Frank, U.; Klein, S. (1992 a): Unternehmensmodelle als Basis und Bestandteil integrierter betrieblicher Informationssysteme. Sankt Augustin

Frank, U.; Klein, S. (1992 b): Three integrated tools for designing and prototyping object-oriented enterprise models. Sankt Augustin

Frank, U. (1993 a): An Integrated Environment for Designing Object-Oriented Enterprise Models. In: Whitley, E.A. (Hg.): Proceedings of the First European Conference On Information Systems. Henley, S. 131-140

Frank, U. (1993 b): Eine integrierte Entwicklungsumgebung für den Entwurf objektorientierter Unternehmensmodelle. In: Kurbel, K. (Hg.): Wirtschaftsinformatik 93 - Innovative Anwendungen, Technologie, Integration. Heidelberg, S. 139-152

Frank, U.; Hildebrand, K. (1993): Die Abbildung und Verwaltung temporaler Aspekte in Datenbankmanagement-Systemen. In: HMD, Heft 172, S. 107-119

Freeman, P. (1983): Reusable Software Engineering: Concepts and Research Di-

rections. In: Freeman, P., Wasserman, A.I. (Hg.): Tutorial on software design techniques. Silver Spring/Md., S. 63-78

Frese, E. (1971): Heuristische Entscheidungsstrategien der Unternehmensführung. In: ZfbF 23, S. 283 ff.

Gaube, W.; Lockemann, P.C.; Mayr, H. (1986): Wiederfinden zum Wiederverwenden: Rechnergestützter Modul-Nachweis auf der Basis formaler Spezifikationen. In: Wippermann, H.-W. (Hg.): Software-Architektur und modulare Programmierung. Berichte des German Chapter of the ACM. Stuttgart, S. 66-80

Gaugler, E.; Jacobs, O.; Kieser, A. (1984): Strategische Unternehmensführung. Stuttgart

Geihs, K. (1990): The road to open distributed processing (ODP). Stuttgart

Gibbons, A.J. (1988): A Standards Framework for the Computer-Integrated Enterprise. In: CIM Review, Spring, S. 10-17

Gibbs, S.T. (1990 a): Class Management for Software Communities. In: Communications of the ACM, Vol. 33, No. 9, S. 90-103

Gibbs, S.T. (1990 b): Querying Large Class Collections. In: Tsichritzis, D.C. (Hg.): Object Management. Genf, S. 63-77

Gibson, M.L.; Snyder, C.A.; Carr, H.H. (1990): Implementing a Corporatewide Information Strategy Through CASE. In: Journal of Information Systems Management, summer, S. 8-17

Gillner, R. (1974): Planung, Realisation und Kontrolle in integrierten Gesamtmodellen der Datenverarbeitung. In: Grochla, E. et al. (Hg.): Integrierte Gesamtmodelle der Datenverarbeitung. München, Wien, S. 61-74

Gluck, F.W. (1980): Strategic Choice and Resource Allocation. In: The McKinsey Quarterly, Winter, S. 22-23

Goguen, J.A. (1989): Principles of Parameterized Programming. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software reusability. Vol. 1: Concepts and Models. Reading/Mass., S. 159-225

Goldberg, A.; Robson, D. (1983): Smalltalk-80 - The Language and its Implementation. Reading/Mass.

Goldberg, A.; Robson, D. (1989): Smalltalk-80 - the language. Reading/Mass.

Gongla, P.; Prentice, R.; Ross, R.; Sakamoto, G.; Schumann, M.; Summers, R. (1988): Strategic Information Management Support. (SIMS). IBM Los

Angeles Scientific Center, Report Nr. 1988-2829. Los Angeles

Goodell, M. (1989): What Business Programs do: Recurring Functions in a Sample of Commercial Applications. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software reusability. Vol. 2: Applications and Experience. Reading/ Mass., S. 197-211

Graham, I. (1991): Object oriented methods. Wokingham. u.a.

Grochla, E. (1970): Das Kölner Integrationsmodell. Köln

Grochla, E. et al. (1974) (Hg.): Integrierte Gesamtmodelle der Datenverarbeitung. München, Wien

Grochla, E. (1974 a): Das Konzept des Kölner Integrationsmodells. In: Grochla, E. et al. (Hg.): Integrierte Gesamtmodelle der Datenverarbeitung. München, Wien, S. 35-46

Grochla, E. (1974 b): Das Kölner Integrationsmodell. In: Grochla, E. et al. (Hg.): Integrierte Gesamtmodelle der Datenverarbeitung. München, Wien, S. 189-360

Grochla, E. (1978): Einführung in die Organisationstheorie. Stuttgart

Grochla, E. (1982): Grundlagen der organisatorischen Gestaltung. Stuttgart

Gutenberg, E. (1953): Zum "Methodenstreit". In: ZfhF, S. 327-356

Götzer, K.G. (1990): Optimale Wirtschaftlichkeit und Durchlaufzeit im Büro. Ein Verfahren zur integrierten Optimierung der Büroinformations- und -kommunikationstechnik. Berlin, Heidelberg, New York

Güntzer, U.; Bayer, R.; Sarre, F.; Werner, J.; Myka, A. (1992): Kooperative Zugangssysteme zu Objektbanken. In: Bayer, R.; Härder, T.; Lockemann, P. (Hg.): Objektbanken für Experten. Berlin, Heidelberg, S. 55-82

Habermas, J. (1981): Theorie des kommunikativen Handelns. Bd. 2: Zur Kritik der funktionalistischen Vernunft. Frankfurt/M.

Habermas, J. (1984): Vorstudien und Ergänzungen zur Theorie kommunikativen Handelns. Frankfurt/M.

Hahn, D.; Hölter, E.; Steinmetz, D. (1986): Gesamtunternehmensmodelle als Entscheidungshilfe im Rahmen der Zielplanung, strategischen und operativen Planung. In: Hahn, D.; Taylor, B. (Hg.): Strategische Unternehmensplanung, 2. Aufl. Heidelberg, Wien, S. 565-595

Hasenkamp, U. (1987): Konzipierung eines Bürovorgangssystems. Informations- und Kommunikationstechnik zur aktiven Steuerung von Bürovorgängen.

Habilitationschrift. Köln

Hasenkamp, U.; Syring, M. (1993): Konzepte und Einsatzmöglichkeiten von Workflow-Management-Systemen. In: Kurbel, K. (Hg.): Wirtschaftsinformatik 93 - Innovative Anwendungen, Technologie, Integration. Heidelberg, S. 405-422

Hawryszkiewycz, T.H. (1991): Introduction to systems analysis and design. New York u.a.

Hazzah, A. (1990): Rosetta Stone for Developers? - Model built by IBM and partners charts new course for application development. In: Software Magazine, July, S. 87-96

Heilmann, H.; Sach, W.; Simon, M. (1988): Organisationsdatenbank und Organisationsinformationssystem. In: HMD, Heft 142, S. 119-129

Heilmann, H. (1989): Integration: Ein zentraler Begriff der Wirtschaftsinformatik im Wandel der Zeit. In: HMD, Heft 150, S. 46-58

Heinrich, L.J.; Burgholzer, P. (1987): Informationsmanagement. München

Hembry, D.M. (1990): Knowledge-based systems in the AD/Cycle environment. In: IBM Systems Journal, Vol. 29, No. 2, S. 274-286

Henderson, J.C.; Sifonis, J.G. (1988): The Value of Strategic IS-Planning: Understanding, Consistency, Validity, and IS-Markets. In: MIS Quarterly, No. 2, S. 187 - 200

Henderson-Sellers, B.E. (1990): The Object-Oriented Systems Life Cycle. In: Communications of the ACM, Vol. 33, No. 9, S. 142-159

Henderson-Sellers, B.; Constantine, L.L. (1991): Object-oriented development and functional decomposition. In: JOOP, Vol. 3, No. 5, S. 11-16

Hewitt, C.; Bishop, P.; Steiger, R. (1973): A Universal Modular ACTOR Formalism for Artificial Intelligence. In: o.A. (Hg.): Proceedings of the 3rd International Conference on Artificial Intelligence. Stanford/Ca.

Hewitt, C.; De Jong, P. (1984): Open Systems. In: Brodie, M.L.; Mylopoulos, J.; Schmidt, J.W. (Hg.): On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming. Berlin, Heidelberg u.a., S. 147-164

Hewlett Packard (1991): An Evaluation of Five Object-Oriented Development Methods. Software Engineering Department, HP Laboritories. Bristol

Hilbers, K. (1989): Informationssystem-Architekturen: Zielsetzung, Bestandteile,

Erfolgsfaktoren. Arbeitsbericht Nr. IM2000/CC IM2000/1. St. Gallen

Hildebrand, K. (1990): Software Tools: Automatisierung im Software Engineering. Berlin, Heidelberg u.a.

Hildebrand, K. (1991): Repository-unterstütztes Prototyping. In: HMD, Heft 161, S. 107-116

Hildebrand, K. (1992): Ein Referenzmodell für Informationssystem-Architekturen. In: Information Management, Heft 3, S. 6-12

Hofinger, A. (1991): Der IBM Repository Manager. In: HMD, Heft 161, S. 45-54

Hogg, J. (1987): OTM: A Language for Representing Concurrent Office Tasks. In: Lochovsky, F. (Hg.): Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization. Toronto, S. 10-12

Hong, S.; Goor, G. (1993): A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In: Nunamaker, J.F.; Sprague, R.H. (Hg.): Information Systems: Collaboration Technology, Organizational Systems, and Technolgoy. Proceedings of the 26th HICSS. Los Alamitos, S. 689-698

Honiden, S.; Sueda, N.; Hoshi, A.; Uchihira, N.; Mikame, K. (1988): Software Prototyping with Reusable Components. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 113-119

Horowitz, E.; Munson, J.B. (1984): An Expansive View on Reusable Software. In: IEEE Transactions on Software Engineering, Vol. SE-10, No., S. 477-487

Horowitz, E.; Kemper, A.; Narasimhan, B. (1985): A Survey of Application Generators. In: IEEE Software, Vol. 2, No. 1, S. 40-54

Howden, W.E. (1987): Top-down versus bottom-up reusable code. Berkeley/Ca.

Hsieh, D. (1992): Survey of object-oriented analysis/design methodologies and future CASE frameworks. Menlo Park/Ca.

Huber, G.P. (1990): A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence and Decision Making. In: Academy of Management Review. Vol. 15, S. 47-71

Hutchins, E.L.; Holland, J.D.; Norman, D.A. (1986): Direct Manipulation Interfaces. In: Norman, D.A.; Draper, W. (Hg.): User Centered System Design. Hillsdale/N.J., S. 87-124

IBM (1990): IBM Enterprise Business Process Reference Model. July. Mainz

ISO (1991): Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models, and Services. Recommendation X.500. ISO/IEC 9594-1. Ohne Ort

ISO/IEC (1992): Draft IRDS Conceptual Schema. JTC 1/SC 21 N 7486. New York

Ingari, F. (1991): The object database market: stranger than it needs to be. In: JOOP, Vol. 4, No. 4, S. 16-18

Iscoe, N.; Williams, G.B.; Arango, G. (1991): Domain Modelling for Software Engineering. In: IEEE. (Hg.): Proceedings of the 13th International Conference on Software Engineering - Austin, Texas, May 13-15, 1991. Washington/D.C. u.a., S. 340-343

Ives, B.; Learmonth, G.P. (1984): The information system as a competetive weapon. In: Communications of the ACM, Vol. 26, No. 12, S. 1193-1201

Jacobson, I.; Christerson, M.; Jonsson, P.; Overgaard, G. (1992): Object-Oriented Engineering. A Use Case Driven Approach. Reading/Mass.

Jarillo, J.C.; Stevenson, H.H. (1991): Co-operative Strategies - The Payoffs and the Pitfalls. In: Long Range Planning, Vol. 24, No. 1, S. 64-70

Jensen, K. (1987): Coloured Petri nets. In: Brauer, W.; Reisig, W.; Rozenberg, G. (Hg.): Petri Nets: Central Models and Their Properties. Berlin, Heidelberg u.a., S. 248-299

Jiang, Z.; Bourne, J.R. (1991): A visual programming environment for building Smalltalk-80 views. In: JOOP, Vol. 4, No. 2, S. 32-36

Johnson, R.E.; Foote, B. (1985): Designing Reusable Classes. In: JOOP, Vol. 1, No. 2, S. 22-35

Johnson, R.E. (1988): Designing reusable classes. In: JOOP, Vol. 4, No. 3, S. 22-35

Jones, G. (1988): Methodology/Environment Support for Reusability. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 190-193

Jones, T.C. (1984): Reusability in Programming: A Survey of the State of the Art. In: IEEE Transaction on Software Engineering, Vol. SE-10, No. 5, S. 488-493

Jorysz, H.R.; Vernadat, F.B. (1990 a): CIM-OSA Part 1: total enterprise model-

ling and function view. In: Inter. Journal of Computer Integrated Manufacturing 3, S. 144-156

Jorysz, H.R.; Vernadat, F.B. (1990 b): CIM-OSA Part 2: information view. In: Inter. Journal of Computer Integrated Manufacturing 3, S. 157-167

Kadie, C. (1986): Refinement Through Classes: A Development Methodology for Object-Oriented Languages. Urbana/Il.

Kaiser, G.E.; Garlan, D. (1988): Melding Software Systems from Reusable Building Blocks. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 267-274

Kandt, K. (1984): Pegasus: A Tool for the Acquisition and Reuse of Software Designs. In: COMPSAC 84, The IEEE Computer Society´s Eigth International Computer Software & Applications Conference. Nov. 7-9, Chicago/ Ill. Silver Spring, S. 288-293

Kappel, G.N. (1989): Prototyping in einer objektorientierten Entwicklungsumgebung. In: HMD, Heft 145, S. 116-125

Kappel, G.; Schrefl, M. (1991): Using an Object-Oriented Diagram Technique for the Design of Information Systems. In: Sol, H.G.; Van Hee, K.M. (Hg.): Dynamic Modelling of Information Systems. Amsterdam, New York u.a., S. 121-164

Katz, S.; Richter, C.A.; The, K. (1987): PARIS: A System for Reusing Partially Interpreted Schemas. In: 9th International Conference on Software Engineering, March 30-April 2, Monterey/Ca. Washington/D.C., S. 377-385

Katz, R.L. (1990): Business/enterprise modelling. In: IBM Systems Journal, Vol. 29, No. 4, S. 509-525

Keen, P.G.; Scott Morton, M.S. (1978): Decision Support Systems. An Organizational Perspective. Reading/Mass.

Kelter, U. (1992): H-PCTE - a high-performance object management system for development environments. Hagen

Kieser, A.; Kubicek, H. (1983): Organisation. 2. Aufl. Berlin, New York

King, R. (1989): My Cat is Object-Oriented. In: Kim, W.; Lochovsky, F.H. (Hg.): Object-Oriented Concepts, Databases, and Applications. New York, S. 23-30

Klas, W.; Neuhold, E.J.; Schrefl, M. (1989): Tailoring object-oriented data models through metaclasses. Sankt Augustin

Klein, S. (1989): Theorie der Unternehmungsplanung - Struktur und Beitrag einer anwendungsorientierten Planungstheorie. Stuttgart

Klittich, M. (1988): CIM-OSA: The implementation viewpoint. In: Puente, E.; MacConaill, P. (Hg.): Computer Integrated Manufacturing. Proceedings of the 4th CIM Europe Conference. Berlin, Heidelberg u.a., S. 251-264

Klopprogge, M.R. (1981): An approach to include the time dimension in the entity-relationship model. In: Chen, P.P. (Hg.): Entity-relationship approach to information modelling and analysis, S. 477-512

Klotz, M.S. (1990): Strategieorientierte Planung betrieblicher Informations- und Kommunikationssysteme. Berlin, Heidelberg u.a.

Kortzfleisch, H. (1992): Rechnergestützte organisatorische Gestaltung. Entwicklungsstand und betriebswirtschaftliche Beurteilung. Köln

Kosanke, K.; Vlietstra, J. (1989): CIM-OSA - Its Goals, Scope, Contents and Achievements. In: Commission of the European Communities. (Hg.): ESPRIT'89. Proceedings of the 6th Annual ESPRIT Conference, Brussels, November 27- December 1, 1989. Dordrecht, Boston, London, S. 661-673

Krcmar, H. (1991): Integration in der Wirtschaftsinformatik - Aspekte und Tendenzen. In: Jacob, H.; Becker, J.; Krcmar, H. (Hg.): Integrierte Informationssysteme. Wiesbaden, S. 3-18

Krcmar, H. (1990): Bedeutung und Ziele von Informationssystem-Architekturen. In: Wirtschaftsinformatik, Heft 5, S. 395-402

Kreifelts, T. (1984): DOMINO: Ein System zur Abwicklung arbeitsteiliger Vorgänge im Büro. In: Angewandte Informatik 4/84, S. 137-146

Kreifelts, T.; Seuffert, P.; Woetzel, G. (1987): Ausnahmebehandlung, Verteilung und Adressierung im Vorgangssystem DOMINO. In: Paul, M. (Hg.): GI - 17. Jahrestagung Computerintegrierter Arbeitsplatz im Büro. Berlin; Heidelberg u.a., S. 682 - 696

Kreifelts, T.; Woetzel, G. (1987): Distribution and Error Handling in an Office Procedure System. In: Bracchi, G.; Tsichritzis, D. (Hg.): Office Systems: Methods and Tools. Proceedings of the IFIP TC 8 WG 8.4 1986. Amsterdam, New York u.a., S. 197-208

Kreifelts, T. (1991): Coordination of Distributed Work: From Office Procedures to Customizable Activities. In: Brauer, W.; Hernandez, D. (Hg.): Verteilte Künstliche Intelligenz und kooperatives Arbeiten. Berlin, Heidelberg u.a., S. 148-159

Kronen, J. (1990): Agentur-Informationssysteme und Marktstrukturentwicklungen in der Versicherungswirtschaft. Eine Delphi-Befragung. Arbeitspapiere der GMD, Nr. 480. Sankt Augustin

Kuhn, T.S. (1969): Die Struktur wissenschaftlicher Revolutionen. Frankfurt/M.

Kurpicz, F.J. (1990): Die Organisationsdatenbank strukturiert Unternehmen Büroorganisation ist ohne computergestützte Werkzeuge nicht mehr denkbar. In: Schönecker, H.G.; Nippa, M. (Hg.): Computerunterstützte Methoden für das Informationsmanagement. Baden-Baden, S. 289-307

König, H. (1987): ADV/ORGA-Methode - Gesamtplanung der Bürokommunikation. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 39 - 52

Küttner, M. (1987): Deskriptive Modelle und Handlungsempfehlungen in der Betriebswirtschaftslehre. In: Schmidt, R.H.; Schor, G. (Hg.): Modelle in der Betriebswirtschaftslehre. Wiesbaden, S. 255-272

Lanergan, R.T.; Grasso, C.A. (1984): Software Engineering with Reusable Designs and Code. In: IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, S. 498-501

Ledbetter, L.; Cox, B. (1985): Software-ICs. In:BYTE, Vol. 10, No. 6, S. 307-316

Lederer, A.L.; Mendelow, A.L. (1987): Information Resource Planning: Overcomming Difficulties in Identifying Top Management's Objectives. In: MIS Quarterly, 11, Vol. 3, September, S. 388-399

Lee, S.; Carver, D.L. (1991): Object-oriented analysis and specification: a knowledge base approach. In: JOOP, Vol. 3, No. 5, S. 35-43

Lehner, F. (1991): Software für Organisationsaufgaben. In: Praxis der Informationsverarbeitung und Kommunikation. 14., S. 107-116

Lenz, H. (1987): Entscheidungsmodell und Entscheidungsrealität - Metatheoretische Überlegungen zum logischen Status von Entscheidungsmodellen und dem Problem ihrer Anwendung auf die Realität. In: Schmidt, R.H.; Schor, G. (Hg.): Modelle in der Betriebswirtschaftslehre. Wiesbaden, S. 273-307

Lenz, M.; Schmidt, H.A.; Wolf, P.F. (1988): Software Reuse through Building Blocks. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 100-108

Liebelt, W. (1992): Ablauforganisation, Methoden und Techniken der. In: Frese, E. (Hg.): Handwörterbuch der Organisation. Stuttgart, S. 19-34

Lindblom, C.E. (1964): The Science of "Muddling Through". In: Leavitt, H.J.; Pondy, L.R. (Hg.): Readings in managerial psychology. Chicago, London, S. 61-78

Liskov, B.; Guttag, J. (1986): Abstraction and specification in program development. Cambridge/Mass. u.a.

Love, T. (1988): The Economics of Reuse. In: COMPCON spring 88, 33rd IEEE Computer Society International Conference, San Francisco. Washington/ D.C., S. 238-241

Luhmann, N. (1967): Soziologische Aufklärung. In: Soziale Welt, 18. Jg., S. 97-123

Luhmann, N. (1977): Zweckbegriff und Systemrationalität. 2. Aufl. Frankfurt/M.

Luhmann, N. (1984): Soziale Systeme. Grundriß einer allgemeinen Theorie. Frankfurt/M.

Lyles, M.; Schwenk, C.R. (1992): Top Management, Strategy and Organizational Knowledge Structures. In: Journal of Management Studies. Vol. 29, No. 2, S. 155-174

MacClure, C. (1992): The three Rs of software automation- re-engineering, repository, reusability. Englewood Cliffs/N.J.

Mannino, P. (1987): A Presentation and Comparison of Four Information System Development Methodologies. In: Software Engineering Notes, Vol. 12, No. 2, S. 26-27

Marchand, D.A.; Horton, F.W. (1986): INFOTRENDS - Profiting from Your Information Resources. New York u.a.

Marques, J.M.; Navarro, L.; Sarmiento, M. (1993): ODP Enterprise Models & Functions. Working Paper, Universidad Politecnia Catalunia. Barcelona

Martiny, L.; Klotz, M. (1989): Strategisches Informationsmanagement - Bedeutung und organisatorische Umsetzung. München, Wien

Martial, F. v. (1988): Das elektronische Organisationshandbuch: Anforderungen und Spezifikationen. WISDOM Forschungsbericht. Sankt Augustin

Masiero, P.G. (1988): JSD as an Object-Oriented Design Method. In: Software Engineering Notes. Vol. 13, No. 3, S. 22-23

Mason, R.O.; Mitroff, I.I. (1973): A Program for Research on Management Information System. In: Management Science, Vol.19, No.5, S. 475-487

Mayr, H.C.; Dittrich, K.L.; Lockemann, P.C. (1987): Datenbankentwurf. In: Lok-

kemann, P.C.; Schmidt, J.W. (Hg.): Datenbank-Handbuch. Berlin, Heidelberg u.a., S. 481-557

McDermott, J. (1982): A rule-based configurer of computer systems. In: AI, No. 19, S. 39-88

McGregor, J.D.; Sykes, D.A. (1992): Object-Oriented Software Development. Engineering for Reuse. New York

McIllroy, M.D. (1969): Mass produced Software Components. In: o.H. (Hg.): Software Engineering Concepts and Techniques. Brüssel, S. 88-89

Mercurio, V.J.; Meyers, B.F.; Nisbet, A.M.; Radin, G. (1990): AD/Cycle strategy and architecture. In: IBM Systems Journal, Vol. 29, No. 2, S. 170-188

Mertens, P. (1985): Zwischenbetriebliche Integration der EDV. In: Informatik Spektrum, 8. Jg., S. 81-90

Meyer, B. (1987): Reusability: The Case for Object-Oriented Design. In: IEEE Software, March, S. 50-64

Meyer, B. (1988): Object-Oriented Software Construction. Prentice Hall

Meyer, B. (1989 a): Eiffel - the language; version 2.2. Goleta/Ca.

Meyer, B. (1989 b): Eiffel - the libraries; version 2.2. Goleta/Ca.

Meyer, B. (1989 c): Eiffel - the environment; version 2.2. Goleta/Ca.

Meyer, B. (1990): Lessons from the Design of the Eiffel Libraries. In: Communications of the ACM, Vol. 33, No. 9, S. 68-89

Minsky, M. (1975): A Framework for Representing Knowledge. In: Winston, P.H. (Hg.): The Psychology of Computer Vision. New York, S. 211-277

Mockler, R.J.; Dologite, D.G. (1987): Knowledge-based systems for strategic corporate planning. Oxford/Ohio

Mori, R. (1990): On Superdistribution. In:BYTE, Sept., S. 346

Moss, S.P. (1988): CIM-OSA: The enterprise model. In: Puente, E.; MacConaill, P. (Hg.): Computer Integrated Manufacturing. Proceedings of the 4th CIM Europe Conference. Berlin, Heidelberg u.a., S. 237-250

Mullin, M. (1989): Object-oriented program design - with examples in C++. Reading/Mass. u.a.

Muralidharan, S.; Weide, B.W. (1988): On distributing programs built from reusable software components. Columbus/Ohio

Mylopoulus, J.; Levesque, H.J. (1984): An Overview of Knowledge Representation. In: Brodie,M.L.;Mylopoulos,J.;Schmidt,J. (Hg.): On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming. Berlin/Heidelberg., S. 3-17

Müller-Pleuß, J. (1992): Organisationshandbuch. In: Frese, E. (Hg.): Handwörterbuch der Organisation. Stuttgart, S. 1506-1514

Müller-Merbach, H. (1989): Komprehensive Informationssysteme und Allgemeine Betriebswirtschaftslehre. In: ZfB, 59. Jg., Heft 10, S. 1023-1045

Müller-Nobiling, H.M. (1980): Organisationshandbuch. In: Grochla, E. (Hg.): Handwörterbuch der Organisation. 2. Aufl., Stuttgart, S. 1557-1563

Neighbors, J.M. (1989): Draco: A Method for Engineering Reusable Software Systems. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software Reusability. Reading/Mass., S. 295-319

Nelson, T.H. (1987): Literary machines - The report on, and of, project Xanadu. South Bend, Ind.

Nielsen, R. (1988): Cooperative Strategy. In: Strategic Management Journal, Vol. 9, S. 475-492

Nielsen, K. (1988): An Object-Oriented Design Methodology for Real-Time System in Ada. San Diego/Ca.

Niemeier, J. (1988): Computer Integrated Business. Erfahrungen aus Experimenten mit Integrationsansätzen. In: Office Management Nr. 5, S. 6-12

Nierstrasz, O. (1989): A Survey of Object-Oriented Concepts. In: Kim, W.; Lochovsky, F.H. (Hg.): Object-Oriented Concepts, Databases, and Applications. New York, S. 3-21

Nierstrasz, O.; Tsichritzis, D.C. (1989): Integrated Office Systems. In: Kim, W.; Lochovsky, F.H. (Hg.): Object-Oriented Concepts, Databases, and Applications. New York, S. 199-215

Nierstrasz, O.; Dami, L.; De Mey, V.; Stadelmann, M.; Tsichritzis, D.; Vitek, J. (1990): Visual Scripting: Towards Interactive Construction of Object-Oriented. In: Tsichritzis, D. C. (Hg.): Object Management. Genf, S. 315-331

Nießen, H.; Zwickler, P. (1990): Datenmodellierung im Versicherungsunternehmen. In: HMD 153, S. 62-73

Nippa, G.; Schönecker, H.G. (1987): Bedeutung und Einordnung computergestützter Büroananlyse- und -gestaltungsmethoden. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München

Nippa, M. (1988): Gestaltungsgrundsätze für die Büroorganisation. Berlin

Nippa, M.; Schönecker, H.G. (1990): Computergestützte Büroanalyse- und Gestaltungsmethoden - Organisationshilfen für die Praxis. In: Schönecker, H.G.; Nippa, M. (Hg.): Computerunterstützte Methoden für das Informationsmanagement. Baden-Baden, S. 9-49

Norman, D.A. (1986): Cognitive Engineering. In: Norman, D.A.; Draper, W. (Hg.): User Centered System Design. Hillsdale/N.J., S. 31-61

Nüttgens, M.; Scheer, A.W. (1993): ARIS-Navigator - hypermediabasierte Dokumentation unternehmensweiter Informationsmodelle für Beratung und Einführungsunterstützung. In: Information Management, Heft 1, S. 20-26

Odell, J.J. (1992 a): Dynamic and multiple classification. In: JOOP, Vol. 4, No. 8, S. 45-48

Odell, J.J. (1992 b): Modelling objects using binary- and entity-relationship approaches. In: JOOP, Vol. 5, No. 3, S. 12-18

Odemer, W. (1987): COMplan - Methode zur Planung und Gestaltung der Bürokommunikation. In: Schönecker, H.G.; Nippa, M. (Hg.): Neue Methoden zur Gestaltung der Büroarbeit. München, S. 53 - 70

Österle, H.; Brenner, W.; Hilbers, K. (1990): Forschungsprogramm IM2000: Umsetzung von Informationssystem-Architekturen. St. Gallen

Olle, T.W.; Hagelstein, J.; MacDonald, I.G. (1988): Information Systems Methodologies: a framework for understanding. Wokingham, Reading u.a.

OMG (Hg.) (1992): The OMG Object Model. Framingham/Mass.

Onuegbe, E.O. (1988): Software Classification as an Aid to Reuse: Initial Use as Part of a Rapid Prototyping System. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 161-167

Ortmann, G. (1976): Unternehmensziele als Ideologie. Köln

Ortner, E. (1983): Aspekte einer Konstruktionssprache für den Datenbankentwurf. Darmstadt

Page, T.W.; Berson, S.E.; Cheng, W.C.; Muntz, R.R. (1989): An Object-Oriented Modeling Environment. In: Meyrowitz, N. (Hg.): Object-Oriented Programming: Systems, Languages and Applications. New York, S. 287-296

Peters, L.; Schultz, R. (1993): The Application of Petri-Nets in Object-Oriented Enterprise Simulations. In: Nunamaker, J.F.; Sprague, R.H. (Hg.): Information Systems: Collaboration Technology, Organizational Systems, and

Technolgoy. Proceedings of the 26th HICSS. Los Alamitos, S. 390-398

Petri, C.A. (1962): Kommunikation mit Automaten. Schriften des Instituts für Instrumentelle Mathematik. Bonn

Pfeffer, J. (1978): The Ambiguity of Leadership. In: McCall jr., M.W.; Lombardo, M.M. (Hg.): Leadership: Where else can we go? Durham/N.C., S. 13-34

Pfeffer, J. (1981): Power in Organizations. Marshfield/Mass.

Pintado, X. (1990): Selection and Exploration in an Object-Oriented Environment: The Affinity Browser. In: D. C. Tsichritzis. (Hg.): Object Management. Genf, S. 79-88

Popien, C.; Spaniol, O.; De Meer, J. (1991): Systementwurf mit ODP. In: PIK, 14. Jg., Heft 4, S. 192-201

Porter, M.E. (1985): Competitive Advantage. New York

Porter, M.E.; Millar, V.E. (1985): How Information Gives You Competitive Advantage. In: HBR, Vol. 63, July/August, S. 149-160

Prahalad, C.K.; Hamel, G. (1990): The Core Competence of the Corporation. In: HBR, Vol. 68, May/June, S. 79-91

Prieto-Diaz, R.; Freeman, P. (1987): Classifying Software for Reusability. In: IEEE Software, Vol., No. 1, S. 6-16

Prieto-Diaz, R. (1989): Classification of Reusable Modules. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software reusability. Vol. 1: Concepts and Models. Reading/Mass., S. 99-123

Prieto-Diaz, R. (1991): Implementing Faceted Classification for Software Reuse. In: Communications of the ACM, Vol. 33, No. 5, S. 88-97

Prince, T.R. (1970): Information Systems for Management Planning and Control. Homewood/Ill.

Prinz, W. (1989): Application of the X.500 directory by office systems. München, Wien

Prinz, W. (1993): TOSCA - Providing organisational information to CSCW applications. Erscheint in: De Michelis, G. (Hg.): Proceedings of the European Conference on Computer Supported Cooperative Work - Mailand.

Pröfrock, A.K.; Tsichritzis, D.; Müller, G.; Ader, M. (1989): ITHACA: an Integrated Toolkit for Highly Advanced Computer Applications. In: D. C. Tsichritzis. (Hg.): Object Oriented Development. Genf, S. 321-344

Rajlich, V.; Silva, J. (1987): Two Object-Oriented Decomposition Methods. Detroit

Ranganathan, S.R. (1967): Prologomena to Library Classification. Bombay

Reichwald, R. (1990): EDV-gestützte Werkzeuge der Organisationsanalyse. In: Zahn, E. (Hg.): Organisationsstrategie und Produktion. München, S. 389-423

Reisig, W. (1990): Petrinetze - eine Einführung. Berlin. u.a.

Remer, A. (1989): Organisationslehre. Berlin, New York

Rentsch, T. (1982): Object-Oriented Programming. In: SIGPLAN Notices, Vol. 17, No. 12

Rescher, N.; Urquhart, A. (1971): Temporal Logic. Berlin

Robinson, P. (1992): Object-Oriented Design. London u.a.

Rockart, J.F. (1979): Chief Executives Define their own Data Needs. In: HBR, Vol. 57, No. 2, S. 81-92

Rockart, J.F.; Short, J.E. (1991): The Networked Organization and the Management of Interdependence. In: Scott Morton, M.S. (Hg.): The Corporation of the 1990s: Information Technology and Organizational Transformation. New York, Oxford, S. 189-219

Rotenberg, J.J.; Saloner, G. (1991): Interfirm Competition and Collaboration. In: Scott Morton, M.S. (Hg.): The Corporation of the 1990s: information technology and organizational transformation. New York, Oxford, S. 95-121

Rumbaugh, J. et al. (1991): Object-oriented modeling and design. Englewood Cliffs/N.J. u.a.

Rumbaugh, J. (1993): Objects in the Constitution: Enterprise Modeling. In: JOOP, Vol. 5, No. 8, S. 18-24

Sagawa, J.M. (1990): Repository Manager technology. In: IBM Systems Journal, Vol. 29, No. 2, S. 209-227

Santifaller, M. (1990): TCP/IP und NFS in Theorie und Praxis - UNIX in lokalen Netzen. Bonn. u.a.

Savage, C.M. (1990): Fifth generation management - integrating enterprises through human networking. Bedford/Mass.

Schäfer, G.; Wolfram, G. (1986): FAOR-Methode zur Analyse und Bewertung

von Kosten- und Nutzenfaktoren von Bürosystemen. In: HMD, S. 54-64

Schank, R.C. (1972): Conceptual Dependency: A Theory of Natural Language Understanding. In: Cognitive Psychology, Bd.3, No.4, S. 552-631

Scheer, A.W. (1988 a): Unternehmensmodell (UDM) als Grundlage integrierter Informationssysteme. In: ZfB Nr. 10, S. 1091-1114

Scheer, A.W. (1988 b): Wirtschaftsinformatik - Informationssysteme im Industriebetrieb. Berlin u.a.

Scheer, A.W. (1990 a): Enterprise-Wide Data Modelling. Information Systems in Industry. Berlin, Heidelberg u.a.

Scheer, A.W. (1990 b): EDV-orientierte Betriebswirtschaftslehre - Grundlagen für ein effizientes Informationsmanagement. Berlin u.a.

Scheer, A.W. (1991): Architektur integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung. Berlin, Heidelberg u.a.

Scheer, A.W.; Hars, A. (1992): Extending Data Modeling to Cover the Whole Enterprise. In: Communications of the ACM, Vol. 35, No. 9, S. 166-171

Scheschonk, G. (1990): Design/CPN - das Simulationswerkzeug für hierarchische Petri-Netze auf der Basis von Design/OA. In: Balzert, H. (Hg.): CASE - Systeme und Werkzeuge. Mannheim, S. 199-220

Schmidt, R.H.; Schor, G. (1987): Modelle in der Betriebswirtschaftslehre. Wiesbaden

Schmidt, J.W. (1987): Datenbankmodelle. In: Lockemann, P.C.; Schmidt, J.W. (Hg.): Datenbankhandbuch. Berlin, Heidelberg u.a., S. 1-83

Schmidt, R.H.; Schor, G. (1987): Modell und Erklärung in den Wirtschaftswissenschaften. In: Schmidt, R.H.; Schor, G. (Hg.): Modelle in der Betriebswirtschaftslehre. Wiesbaden, S. 9-36

Schmidt, G. (1991): Methoden und Techniken der Organisation. 9. Aufl. Gießen

Schneider, D.J. (1978): Ziele und Mittel in der Betriebswirtschaftslehre. Wiesbaden

Schönecker, H.G.; Nippa, M. (1987): Neue Methoden zur Gestaltung der Büroarbeit. München

Schöllmann, M. (1986): Eine Organisationsdatenbank schafft Ordnung. In: Computerwoche, 13. Jg., Nr. 43, S. 22-24

Schussel, G. (1990): The Promise and the Reality of AD/Cycle. In: Datamation

Sept. 15, S. 69-73

Scott Morton, M.S. (1986): Strategy formulation methodologies. Cambridge/ Mass.

Searle, J. (1980): Minds, brains and programs. In: The Behavioral and Brain Sciences 3, S. 417-457

Seidewitz, E.; Stark, M. (1987): Towards a General Object-Oriented Software Development Methodology. In: Peterson, G.E. (Hg.): Object-Oriented Computing - Tutorial. Vol. 2: Implementations. Washington/D.C., S. 16-29

Selby, R.W. (1988): Empirically Analyzing Software Reuse in a Production Environment. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 176-189

Selby, R.W. (1989): Quantitative Studies of Software Reuse. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software Reusability. Vol. 2: Applications and Experience. Reading/Mass., S. 213-233

Shlaer, S.; Mellor, S.J. (1992): Object lifecycles- modeling the world in states. Englewood Cliffs/N.J.

Simon, H.A. (1949): Administrative Behavior. A Study of Decision-Making Processes in Administrative Organizations. New York

Simon, H.A. (1964): The Corporation: Will It Be Managed by Machines. In: Leavitt, H.J.; Pondy, L.R. (Hg.): Readings in managerial psychology. Chicago/London, S. 592-617

Sims, H.P.; Gioia, D.A. (1986): The Thinking Organization. San Francisco

Sinz, E.J. (1987): Datenmodellierung betrieblicher Probleme und ihre Unterstützung durch ein wissensbasiertes Entwicklungssystem. Habilitationsschrift. Regensburg

Soley, R.M. (1992): Object Management Architecture Guide. Framingham/Mass.

Sommerville, I. (1989): Software engineering. Wokingham u.a.

Sowa, J.F.; Zachman, J.A. (1992): Extending and formalizing the framework for information systems architecture. In: IBM Systems Journal, Vol. 31, No. 3, S. 590-616

Spenke, M. (1992): GINA reference manual. Arbeitspapiere der GMD, Nr. 615. Sankt Augustin

Sutherland, J. (1983): An Office Analysis and Diagnosis Methodology. MIT, Laboratory for Computer Science. Cambridge/Mass.

Szyperski, N. (1990): Die Informationstechnik und unternehmensübergreifende Logistik. In: Adam, P. et al. (Hg.): Integration und Flexibilität. Wiesbaden, S. 79-96

Teufel, B. (1992): Ressourcen-Beschreibung und -Findung: Die Basis für rechnergestützte Gruppenarbeit und erfolgreiches Lean Management. In: Informationstechnik, 34. Jg., Heft 6, S. 358-367

Tracz, W. (1988 a) (Hg.): Software reuse - emerging technology. Washington/ D.C.

Tracz, W. (1988 b): Software Reuse: Motivators and Inhibitors. In: Tracz, W. (Hg.): Tutorial: Software reuse - emerging technology. Washington/D.C., S. 62-67

Tsichritzis, D.C. (1982): Form Management. In: Communications of the ACM, Vol. 25, No.7, S. 453-478

Tsichritzis, D.C. (1989): DB Ideas for KBMS. In: Schmidt, J.W.; Thanos, C. (Hg.): Foundations of Knowledge Base Management. Berlin, Heidelberg u.a., S. 499-501

Tsichritzis, D.C.; Nierstrasz, O.M. (1989): Directions in Object-Oriented Research. In: Kim, W.; Lochovsky, F.H. (Hg.): Object-Oriented Concepts, Databases, and Applications. New York, S. 523-536

Tsichritzis, D.C.; Gibbs, S. (1990): Towards Integrated Software Communities. In: Tsichritzis, D.C. (Hg.): Object Management. Genf, S. 1-9

Tulowitzki, U. (1991): Anwendungsarchitekturen im strategischen Informationsmanagement. In: Wirtschaftsinformatik, Heft 2, S. 94-99

Turing, A.M. (1950): Computing Machinery and Intelligence. In: Mind, Bd. 59, S. 433-460

Uhl, J.; Schmid, H.A. (1990): A systematic catalogue of reusable abstract data types. Berlin u.a.

Velho, A.V.; Carapuca, R. (1992): SOM: A Semantic Object Model - Towards an Abstract, Complete and Unifying Way to Model the Real World. In: Sol. H. (Hg.): Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems. Delft, S. 65-93

Venkatraman, N.; Short, J.E. (1990): Strategies for Electronic Integration: From Order-Entry to Value-Added Partnerships at Baxter. Center for Information Systems Research Working Paper, No. 210

Venkatraman, N. (1991): IT-Induced Business Reconfiguration. In: Scott Morton,

M.S. (Hg.): The Corporation of the 1990s: information technology and organizational transformation. New York, Oxford, S. 122-158

Vetter, M. (1987): Strategy for data modelling - application- and enterprise-wide. Chichester u.a.

Walsh, J.P.; Ungson, G.R. (1991): Organizational Memory. In: Academy of Management Review. Vol. 16

Ward, J.; Griffiths, P.; Whitmore, P. (1990): Strategic planning for information systems. Chichester. u.a.

Wasserman, A.I.; Pircher, P.A.; Muller, R.J. (1990): The Object-Oriented Structured Design Notation for Software Representation. In: Computer 23, No. 3, S. 50-63

Wegner, P. (1989): Capital-Intensive Software Technology. In: Biggerstaff, T.J.; Perlis, A.J. (Hg.): Software Reusability. Reading/Mass., S. 43-97

Wegner, P. (1991): Perspectives on object-oriented design. Providence/R.I.

Weick, K.E. (1985): Der Prozeß des Organisierens. Frankfurt/M.

Wiborny, W. (1990): Datenmodell "Betriebliches Vorschlagswesen" der BMW AG. In: HMD 152, S. 76-89

Wiederhold, G.; Wegner, P.; Ceri, S. (1990): Towards megaprogramming. Stanford/Ca.

Wileden, J.C.; Wolf, A.L.; Rosenblatt, W.R.; Tarr, P.L. (1991): Specification-Level Interoperability. In: Communications of the ACM, Vol. 43, No. 5, S. 72-87

Winograd, T.; Flores, F. (1986): Understanding computers and cognition - a new foundation for design. Norwood/N.J.

Wirfs-Brock, R.J.; Wilkerson, B. (1990): Designing Object-Oriented Software. Englewood Cliffs, N.J.

Wirfs-Brock, R.J.; Johnson, R.E.: Surveying Current Research in Object-Oriented Design. In: Communications of the ACM, Vol. 42, No. 9, S. 104-124

Wiseman, C. (1985): Strategy and Computers: Information Systems as Competitive Weapons. Homewood/Ill.

Wittgenstein, L. (1980): Philosophische Untersuchungen. 2. Aufl. Frankfurt/M.

Wollnik, M. (1986): Implementierung computergestützter Informationssysteme. Berlin, New York

Wollnik, M. (1988): Aktionsfelder des Informationsmanagements. In: Jahresbericht der GMD 1987, S. 148-166

Wong, E.; Katz, R.H. (1980): Logical design and schema conversion for relational and DBTG databases. In: Chen, P.P. (Hg.): Entity-relationship approach to system analysis and design. Amsterdam u.a., S. 311-321

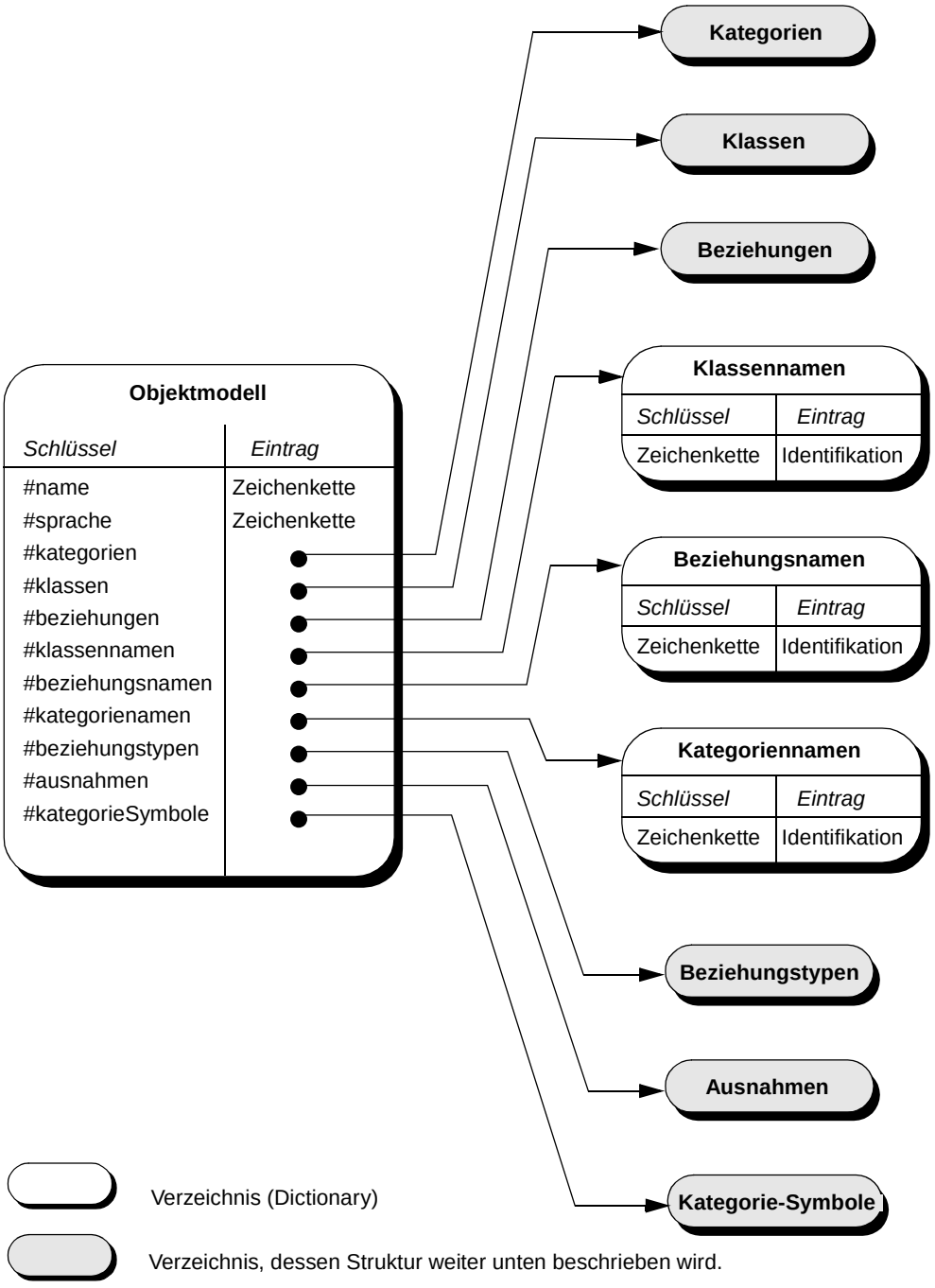
Wright, G.H. (1974): Erklären und Verstehen. Frankfurt/M.

Zachman, J.A. (1987): A framework for information systems architecture. In: IBM Systems Journal, Vol. 26, No. 3, S. 277-293

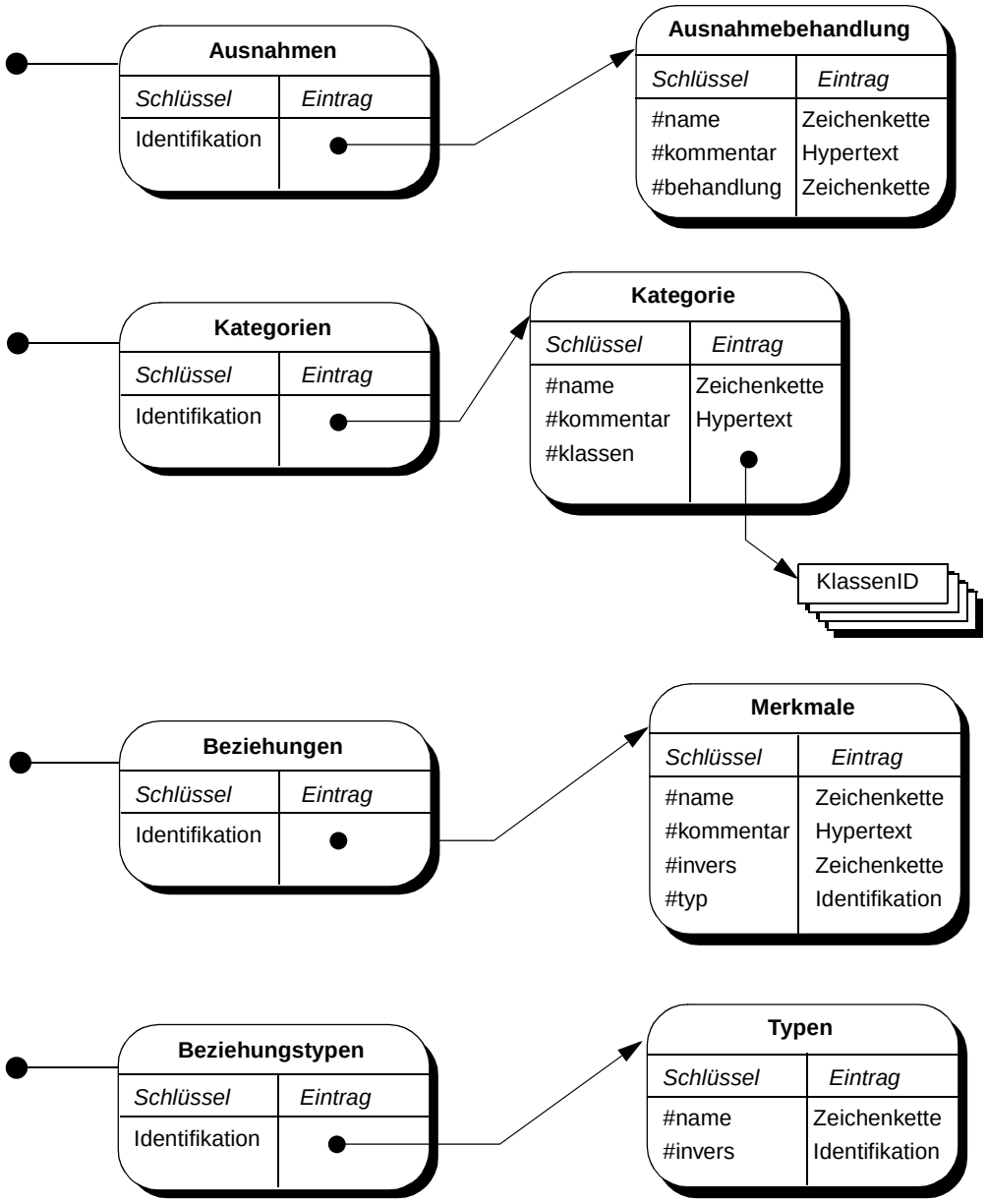
Zelm, M. (1989): Enterprise Modelling. In: ESPRIT Consortium AMICE (Hg.): Open System Architecture for CIM. Berlin, Heidelberg u.a.

Anhang

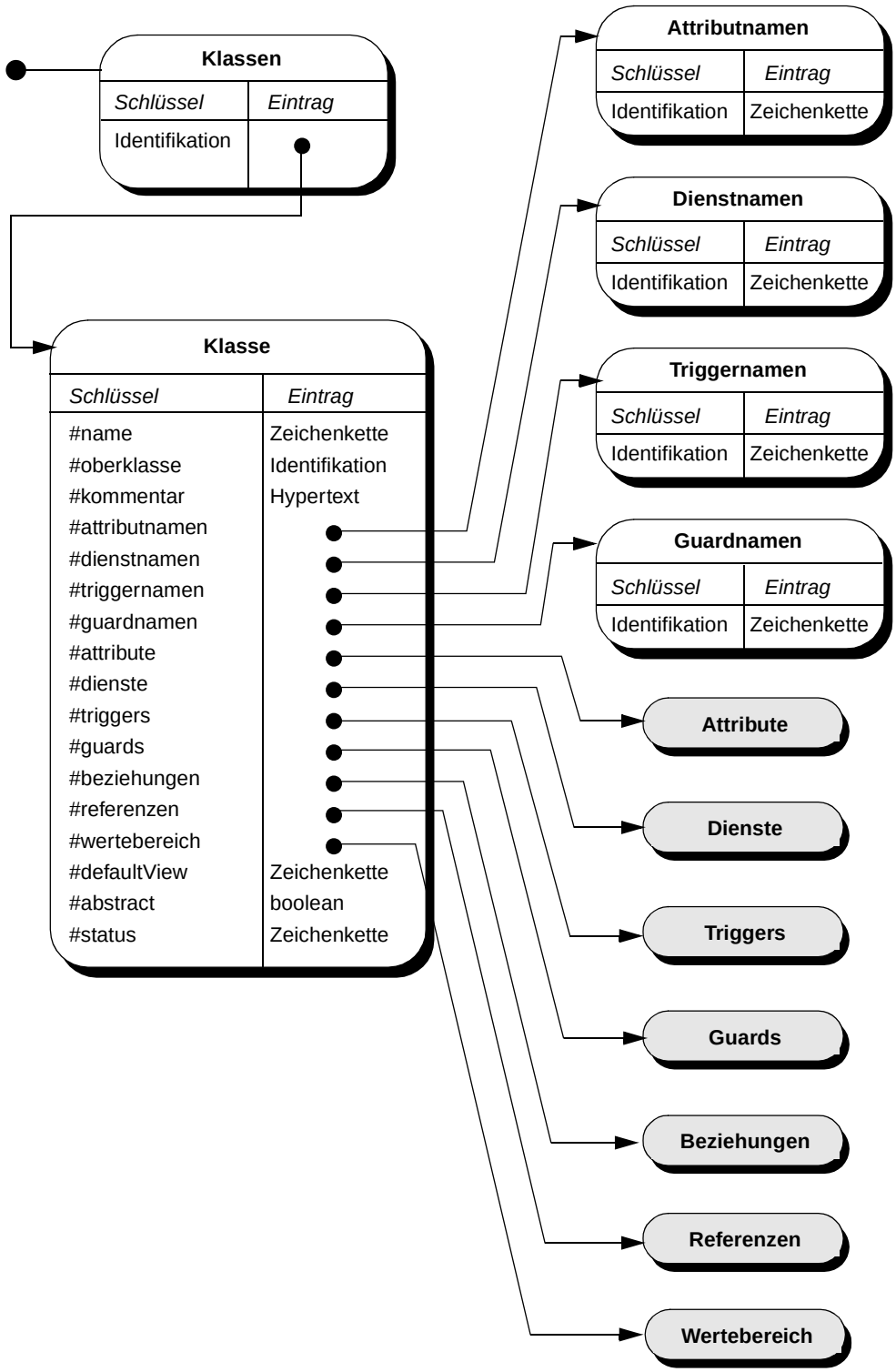
I. Die Abbildung von Objektmodellen im OMD

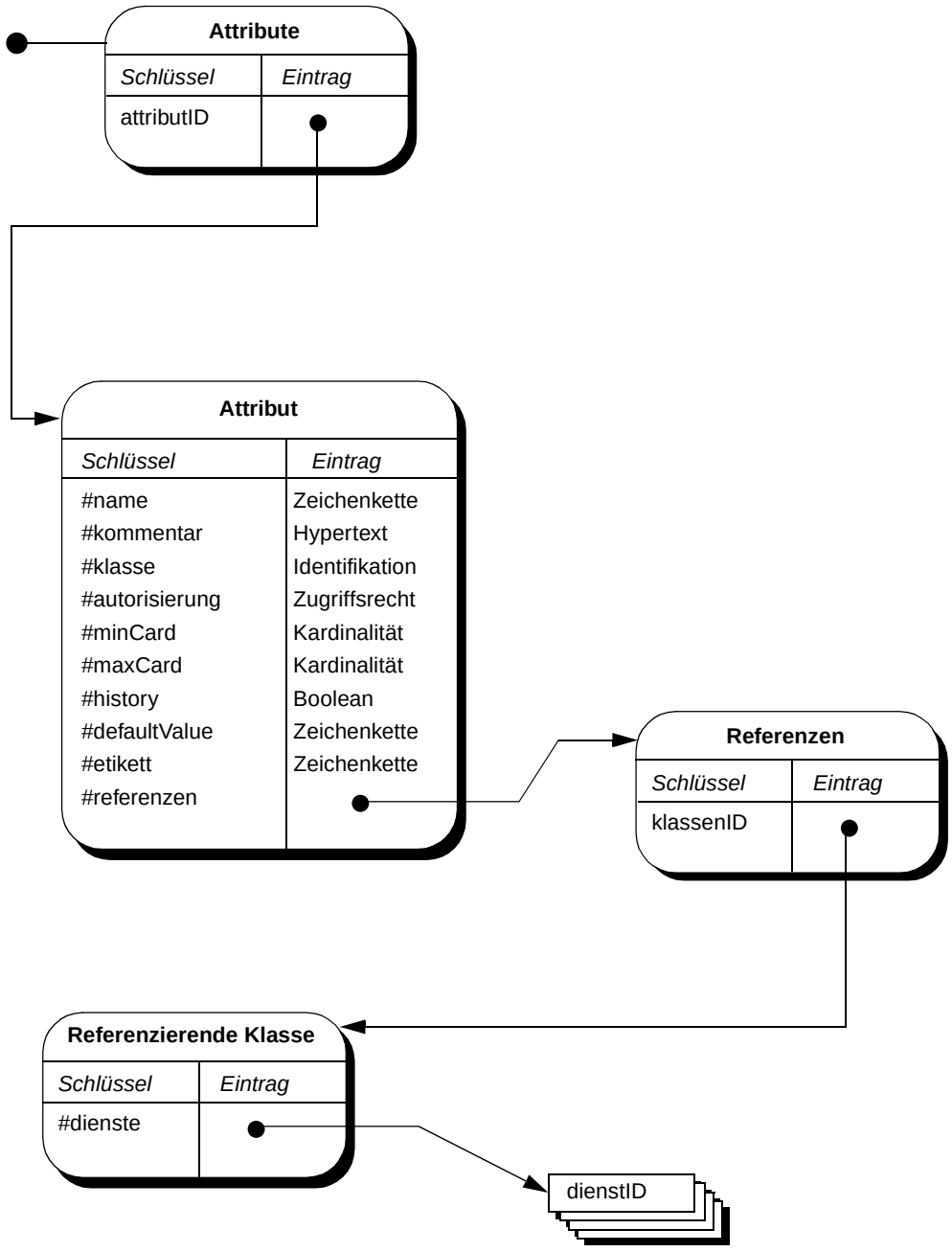


Schlüssel, die mit einem "#" beginnnen, markieren konstante Bezeichner, es existiert also genau ein Schlüssel dieser Art. Andere Bezeichnungen (z.B. "Zeichenkette) stellen Variablen dar. In diesem Fall sind all jene konkreten Schlüssel möglich, die durch den Wertebereich der Variable abgedeckt sind.

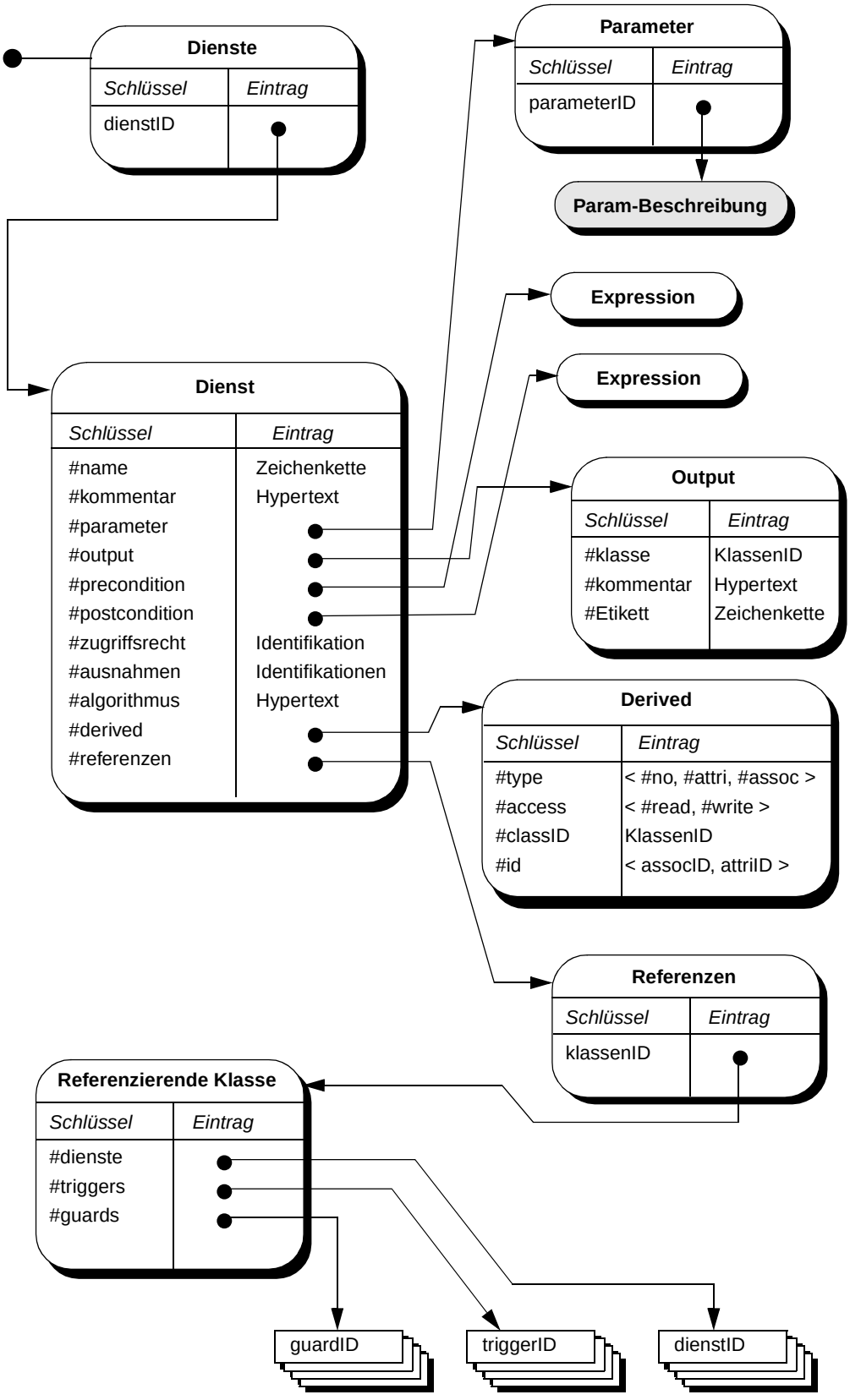


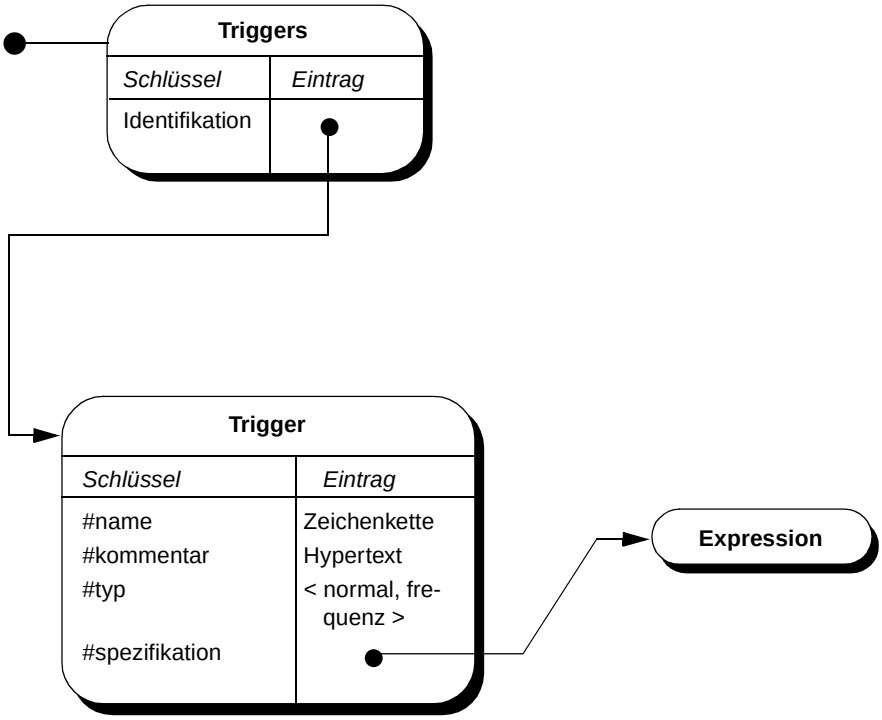
Liste (implementiert als OrderedCollection)

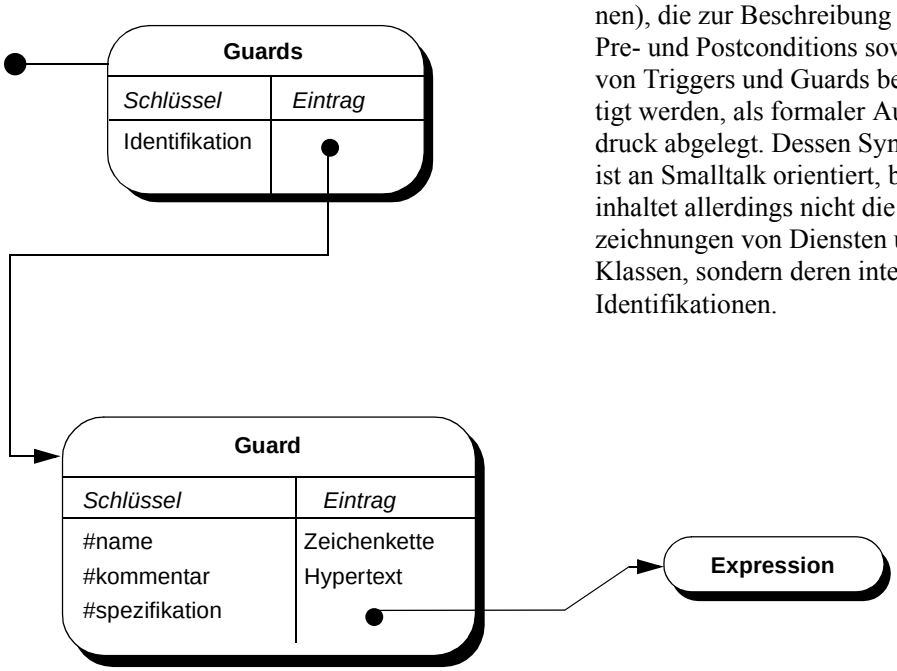

Klassen
Schlüssel
Eintrag
Identifikation
Klasse
Schlüssel
Eintrag
#name
Zeichenkette
#oberklasse
Identifikation
#kommentar
Hypertext
#attributnamen
#dienstnamen
#triggernamen
#guardnamen
#attribute
#dienste
#triggers
#guards
#beziehungen
#referenzen
#wertebereich
#defaultView
Zeichenkette
#abstract
boolean
#status
Zeichenkette
Attributnamen
Schlüssel
Eintrag
Identifikation
Zeichenkette
Dienstnamen
Schlüssel
Eintrag
Identifikation
Zeichenkette
Triggernamen
Schlüssel
Eintrag
Identifikation
Zeichenkette
Guardnamen
Schlüssel
Eintrag
Identifikation
Zeichenkette
Attribute
Dienste
Triggers
Guards
Beziehungen
Referenzen
Wertebereich



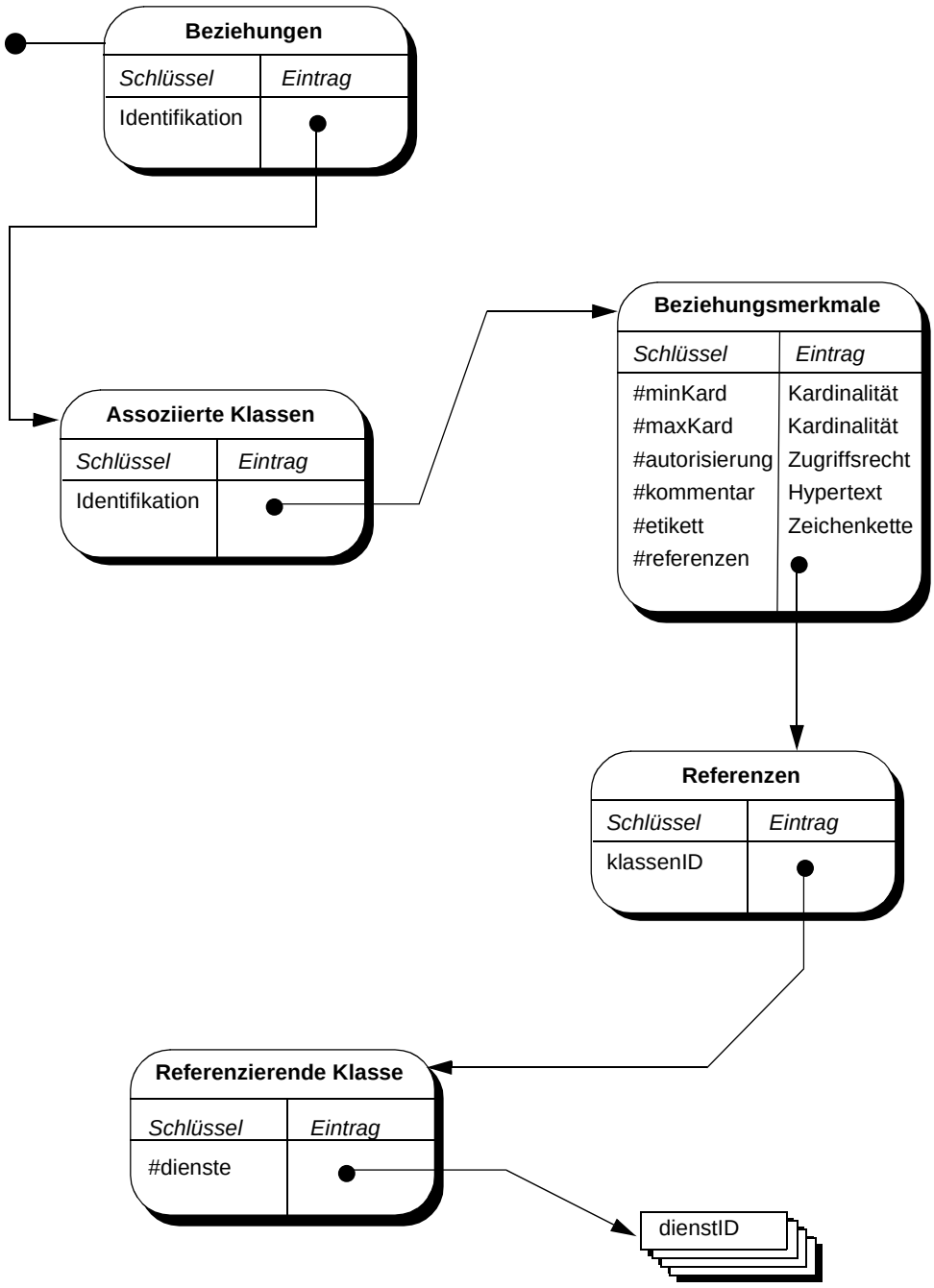
Attribute können gegenwärtig nur bei der Spezifikation von Diensten referenziert werden. Da sich dies aber ändern mag, wird ein Verzeichnis verwendet, in dem auch andere Konstrukte (wie etwa Trigger oder Guards) eingetragen werden können. Die zulässigen Werte von "Zugriffsrecht" sind durch die in IV.2.1.1.1.1 definierten Tupel bestimmt.


Dienste
Schlüssel
Eintrag
dienstID
Parameter
Schlüssel
Eintrag
parameterID
Param-Beschreibung
Expression
Expression
Dienst
Schlüssel
Eintrag
#name
Zeichenkette
#kommentar
Hypertext
#parameter
#output
#precondition
#postcondition
#zugriffsrecht
Identifikation
#ausnahmen
Identifikationen
#algorithmus
Hypertext
#derived
#referenzen
Output
Schlüssel
Eintrag
#klasse
KlassenID
#kommentar
Hypertext
#Etikett
Zeichenkette
Derived
Schlüssel
Eintrag
#type
< #no, #attri, #assoc >
#access
< #read, #write >
#classID
KlassenID
#id
< assocID, attriID >
Referenzen
Schlüssel
Eintrag
klassenID
Referenzierende Klasse
Schlüssel
Eintrag
#dienste
#triggers
#guards
guardID
triggerID
dienstID

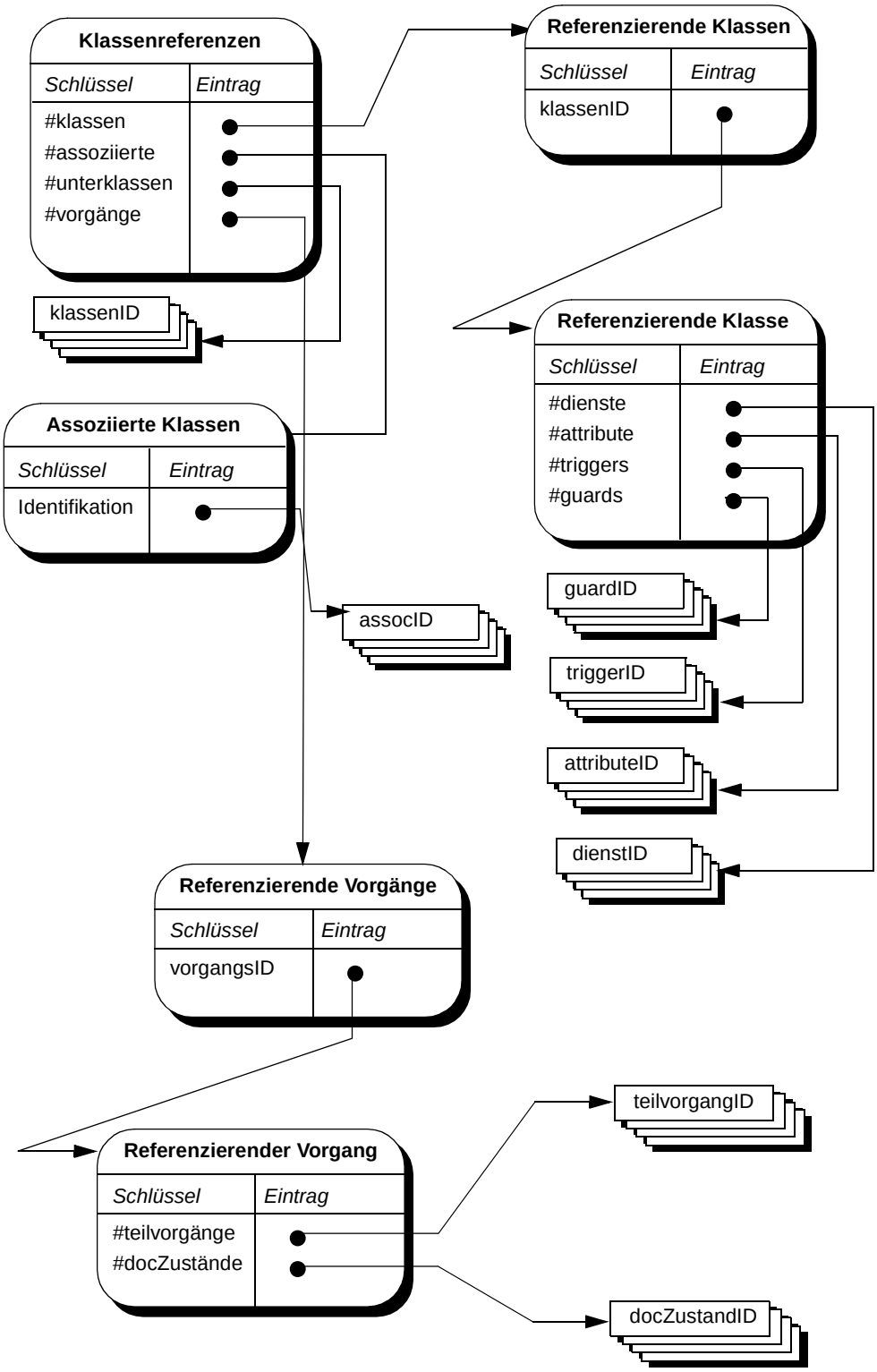


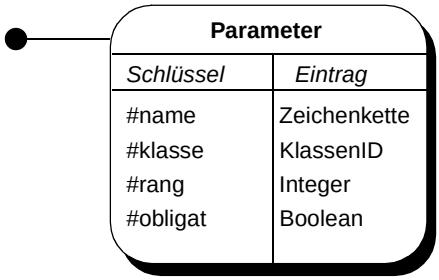


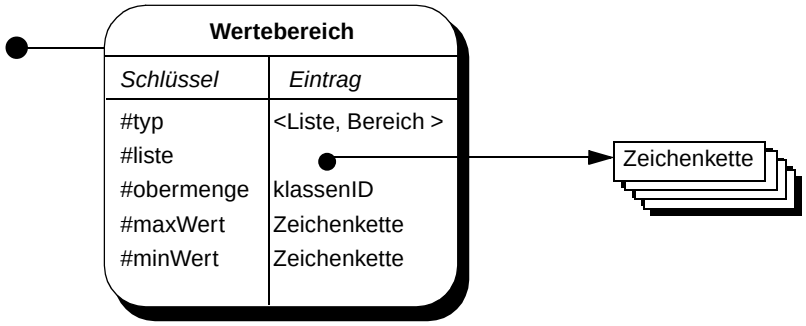
Um für zukünftige Erweiterungen offen zu sein, werden die Bedingungen (und ggfs. Aktionen), die zur Beschreibung von Pre- und Postconditions sowie von Triggers und Guards benötigt werden, als formaler Ausdruck abgelegt. Dessen Syntax ist an Smalltalk orientiert, beinhaltet allerdings nicht die Bezeichnungen von Diensten und Klassen, sondern deren interne Identifikationen.



Wenn eine Beziehung zwischen den Instanzen zweier Klassen definiert wird, werden damit auch Referenzen auf die zum Zugriff auf die assoziierten Objekte generierten Dienste etabliert. Ein entsprechender Eintrag in ein Verzeichnis erleichtert die Wahrung der damit verbundenen referentiellen Integrität.


Klassenreferenzen
Schlüssel
Eintrag
#klassen
#assoziierte
#unterklassen
#vorgänge
Referenzierende Klassen
Schlüssel
Eintrag
klassenID
klassenID
Referenzierende Klasse
Schlüssel
Eintrag
#dienste
#attribute
#triggers
#guards
Assoziierte Klassen
Schlüssel
Eintrag
Identifikation
assocID
guardID
triggerID
attributeID
dienstID
Referenzierende Vorgänge
Schlüssel
Eintrag
vorgangsID
Referenzierender Vorgang
Schlüssel
Eintrag
#teilvorgänge
#docZustände
teilvorgangID
docZustandID





Ein Wertebereich kann entweder mit Hilfe einer Aufzählung der zulässigen Werte definiert werden (dazu dient der Eintrag unter #liste) oder mit Hilfe eines Intervalls. Dazu wird eine Obermenge verwendet, die durch Rückgriff auf eine dazu geeignete Klasse beschrieben wird.

II. Mögliche Strukturierung ausgewählter Klassen für die Modellierung der Systemverwaltung

Falls eine Klasse über eine Oberklasse verfügt, die ebenfalls zu dem Teilmodell gehört, ist deren Bezeichnung in Klammern unter der der Klasse aufgeführt. Abstrakte Klassen sind kursiv gesetzt.

Periphere Geräte

Klasse	Merkmal	Kardinalität	Formalisierungshinweis
Präsentationsgerät (PG) (Hardware)	Schnittstelle	1,*	Zeichenkette
	Max. Anzahl Graustufen	1,*	positive ganze Zahl
	Max. Anzahl Farben	1,*	positive ganze Zahl
	Technologie	1,1	Zeichenkette
PG mit nicht flüchtigem Medium (PG)	Medium	1,1	< Normalpapier, Folie, Spezialpapier >
	Medium-Maße	1,*	Tupel [positive ganze Zahl, positive ganze Zahl | * (für "endlos")]
	Korrespondenzschrift	1,1	boolescher Wert
Zeichenorientierter Drucker (*PG mit nicht flüchtigem M.*)	Seiten pro min./Korrespondenzdruck	1,1	positive ganze Zahl
	Seiten pro min./Listendruck	1,1	positive ganze Zahl
	Zeichensatz	1,1	Zeichenkette
	Schrift	1,*	Zeichenkette
Grafik-Drucker (*PG mit nicht flüchtigem M.*)	Max. Auflösung/cm^2	1,*	positive ganze Zahl
	Seiten pro min./geringste Auflösung	1,1	positive ganze Zahl
	Seiten pro min./höchste Auflösung	1,1	positive ganze Zahl
	Postscript		
Plotter	Meter pro min.	1,1	positiver ganze Zahl
PG mit flüchtigem Medium (*PG*)	Max. Auflösung	1,1	Tupel von zwei positiven ganzen Zahlen
	Max. Bits pro Pixel	1,1	positive ganze Zahl
	videofähig	1,1	boolescher Wert

Periphere Geräte

Klasse	Merkmal	Kardinalität	Formalisierungshinweis
Bildschirm (*PG mit flüchtigem M.*)	Auflösung pro cm^2	1,1	positive ganze Zahl
	Diagonale in cm	1,1	positive ganze Zahl
Projektionsgerät (*PG mit flüchtigem M.*)	Max. Diagonale in cm	0,1	positve ganze Zahl
	Min. Leinwandabstand	0,1	positve ganze Zahl
	Max. Leinwandabstand	0,1	positve ganze Zahl
	Schnittstelle	1,1	Zeichenkette
Lautsprecher (*Hardware*)	Leistung in Watt	1,1	positive ganze Zahl
	Impedanz	1,1	positive ganze Zahl
	Klangqualität	1,1	< mäßig, befriedigend, hoch >
Eingabegerät (*Hardware*)	Schnittstelle	1,1	Zeichenkette
Tastatur (*Eingabege.*)	Zeichensatz (Nationalität)	1,1	Zeichenkette
	Zehnerblock	1,1	boolescher Wert
Maus (*Eingabege.*)	Anzahl Tasten	1,1	positive ganze Zahl
	Unterlage erforderlich	1,1	boolescher Wert
Trackball (*Eingabege.*)	Anzahl Tasten	1,1	positive ganze Zahl
	In Tastatur integriert	1,1	boolescher Wert
Mikrophon (*Eingabege.*)	Aufnahmequalität	1,1	< mäßig, befriedigend, hoch >
	Max. Entfernung in m	1,1	positive ganze Zahl
Kamera (*Eingabege.*)	Min. Brennweite in mm	1,1	positive ganze Zahl
	Max. Brennweite in mm	1,1	positive ganze Zahl
	Mikrophon integriert	1,1	boolescher Wert
	Bewegtbilder	1,1	boolescher Wert
	Standbilder	1,1	boolescher Wert
	Min. Verschlußzeit	1,1	positive ganze Zahl
	Max. Verschlußzeit	1,1	positive ganze Zahl
	Farbe	1,1	boolescher Wert
	Max. Auflösung in Pixel	1,1	positive ganze Zahl

	Klasse	Merkmal	Kardinalität	Formalisierungshinweis
Periphere Geräte	*Massenspeicher* (*Hardware*)	Datentransferrate (KB pro sek.)	1,1	positive ganze Zahl
		transportabel	1,1	boolescher Wert
	Direktzugriff-Speicher (*Massensp.*)	Zugriffszeit in ms	1,1	positive ganze Zahl
	Festmedium-Speicher (*Direktzugriff-Speicher*)	Max. Kapazität in KB	1,1	postive ganze Zahl
	Direkt-Wechselmedium (*Direktzugriff-Speicher*)	Max. Kapazität/Medium in KB	1,1	positive ganze Zahl
	Sequentiell-Speicher (*Massensp.*)	Max. Kapazität/Medium in KB	1,1	positive ganze Zahl
Netzinfrastruktur	Netzwerk (*Hardware*)	Topologie	1,1	< Bus, Ring, Stern >
		Medium	1,1	< Koaxial, Glasfaser, Infrarot >
		Kapazität (MB/sek.)	1,1	positive ganze Zahl
		Protokoll	1,*	Zeichenkette
	Externer Anschluß (*Hardware*)	Dienst	1,1	< ISDN, Datex-P, Datex-L, VBN, ATM >
		Übertragungsrate (KB/sek.)		positive ganze Zahl
	Scanner (*Eingabege.*)	Farbe	1,1	boolescher Wert
		Graustufen	1,1	boolescher Wert
		Auflösung pro cm^2	1,1	positive ganze Zahl
		Seitengröße	1,1	Tupel [positive ganze Zahl, positive ganze Zahl | * (für "endlos")]
		Seiten pro min.		positive ganze Zahl

III. Mögliche Strukturierung ausgewählter Klassen für die Modellierung der strategischen Perspektive

Konzept	**Merkmal**	**Formalisierungshinweis**
Wertsystem	Wertketten	Liste von Wertketten
	Gültig von	Datum
	Kommentar	Multimedia
	Beispiele	Liste instanzierter Wertssysteme
Wertkette	Bezeichnung	Zeichenkette
	Gehört zu	Wertsystem
	Gültig von	Datum*
	Aktivitätsgruppen	Liste Aktivitätsgruppen
	Ressourcen*	Liste von Ressourcen
	Erbrachte Leistungen*	Liste erbrachter Leistungen
	Gesamtumsatz*	Geldbetrag
	Gesamtkosten*	Geldbetrag
	Gewinn*	Geldbetrag
	Kommentar	Multimedia
	Beispiele	Liste instanzierter Wertketten
Aktivitätsgruppe	Bezeichnung	Zeichenkette aus vorgegebener Menge von Zeichenketten
	Gehört zu	Wertkette
	Gültig von*	Datum
	Typ	< primär | unterstützend >
	Beinhaltet	Liste von Aktivitäten
	Vorgängergruppe	Aktivitätsgruppe | nil
	Nachfolgergruppe	Aktivitätsgruppe | nil
	Ressourcen*	Liste von Ressourcen
	erbrachte Leistungen*	Liste erbrachter Leistungen
	Kosten*	Geldbetrag
	Umsatz*	Geldbetrag
	relative Kosten*	Prozentsatz
	relative Wertschöpfung	Prozentsatz
	Kommentar	Multimedia
	Beispiele	Liste instanzierter Aktivitätsgruppen
	Differenzierungsaktivitäten	Liste von Aktivitäten

Konzept	**Merkmal**	**Formalisierungshinweis**
Infrastruktur (*Aktivitätsgruppe*)	Rechtsform	Zeichenkette aus vorgegebener Liste
	Führungsstile	Liste von Beschreibungen (z.B. Management By-Konzepte)
	Unternehmensphilosophie bzw. -kultur	Multimedia
Personalwesen (*Aktivitätsgruppe*)	Rollenprofile	Liste von Tupeln < Bezeichnung, Qualifikationsstufe, Text >
	Weiterbildung	Liste von Weiterbildungsmaßnahmen
	Anreizsysteme	Liste von Anreizsystemen
	Personalentwicklung	Liste von Personalentwicklungsmaßnahmen
Forschung und Entwicklung (*Aktivitätsgruppe*)	Charakterisierung	Liste von F&E-Beschreibungen
	Anteil der extern erworbenen F&E-Leistungen	Prozentsatz
	Anteil der Auftragsforschung	Prozentsatz
Beschaffung (*Aktivitätsgruppe*)	Beschaffungsmarktforschung	Liste von Tripeln < Verfahren, Informationsquelle, Kosten >
	Bedarfsermittlung	Liste von Verfahren bzw. Ansätzen
	Qualitätskontrolle	Liste von Verfahren
	Lieferantenpflege	Liste von Maßnahmen
Eingangslogistik (*Aktivitätsgruppe*)	Lager	Liste von Lagerbeschreibungen
	Transportsystem	Liste von Transportarten
Leistungserstellung (*Aktivitätsgruppe*)	Charakterisierung	Charakterisierung der Leistungserstellung, beispielsweise durch Liste von Fertigungsverfahren
Ausgangslogistik (*Aktivitätsgruppe*)	Lager	Liste von Lagerbeschreibungen
	Transportsystem	Liste von Transportarten
	Durchschnittliche Lieferzeit	Zeit

Konzept	Merkmal	Formalisierungshinweis
Marketing&Vertrieb (*Aktivitätsgruppe*)	Marktsegmente	Liste von Marktsegmenten
	Marktforschung	Liste von Marktforschungsbeschreibungen
	Vertriebskanäle	Liste von Vertriebskanälen
	Werbemedien	Liste von Werbemedien
	Richtlinien für die Werbung	Text
	Verkaufsförderung	Liste von Verkaufsförderungsmaßnahmen
	Konditionenpolitik	Text
Kundendienst (*Aktivitätsgruppe*)	Gewährleistungen	Liste von Gewährleistungsvereinbarungen
	Betreuungsarten	< Kundenbesuche, Telefongespräche, Korrespondenz ... >
	Beanstandungsstatistik	Statistische Angaben über Beanstandungshäufigkeiten und Beanstandungsursachen
	Durchschnittliche Kosten pro Beanstandung	Geldbetrag
	Durchschnittliche Reparaturzeit	Zeit
Aktivität	Bezeichnung	Zeichenkette
	Übergeordnete Aktivität	Aktivitätsgruppe | Aktivität
	Beinhaltet	Liste von Aktivitäten
	Gültig von (*)	Datum
	Ausrichtung	< direkt, indirekt, qualitätssichernd >
	Ressourcen (*)	Liste von Ressourcen
	Erbrachte Leistungen (*)	Liste erbrachter Leistungen
	Kosten*	Geldbetrag
	Wert*	Geldbetrag
	Beziehungen zu anderen Aktivitäten	Liste von Tupeln < Beziehungsart, Liste von Aktivitäten >
	Kommentar	Multimedia
	Beispiele	Liste instanzierter Aktivitäten

Konzept	Merkmal	Formalisierungshinweis
Generische Strategie	Bezeichnung	Zeichenkette aus Liste vorgegebener Zeichenketten
	Beschreibung	Multimedia
	Gültig seit	Datum
	Zugeordnete Ziele	Liste von Zielen
	Schlüsselaktivitäten	Liste von Zeigern auf Aktivität
Ressource	Bezeichnung	Zeichenkette
	Bezogen von	Aktivität | extern
	Benötigte Kapazität	Menge
	Vorhandene Kapazität	Menge
	Mengeneinheit	Symbol aus vorgegebener Liste
	Bewertungsansatz	< Buchwert, Wiederbeschaffungswert, Gebrauchswert .... >
	Kosten	Geldbetrag
	Kommentar	Multimedia
Human-Ressource (Ressource)	Rolle	Zeichenkette
	Arbeitsmarktsituation	Text | Auszeichnung auf Ordinalskala (z.B.: "günstig", "unkritisch", "problematisch")
	Qualifikationsstand	< herausragend, durchschnittlich, unterentwickelt >
	Investitionen in Weiterbildung	Geldbetrag
Kapital (Ressource)	Finanzierungsart	Zeichenkette aus vorgegebener Liste
	Kapitalbindung	Geldbetrag | Auszeichnung auf Ordinalskala (z.B: < hoch, mittel, gering >
Immobilie (Kapital)	Art	Zeichenkette aus vorgegebener Liste
	Ausstattung	Zeichenkette aus vorgegebener Liste | Auszeichnung auf Ordinalskala (z.B.: < sehr gut, mittelmäßig, unzureichend >

Konzept	Merkmal	Formalisierungshinweis
Maschine (Kapital)	Maschinentyp	Zeichenkette aus vorgegebener Liste
	Funktion	Multimedia
	Beurteilung der Technologie	< fortschrittlich, durchschnittlich, rückständig >
	Human-Ressourcen	Liste von Human-Ressourcen
	Werkstoffe	Liste von Werkstoffen
	Auswahlkriterien	Text, Regeln
IT-Hardware (Kapital)	Hardware-Typ	Zeichenkette aus vorgegebener Liste
	Funktion	Multimedia
	Beurteilung der Technologie	< fortschrittlich, durchschnittlich, rückständig >
	Software	Liste der Software, die Hardware nutzt
	Human-Ressourcen	Liste von Human-Ressourcen
	Auswahlkriterien	Text, Regeln
IT-Software (Kapital)	Art	Zeichenkette aus vorgegebener Liste
	Funktion	Multimedia
	Beurteilung der Technologie	< fortschrittlich, durchschnittlich, rückständig >
	Hardware	Liste der genutzten Hardware
	Portierbarkeit	Liste alternativer Hardware
	Information	Liste benötigter Information
Werkstoff (Ressource)	Art	Zeichenkette aus vorgegebener Liste
	Mengeneinheit	Zeichenkette aus vorgegebener Liste
	Kosten pro Mengeneinheit	Geldbetrag
	Preisschwankungen	< hoch, mittel, gering >
	Abhängigkeit	< hoch, mittel, unbedeutend >
	Lagerhaltung in Tagesbedarfsmengen	Mengenangabe
	Substitute	Liste von Werkstoffen

Konzept	Merkmal	Formalisierungshinweis
Information (Ressource)	Speichermedium	Zeichenkette aus vorgegebener Liste
	Quelle	< intern, extern >
	Wichtigkeit	< notwendig, wichtig, entbehrlich >
	Kosten für Beschaffung bzw. Pflege	Geldbetrag | < hoch, mittel, gering >
Weiterbildungsmaßnahme	Bezeichnung	Zeichenkette
	Durchgeführt von	< intern, extern >
	Ziel	Text
	Zielgruppe	Liste von Rollen
	Kosten pro Maßnahme	Geldbetrag
	Erfahrungen	Text
Personalentwicklungs-maßnahme	Bezeichnung	Zeichenkette
	Ziel	Text
	Zielgruppe	Liste von Rollen
	Anforderungen	Liste von Kriterien (Text)
	Anreiz-Mix	Liste von Anreizen (Text)
	Erfahrungen	Text
Marktsegment	Produkt oder Produktgruppe	Zeichenkette
	Phase im Lebenszyklus	Zeichenkette aus vorgegebener Liste
	Zielgruppe	Zeichenkette
	Marktanteil	Prozentsatz
	Wettbewerbsposition	< dominierend, stark, durchschnittlich, schwach >
	Zukünftige Potentiale	< zunehmend, gleichbleibend, abnehmend >
	Konkurrenten	Liste von Konkurrenten
	Potentielle Konkurrenten	Liste potentieller Konkurrenten
Verkaufsförderung	Maßnahme	< Händlerschulung, Verkäuferseminar, Fachkonferenz ... >
	Produkt bzw. Produktgruppe	Zeichenkette
	Kosten	Geldbetrag | < hoch, mittel, gering >
	Einsatz pro Jahr	Anzahl

Konzept	Merkmal	Formalisierungshinweis
Werbemedium	Bezeichnung	Zeichenkette
	Marktsegment	Marktsegment
	Relative Reichweite	Prozentsatz
	Kosten pro Kontakt	Geldbetrag
	Einsatz pro Jahr	Anzahl
	Erfahrungen	Multimedia
Marktforschungs-beschreibung	Marktsegment	Marktsegment
	Auswahlverfahren	< Zufallsauswahl, Klumpenauswahl, geschichtete Auswahl >
	Erhebungsverfahren	< persönliches Interview, Telefonumfrage, schriftliche Befragung, Verhaltensbeobachtung, Bestandsbewegungen ... >
	Analyseverfahren	< Einfache statistische Verfahren, Verfahren der induktiven Statistik >
	Kosten	Geldbetrag | < hoch, mittel, gering >
	Erfahrungen	Multimedia
Vertriebskanal	Typ	< direkt (Versand, Fabrikverkauf ...), indirekt (Großhändler, Einzelhändler ...)>
	Bindung	< notwendig, eng, ersetzbar >
Lagerbeschreibung	Standort	Zeichenkette
	Lieferorte	Liste von Tupeln < Orte, die von dem Lager bedient werden, Entfernung >
	Kapazität (in relevanter Mengeneinheit)	Kapazitätsangabe
	Kosten (pro relevanter Mengeneinheit)	Geldbetrag
	Beurteilung der eingesetzten Technologie	< fortschrittlich, durchschnittlich, rückständig >

Konzept	Merkmal	Formalisierungshinweis
Transportart	Transportmittel	Kombination aus < Eigener LKW, Fremd-LKW, Spedition, Bahn, Flugzeug, Schiff ... >
	Administration	< Eigenregie, Spedition, sonstige >
	Ort	Zeichenkette
	Lager	Lagerbeschreibung
	Entfernung	Kilometer
	Transportdauer	Zeit
	Kosten (pro Kilometer und relevanter Mengeneinheit)	Geldbetrag
Unternehmen	Bezeichnung	Zeichenkette
	Ort	Zeichenkette
	Wertkette	Wertkette
Lieferant (Unternehmen)	Produkte	Liste von Produkten
	Abhängigkeit	< hoch, mittel, gering >
	Alternativen	Liste von Lieferanten
Konkurrent (Unternehmen)	Marktsegmente	Liste von Marktsegmenten
	Relative Stärke	< bedrohlich, hoch, gleichstark, schwächer >
Potentieller Konkurrent (Konkurrent)	Möglicher Markteintritt	Liste von Tupeln aus < Datum, Kommentar>, Marktsegment
Substitutionskonkurrent (Konkurrent)	Neigung der Kunden zu wechseln	Liste von Tupeln aus < hoch, indifferent, gering >, Marktsegment
Erbrachte Leistung	Bezeichnung	Zeichenkette
	Erstellt für	Aktivität | extern | Marktsegment
	als Ressource	Ressource
	Kalkulation	Text
	Mengeneinheit	Symbol aus vorgegebener Menge
	Menge	Mengenangabe
	Wert	Geldbetrag
	Kommentar	Multimedia

Konzept	Merkmal	Formalisierungshinweis
F&E-Beschreibung	Typ	Kombinationen aus < Forschung, Entwicklung, Auftragsforschung >
	Gegenstand	Liste von FuE-Gegenständen (z.B.: Software, Aggregate, Teile, Maschinen ...)
	Eigenes Know-How	< herausragend, durchschnittlich, unterentwickelt >
	Abhängigkeit von einzelnen Mitarbeitern	< hoch, mittel, gering >
	Ressourcenbindung	< kapitalintensiv, arbeitsintensiv, materialintensiv >
	Beurteilung der eingesetzten Technologie	< fortschrittlich, durchschnittlich, rückständig >
Fertigungsverfahren	Produkt- bzw. Produktgruppenbezeichnung	Zeichenkette
	Fertigungsart	< Werkstattfertigung, Reihenfertigung, Fließbandfertigung >
	Losgröße	< Massenfertigung, Sortenfertigung, Serienfertigung, Einzelfertigung >
	Fertigungstiefe	< hoch, mittel, gering >
	Beurteilung der eingesetzten Technologie	< fortschrittlich, durchschnittlich, rückständig >
	Eigenes Know-How	< herausragend, durchschnittlich, unterentwickelt >
	Ressourcenbeanspruchung	< kapitalintensiv, arbeitsintensiv, materialintensiv >