Die objektorientierte Modellierung von Geschäftsprozessen

Die erweiterte Object Behavior Analysis (OBA++) als
Ergänzung der Unified Modeling Language (UML)

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaft
(doctor rerum oeconomicarum)
am Fachbereich Wirtschafts- und Sozialwissenschaften
der Bergischen Universität (GH) Wuppertal

Vorgelegt von Diplom-Ökonom
OLIVER LINSSEN
aus Mönchengladbach

Wuppertal, im November 2002

Erstgutachter: Prof. Dr. M.-R. Wolff

Zweitgutachter: Prof. Dr. W. Matthes

Inhalt

Abkürzungen

Abkürzung	Definition
ACM	Association for Computing Machinery
ADT	Abstrakter Datentyp
AI	Artificial Intelligence
AL	Abteilungsleiter
ANSI	American National Standards Institute
ARIS	Architektur integrierter Informationssysteme
BON	Business Object Notation
BPI	Business Process Improvement
BPR	Business Process Reengineering
CA	Computer Associates
CASE	Computer Aided Software Engineering
CCITT	International Telegraph & Telephone Consultative Committee
CIB	Computer Integrated Business
CIS	Computergestütztes Informationssystem (auch als Plural verwendet)
CM	Conceptual Model
CRC	Classes, Responsibilities and Collaborations
CSCW	Computer Supported Cooperative Work
d	day (lat. *dies*)
DBMS	Database Management System
DBW	Die Betriebswirtschaft
DDL	Data Definition Language
DLL	Dynamic Link Library
dtv	Deutscher Taschenbuch-Verlag
DV	Datenverarbeitung

Abkürzung	Definition
eBNF	extended Backus-Naur Form
ECA	Event-Condition-Action
ECOOP	European Conference on Object-Oriented Programming
EMISA	Entwicklungsmethoden für Informationssysteme und deren Anwendung
EMK	Ereignis-Methoden-Kette
EPC	Event driven Process Chain
EPK	Ereignisgesteuerte Prozesskette
ER	Entity-Relationship
ESPRIT	European Strategic Programme of Research and Development in Information Technology
et. al.	et alii
etc.	et cetera
ETH	Eidgenössische Technische Hochschule
EVA	Eingabe-Verarbeitung-Ausgabe
FCFS	Fist-Come-First-Served
FiBu	Finanzbuchhaltung
FIPS	Federal Information Processing
GE	Geldeinheit
GI	Gesellschaft für Informatik
GL	Gruppenleiter
GP	Geschäftsprozess (auch als Plural verwendet)
GPM	Geschäftsprozessmodellierung
GR	Geschäftsregel
h	hour (lat. *hora*)
HIPO	Hierarchy and Input-Process-Output
HMD	Handbuch der modernen Datenverarbeitung
I. u. K.	Information und Kommunikation

Abkürzung	Definition
IAO	Institut für Arbeitswissenschaft und Organisation
IBM	International Business Machines
IDEF	Integration Definition
IEEE	Institute of Electrical and Electronics Engineers
IKS	Informations- und Kommunikationssystem
ISO	International Organization for Standardization
IT	Informationstechnik
ITU	International Telecommunication Union
IuK	Informations- und Kommunikations(-Technik)
Iwi	Institut für Wirtschaftsinformatik
KIM	Kölner Integrationsmodell
KL	Kunden-Lieferanten-(Beziehung)
KOZ	Kürzeste Operationszeit
Ltd.	Limited
MEMO	Multi Perspective Enterprise Modeling
MEMO-OML	Multi Perspective Enterprise Modeling – Object Modeling Language
MEMO-OrgML	Multi Perspective Enterprise Modeling – Organisation Modeling Language
MobIS	Modellierung betrieblicher Informationssysteme
MOF	Meta-Object-Facility
MSA	Modern Structured Analysis
MSC	Message Sequence Chart
NATO	North Atlantic Treaty Organisation
NOAC	Next Operation As Customer
o. O.	ohne Ort
OA	Objektansatz
OBA	Object Behavior Analysis

Abkürzung	Definition
OBA++	Object Behavior Analysis plus plus
OCL	Object Constraint Language
ODBMS	Object Data Base Management System
oEPK	objektorientierte Ereignisgesteuerte Prozesskette
OMG	Object Management Group
OML	OPEN Modeling Language
OMT	Object Modeling Technique
OO	Objektorientierung, objektorientiert
OOAD	Object Oriented Analysis & Design
ooGPM	objektorientierte Geschäftsprozessmodellierung
OOPSLA	Object-oriented Programming, Systems, Languages and Applications
ooSe	objektorientierte Softwareentwicklung
OOSE	Object Oriented Software Engineering
OOTC	Object-Oriented Technology Center
OP	Offener Posten
OPEN	Object-oriented Process, Environment and Notation
OSA	Object-oriented Systems Analysis
p	Wahrscheinlichkeit
q. v.	quod videas
RDD	Responsibility Driven Design
REFA	Reichsausschuss für Arbeitszeitermittlung (Verband für Arbeitsstudien und Betriebsorganisation e.V.)
RUP	Rational Unified Process
RTF	Revision Task Force
s	second (lat. *secunda*)
SAP	Systeme, Anwendungen, Produkte
SB	Sachbearbeiter

Abkürzung	Definition
SDL	Specification and Design Language
SE	Software Engineering
SGE	Strategische Geschäftseinheit
SIGPLAN	Special Interest Group for Programming Languages
SIGSOFT	Special Interest Group for Software Engineering
SKRM	Skriptmodell
SLAM	Simulation Language for Alternative Modeling
SOM	Semantisches Objektmodell
SOMA	Semantic Object Modeling Approach
t_d	Zeitraum
TH	Technische Hochschule
t_p	Zeitpunkt
TQM	Total Quality Management
UML	Unified Modeling Language
USDP	Unified Software Development Process
VDM	Vienna Development Method
VES	Vorgangs-Ereignis-Schema
Vgl.	vergleiche
VOCA	Visions, Objectives and Conferences Association
WAM	Werkzeug, Automat und Material
WfMS	Workflow Management System
WiSt	Wirtschaftswissenschaftliches Studium
WiSu	Das Wirtschaftsstudium
ZfB	Zeitschrift für Betriebswirtschaft
zfbf	Zeitschrift für betriebswirtschaftliche Forschung
zgl.	zugleich

1 Einleitung

„Moderne Informationstechnologie, die dem neusten Stand der Technik entspricht, ist ein wesentlicher Träger jedes Reengineering-Prozesses, da sie den Unternehmen ermöglicht, ihre Unternehmensprozesse neu zu gestalten."
[Hammer 1994: S. 112]

1.1 Gegenstand der Wirtschaftsinformatik

Der Gegenstand der Wirtschaftsinformatik sind betriebliche Informationssysteme in Wirtschaft und Verwaltung. Der Bereich Wirtschaft und Verwaltung erstreckt sich auf Industrie und Handel, Banken und Versicherungen, Dienstleistungsunternehmen sowie öffentliche Betriebe und öffentliche Verwaltungen. Betriebliche Informationssysteme sind dabei nicht isoliert, sondern in ihrem ökonomischen und gesellschaftlichen Kontext zu betrachten. Sie bestehen aus Menschen und Maschinen, die durch Kommunikationsbeziehungen verbunden sind und im Rahmen ihrer Aufgaben Informationen erzeugen, erfassen, übertragen, transformieren, speichern, benutzen oder bereitstellen.[1] *„Ein betriebliches Informationssystem unterstützt die Leistungsprozesse und Austauschbeziehungen innerhalb eines Betriebs sowie zwischen dem Betrieb und seiner Umwelt."*[2] Die computergestützten Informationssysteme (CIS) sind ein zentraler Bestandteil der betrieblichen Informationssysteme.[3] Sie stellen das rechnergestützte bzw. automatisierte Teilsystem des betrieblichen Informationssystems dar. Diese als *Anwendungssysteme*, *Anwendungen* oder *Softwaresysteme* bezeichneten Systeme dienen der Erzeugung, Speicherung, Übertragung,

[1] Vgl. [Ferstl 1998: S. 3, 8 f.], [Balzert 1996: S. 24], [Grochla 1974: S. 24, 25], [Rechenberg 1999: S. 1021].

[2] [Hansen 2001: S. 133]. Andere Definitionen finden sich in [Ferstl 1998: S. 8 f.] oder [Schneider 2000: S. 719].

[3] Die Begriffe betriebliches Informationssystem und betriebliches Anwendungssystem werden häufig als Synonyme verwendet. Wenn der Zusammenhang klar ist, wird auch die verkürzte Form Informations- oder Anwendungssystem verwendet. Der Begriff wird auch dann verwendet, wenn ein solches System in einer öffentlichen Verwaltung, einem Verein, einer Partei oder einer Schule verwendet wird.

Auswertung oder Transformation von Informationen, was in der Organisationslehre als *Informationsprozess*[1] bezeichnet wird.

Die vorliegende Arbeit befasst sich mit der Modellbildung von betrieblichen Informationssystemen zum Zweck der Entwicklung computergestützter Informationssysteme. Dies geschieht zum einen durch die Betrachtung der in einer Organisation ablaufenden Prozesse,[2] zum anderen unter Verwendung eines bestimmten Modellierungsansatzes, der als Objektansatz (OA) bekannt ist.[3] Der Schwerpunkt der Betrachtung liegt auf der im Rahmen der Informationsverarbeitung stattfindenden Kommunikationsbeziehungen.[4]

1.2 Ausrichtung der Organisation an den Geschäftsprozessen

So genannte *Geschäftsprozesse* sind zu einem bestimmenden Faktor für die Unternehmensorganisation geworden. Unabhängig von den Nuancen in der Bedeutung von Begriffen wie *Prozess-* oder *Geschäftsprozessmanagement*, *Business Engineering*, *Business Process Improvement* oder *Business Process (Re-)Engineering* bedeutet *Prozessorientierung*, dass der Betrachtung von Abläufen in Organisationen verstärkte Aufmerksamkeit geschenkt wird[5] und die funktionsorientierte Betrachtung überwunden werden soll.[6] Im Unterschied zur traditionellen deutschen Organisationslehre wird die Ablauforganisation auch nicht als Fortsetzung der Aufbauorganisation[7]

---

[1] Vgl. [Wild 1966: S. 139].

[2] *Prozess* wird im Oxford English Dictionary wie folgt definiert: *„A continuous and regular action or succession of actions, taking place or carried on in a definite manner and leading to the accomplishment of some result; a continuous operation or series of operations."* (Hier zitiert nach [Graham 1997: S. 179].)

[3] Der Objektansatz ist nicht identisch mit dem *Objektprinzip* der Betriebswirtschaftslehre. Auf den Objektansatz wird in Kapitel 2 eingegangen.

[4] Als *Kommunikation* wird nach BUHR, KLAUS und LIEBSCHER der wechselseitige Informationsaustausch zwischen mindestens zwei dynamischen, informationserzeugenden und informationsverarbeitenden Systemen bezeichnet. Vgl. [Klaus 1972: S. 585], [Klaus 1979: S. 312], [Hoyer 1988: S. 16 ff.].

[5] Vgl. [Völkner 1998: S. 1].

[6] Vgl. [Staud 2001: S. 22], [Schmelzer 2002: S. 42 ff.].

[7] Aufbauorganisation ist die Zerlegung der Gesamtaufgabe einer Organisation in Teilaufgaben. Die Teilaufgaben werden zu Stellen zusammengefasst, zwischen denen sich Beziehungszusammenhänge ergeben. Vgl. [Kosiol 1962], [Wöhe 1986].

verstanden.[1] Prozesse werden nicht auf der Ebene der von einzelnen Stellen ausgeführten Abläufe betrachtet, sondern stellen-, abteilungs- und unternehmensübergreifend.[2]

Ein Geschäftsprozess wird als ein Ablauf in einem soziotechnischen System betrachtet. Er ist primär informationsverarbeitend.[3] Er besteht aus Aktivitäten, die von Menschen, von Computern oder von Menschen in Kombination mit Computern ausgeführt werden.[4] Geschäftsprozesse stellen grundsätzlich das Zusammenwirken von menschlichen und sachlichen *Aktionsträgern* dar. Als Ablauf stehen die Aktivitäten mindestens teilweise in Reihenfolgebeziehungen. Die von Maschinen und die von Menschen ausgeführten Aktivitäten befinden sich außerdem in einem gegenseitigen Abhängigkeitsverhältnis.

Dieser Sachverhalt verlangt danach, dass bei der Untersuchung vorhandener und der Konzeption zukünftiger Geschäftsprozesse automatisierte und nicht automatisierte Aktivitäten zusammen betrachtet werden müssen.[5] Diese Arbeit befasst sich mit solchen Geschäftsprozessen, die primär ergebnisorientierte Arbeitsabläufe bei der Benutzung von computergestützten Informations- und Kommunikationssystemen darstellen. Das bedeutet, dass menschliche Aufgabenträger computergestützte Informations- und Kommunikationssysteme benutzen, um ihre Aufgaben zu erfüllen.[6]

[1] Vgl. [Kosiol 1962: S. 189], [Gaitanides 1983: S. 17], [Gaitanides 1994b: S. 5], [Schulte-Zurhausen 1999: S. 42], [Krüger 1994: S. 120], [Bleicher 1993: S. 115].

[2] Vgl. [Bullinger 1996], [Österle 1995: S. 20 ff.], [Schmelzer 2002: S. 34 f.]. Auf die Vielzahl der unterschiedlichen Begriffe im Umfeld der Prozessorientierung wird hier nicht eingegangen, insbesondere weil es sich häufig nur um Begriffsnuancen handelt. In der Dissertation von RAUSCHECKER findet sich eine knapp gehaltene Gegenüberstellung von 16 bekannten Ansätzen. Vgl. [Rauschecker 2000: S. 36 - 45].

[3] Hierauf wird im Verlauf der Arbeit noch eingegangen.

[4] Daraus ergibt sich, dass auch ein rein manuell ausgeführter Prozess einen Geschäftsprozess darstellen kann.

[5] Schon KLING und SCACCHI haben darauf hingewiesen, dass Computersysteme nicht isoliert, sondern nur als Teil eines komplexen Netzes betrachtet werden sollen, zu dem auch ökonomische, politische und soziale Beziehungen gehören. Vgl. [Kling 1980], [Kling 1982]. In der Betriebswirtschaftslehre hat sich ebenfalls die Ansicht durchgesetzt, dass es nicht ausreichend ist, bestehende Strukturen und Prozesse zu „elektrifizieren". Auch die Auswirkungen auf die fachlichen Aufgaben und bestehende Organisationsstrukturen sind zu untersuchen. Vgl. [Picot 1994: S. 111].

[6] Ähnlich ist die Definition von STAUD: „*Ein Geschäftsprozess besteht aus einer zusammenhängenden abgeschlossenen Folge von Tätigkeiten, die zur Erfüllung einer betrieblichen Aufgabe notwendig sind. Die Tätigkeiten werden von Aufgabenträgern in organisatorischen Einheiten unter Nutzung der benötigten Produktionsfaktoren geleistet. Unterstützt wird die Abwicklung der Geschäftsprozesse durch das computergestützte Informationssystem (CIS) des Unternehmens.*" [Staud 1999: S. 6].

1.3 Modellierung von Geschäftsprozessen als Bestandteil der Softwareentwicklung

Als ein wesentlicher Faktor für die erfolgreiche Ausrichtung eines Unternehmens auf die Geschäftsprozesse wird der Einsatz der Informationstechnologie gesehen, womit hier die Gesamtheit aller technischen bzw. computergestützten betrieblichen Informations- und Kommunikationssysteme[1] (IKS) bezeichnet wird.[2] Das Ziel ist, durch betriebliche Informationssysteme die Abläufe einer Organisation zu unterstützen. Um dieses Ziel zu erreichen, werden die dazu notwendigen Softwaresysteme in einem Prozess entwickelt, der im *Software Engineering* als *Systementwicklung* oder *Softwareentwicklung* bezeichnet wird. Das Software Engineering versteht sich als das Teilgebiet der praktischen und angewandten Informatik, welches seit dem Ende der 60er-Jahre um eine ingenieurgemäße[3] Softwareentwicklung bemüht ist.[4] Gegenstand der Softwareentwicklung ist die Abbildung von zu automatisierenden Aufgaben eines Informationssystems auf Rechner- und Kommunikationssystemen.[5] Die Entwicklung eines Softwaresystems ist ein aus mehreren Phasen bestehender Prozess, der mit dem Feststellen der notwendigen Leistungen eines zu erstellenden Softwaresystems beginnt und mit dem Einsatzbeginn dieses Systems endet. Die einzelnen Phasen der Softwareentwicklung lassen sich als Modellerstellungsprozesse auffassen.[6] Auch wenn sich die Bezeichnungen der Phasen in der Literatur unterscheiden, sind die Inhalte – abgesehen von Nuancen – im Wesentlichen gleich:[7]

---

[1] Der in der Literatur häufig verwendete Begriff Informations- und Kommunikationssystem (IKS) wird hier synonym zu CIS verwendet.

[2] Dies kann als *Credo der Prozessorientierung* betrachtet werden. Vgl. [Davenport 1993b], [Hammer 1993], [Scott-Morton 1991], [Huber 1997]. Bei der Lektüre der entsprechenden Veröffentlichungen könnte man den Eindruck erhalten, dass sich organisatorische Probleme durch den Einsatz von IKS lösen ließen. Diese Ansicht wird vom Verfasser dieser Arbeit nicht geteilt. Vgl. auch [Schmelzer 2002: S. 22].

[3] Als *ingenieurgemäß* werden Tätigkeiten angesehen, wenn sie planbar, wiederholbar, überprüfbar, methodisch unterstützt Produkte hervorbringen, die festgesetzten Qualitätsansprüchen genügen. Siehe auch die unterschiedlichen Begriffsdefinitionen des Begriffs in [Balzert 1996: S. 35].

[4] Als Beginn des Software Engineerings wird üblicherweise die erste NATO-Konferenz über Software Engineering angesehen. Vgl. [Naur 1968].

[5] [Ferstl 1998: S. 7].

[6] Vgl. [Jacobson 1994], [Züllighoven 1998: S. 142], [Ferstl 1998]. Als Modell wird nach FERSTL die zielorientierte Abbildung eines Systems durch ein anderes System angesehen. Vgl. [Ferstl 1998: S. 18]. Auf den Modellbegriff wird im folgenden Kapitel noch detailliert eingegangen.

[7] Vgl. auch Abschnitt 2.6.

- Die Anforderungen an ein zu erstellendes System werden untersucht und dokumentiert. Dabei werden die Anforderungen in ein Modell umgewandelt. Dies wird als *Anforderungsanalyse*, *Systemanalyse* oder *Requirements Engineering* bezeichnet.
- Unter Beachtung technischer Randbedingungen wird ein Modell der softwaretechnischen Umsetzung der Anforderungen entworfen. Dies wird als *Entwurf* oder *Design* bezeichnet.
- Das System wird unter Verwendung von Programmiersprachen, Datenbankverwaltungssystemen, Halbfabrikaten[1] usw. realisiert. Dies wird als *Implementierung*, *Realisierung* oder *Programmierung* bezeichnet.
- Es wird geprüft, ob das System den Anforderungen entspricht. Dies wird als *Test* bezeichnet. Nach dem erfolgreichen Abschluss der Testphase wird das System eingesetzt.
- Die Modifikation eines Softwaresystems in der Einsatzphase bezeichnet man als *Wartung*.

Da die Unterstützung von Geschäftsprozessen eine Anforderung an betriebliche Informationssysteme ist, wird die Untersuchung und Modellierung der Geschäftsprozesse zu einem Betrachtungsgegenstand der Anforderungsanalyse.[2] In der Anforderungsanalyse für betriebliche Informationssysteme sind das Unternehmen und die in ihm existierenden Geschäftsprozesse zu untersuchen.[3] Zum einen sind die Geschäftsprozesse aus betriebswirtschaftlicher Perspektive als Abläufe in einem soziotechnischen System zu betrachten. Zum anderen ist im Rahmen der informationstechnischen Perspektive zu untersuchen, welche Aspekte des Geschäftsprozesses als Anforderungen bei der Entwicklung von Informationssystemen zu berücksichtigen sind. Prozessmodellierung unterscheidet sich aber von anderen Modellierungstypen der Informatik durch den Umstand, dass viele der modellierten Phänomene nicht von Maschinen in Gang gesetzt werden, sondern von Menschen, schreiben CURTIS, KELLNER und OVER.[4] Die Geschäftsprozessmodellierung (GPM) darf also auf der einen Seite nicht auf die Abbildung der Abläufe in den

[1] Als *Halbfabrikate* werden hier Softwarebibliotheken, Komponenten, Rahmenwerke (*Frameworks*), Generatoren etc. bezeichnet, wie sie bei der Entwicklung von Softwaresystemen heute üblich sind. Diese Ansätze zur Wiederverwendung werden ausführlich in [Linssen 1994] dargestellt.

[2] Dies lässt sich dadurch belegen, dass sich die Fachgruppe *Modellierung betrieblicher Informationssysteme* (MobIS) der Gesellschaft für Informatik (früher Fachgruppe 5.2, heute 5.10) intensiv mit der Modellierung von Geschäftsprozessen auseinandersetzt. Vgl. [MobIS 1996], [MobIS 1997], [MobIS 1998], [MobIS 1999], [MobIS 2000].

[3] Ähnliche Positionen finden sich in [Oestereich 1998], [Jacobson 1994], [Jacobson 1999], [Kruchten 1998], [Taylor 1995], [Leffingwell 2000], [Yourdon 1996: S. 71 ff.], [Balzert 1998], [Sommerville 1997: S. 41 f.], [Lehmann 2000: S. 47]. ÖSTERLE geht so weit, die Betrachtung der Prozesse als integrativen Faktor zwischen der Unternehmensstrategie und der Systementwicklung zu bezeichnen. Vgl. [Österle 1995: S. 21].

[4] [Curtis 1992: S. 75].

Softwaresystemen beschränkt sein, sollte aber auf der anderen Seite eine Form besitzen, die die Verwendung der Ergebnisse im weiteren Verlauf der Softwareentwicklung ermöglicht. Dies ist oft problematisch, da die im Rahmen der betriebswirtschaftlichen Prozessmodellierung entstehenden Modelle weder die Form noch die notwendige Präzision oder Detailliertheit besitzen, die für die Softwareentwicklung notwendig ist.[1] Dieses Problem wird hier als *semantische Lücke* bezeichnet.

Um dieses Problem zu lösen, bieten sich für die Modellierung der Geschäftsprozesse unterschiedliche Ansätze an. Für die Untersuchung der Anforderungen aus betriebswirtschaftlicher Perspektive bieten sich Techniken der Organisationslehre an. Für die softwaretechnische Perspektive bieten sich die Ansätze aus dem Software Engineering an. Eine Zwischenstellung nehmen die Ansätze aus dem Bereich der Wirtschaftsinformatik ein, die Techniken aus der Organisationslehre und aus dem Software Engineering kombinieren. Hier seien exemplarisch ARIS, SOM und PROMET als Ansätze genannt.[2]

Bei der GPM werden sowohl klassische Techniken aus dem Bereich der Organisationslehre, als auch neu entwickelte Konzepte eingesetzt. Am weitesten verbreitet dürften die Ereignisgesteuerten Prozessketten (EPK) der Methode ARIS sein.[3] Darüber hinaus werden so genannte Vierdimensionale Prozessdarstellungen[4], *line of visibility charts*[5], Entity-Prozess-Matrizen[6], IDEF0- und IDEF3-Diagramme[7], Datenflusspläne[8], Wertketten[9], Darstellungen des Dokumentenflusses[10],

---

[1] Vgl. [Chonoles 1995b].

[2] Vgl. [Scheer 1998], [Scheer 1998b], [Österle 1995], [Ferstl 1998], [Rechenberg 1999]. Der Autor hat die Methode ARIS in einem Praxisprojekt zur Analyse eines umfangreichen Geschäftsprozesses (Projektabwicklung) eines mittelständigen Systemhauses eingesetzt. Die Erfahrungen dieses Praxisprojekts stellten einen wichtigen Input für den hier entwickelten Modellierungsansatz dar.

[3] Vgl. [Scheer 1995], [Scheer 1998], [Scheer 1998b], [Staud 1999], [Keller 1998], [Rittgen 2000].

[4] Vgl. [Scholz 1994b].

[5] Vgl. [IBM 1993].

[6] Vgl. [Raster 1994], [Scholz 1994].

[7] Vgl. [IDEF0 1993], [IDEF3 1995], [Mayer], [Mayer 1995], [Coulson-Thomas 1994], [AndrewsD 1994].

[8] Vgl. [Steinbuch 1998].

[9] Vgl. [Porter 1989: S. 62 ff.], [Schüll 1999], [Wenzel 1997].

[10] Vgl. [Raufer 1997].

Eingabe-Verarbeitung-Ausgabe-Tabellen[1], spezielle Symbolsprachen[2], Z-Schemata[3], Flussdiagramme[4] oder Petri-Netze[5] verwendet.

Bekannte Methoden des Geschäftsprozessmanagements wie BPI, ARIS, PROMET oder die Methode von STRIENING verwenden klassische Formalismen der Datenverarbeitung wie Flussdiagramme, Entity-Relationship-Diagramme oder Funktionsbäume.[6] Die Ergebnisse der Modellierung passen nicht zu den heute verwendeten Methoden der Softwareentwicklung und müssen mit großem Aufwand und mit der Gefahr von Umsetzungsfehlern in adäquate Modelle umgeformt werden.

Im Rahmen einer geplanten Softwareentwicklung erscheint es demzufolge nicht sinnvoll, die Geschäftsprozesse mit anderen Techniken abzubilden als die bei der Softwareentwicklung verwendeten Techniken,[7] da sonst ein methodischer Bruch und jene – schon erwähnte – semantische Lücke entsteht.

Die Geschäftsprozesse werden – in der heutigen Praxis – mit Methoden und Modellen der Betriebswirtschaft untersucht und anschließend werden die Anforderungen an die dafür notwendigen Anwendungssysteme mit den Methoden und Modellen der Softwareentwicklung aufgestellt.

Die dabei entstehenden Friktionen zeigen sich regelmäßig auch in dem Phänomen, dass betriebliche Informationssysteme nicht den Anforderungen der betriebswirtschaftlichen Seite entsprechen. Durch den methodischen Bruch zwischen der betriebswirtschaftlichen und der softwaretechnischen Sicht entstehen häufig Missverständnisse und Informationsverluste. Wesentlich sinnvoller wäre es, beide Perspektiven – die im Unternehmen stattfindenden Geschäftsprozesse und die Anforderungen an ein Softwaresystem – mit einem Ansatz zu modellieren.[8] Dafür soll hier eine aus dem Software Engineering stammende Sichtweise verwendet werden.

---

[1] Vgl. [Striening 1988]. Dieser Ansatz basiert auf der von der Firma IBM entwickelten Methode HIPO (Hierarchie + Input-Process-Output). Vgl. [IBM 1979].

[2] Vgl. [Eversheim 1995], [Mielke 2002].

[3] Vgl. [Ginbayashi 1992].

[4] Vgl. [Gaitanides 1983], [Harrington 1991], [Striening 1988], [Coulson-Thomas 1994], [Klepzig 1997: S. 75].

[5] Vgl. [Oberweis 1996], [Berztiss 1996].

[6] Vgl. [Harrington 1991], [Striening 1988], [Österle 1995], [Scheer 1998b].

[7] Vgl. [Frank 1997b: S. 5].

[8] Vgl. [Taylor 1995: S. 9], [Hubert 2002: S. 58 f., 99 ff.].

1.4 Wahl des Objektansatzes

Im Software Engineering wurde im Laufe der letzten 30 Jahre eine Vielzahl von Formalismen und Methoden entwickelt.[1] Zu den *Formalismen* zählen Grammatiken, Syntaxgraphen, logische Formalismen, Funktionsbäume, aber beispielsweise auch Entity-Relationship-Modelle, Zustandsübergangsdiagramme oder Petri-Netze. Durch *Methoden* kommen diese Formalismen in unterschiedlicher Weise und mit unterschiedlichen Schwerpunkten zur Anwendung. Bei der Vielzahl der bis heute entwickelten Methoden lassen sich zwei Ausrichtungen feststellen, die auf spezifischen Sichtweisen basieren:

- Die *strukturierten Ansätze* gehen von der Abbildung des Datenflusses aus. Methodisches Prinzip ist eine Top-Down-Zerlegung der stattfindenden Prozesse mit schrittweiser Verfeinerung. Die Prozesse verarbeiten eingehende Datenströme, die sie aus Datenspeichern erhalten und produzieren ausgehende Datenströme, die sie wiederum in Datenspeicher schreiben. Die Struktur der in einem System verwendeten Daten wird durch Entity-Relationship-Diagramme dargestellt. Die Datenmodellierung erfolgt getrennt von der Funktionsmodellierung. In den einzelnen Phasen (Anforderungsanalyse, Entwurf, Implementierung) der Systementwicklung werden unterschiedliche Diagramme und Modelle verwendet. Wesentlich für den strukturierten Ansatz ist die Dichotomie der Daten- und Funktionsmodellierung.[2]
- Die *objektorientierten Ansätze* gehen von der Abbildung der in der Realität vorzufindenden Objekte und den zwischen diesen bestehenden Beziehungen aus. Im Gegensatz zum strukturierten gründet der objektorientierte Ansatz gerade auf der Integration von Daten und Funktionen zu einer Einheit, den Objekten.[3] Prozesse werden durch Abfolgen von Objektinteraktionen abgebildet. Objekte führen im Rahmen dieser *Interaktionen*[4] Aktivitäten aus. Sie stellen definierte Schnittstellen zur Verfügung, die von anderen Objekten verwendet werden können. Als objektorientiert gilt ein Ansatz, wenn er die Konzepte Kapselung, Polymorphie, Vererbung, das Nachrichtenprinzip und die Dichotomie von Klasse und Objekt enthält.[5] Objektorientierte Ansätze trennen nicht die Daten- von der Funktionsmodellierung. In allen Phasen der Systementwicklung können die gleichen Diagramme und Modelle verwendet werden.

---

[1] Übersichten über gebräuchliche Methoden und Formalismen im Software Engineering finden sich in [Schönthaler 1992], [Balzert 1996], [Frick 1995], [Partsch 1998], [V-Modell 1997].

[2] Vgl. [Schienmann 1997: S. 31].

[3] Vgl. [Schienmann 1997: S. 31].

[4] Zum Begriff *Interaktion* siehe Abschnitt 4.3.5.1 und das Glossar.

[5] Der objektorientierte Ansatz wird in Kapitel 2 noch umfassend dargestellt.

Folgende Überlegungen führten dazu, einen objektorientierten Ansatz zur Abbildung der Geschäftsprozesse zu verwenden:

- Der objektorientierte Ansatz setzt sich in der industriellen Praxis als herrschendes Entwicklungsparadigma durch.[1] Bei Neuentwicklungen von Softwaresystemen in Projekten, die eine Entwicklungsmethode verwenden, ist heute die Verwendung des strukturierten Ansatzes zur Ausnahme geworden.[2]

- Es wird allgemein akzeptiert, dass bei fachgerechter Verwendung des objektorientierten Ansatzes im Bezug auf die Aspekte Offenheit, Wartbarkeit, Wiederverwendbarkeit, Flexibilität, Integrationsfähigkeit bessere Softwaresysteme entstehen.

- Die objektorientierte Sichtweise ähnelt der in der Betriebswirtschaft verbreiteten Sichtweise, nach der Systeme durch ihre Elemente, durch die Beziehungen zwischen den Elementen und im Fall offener Systeme durch ihre Beziehungen zu ihrer Umwelt charakterisiert werden.[3] Die Beziehungen beschreiben entweder den Aufbau oder den Ablauf.[4] *Organisationen* sind komplexe, zielgerichtete, offene, dynamische, soziotechnische Systeme, die Informationen gewinnen und verarbeiten.[5] Sie sind komplex, weil sie ein Gefüge verschiedenartiger Beziehungen darstellen.[6] Sie sind zielgerichtet, weil bestimmte Ziele in Form zukünftiger Zustände angestrebt werden. Sie sind offen, weil sie mit ihrer Umwelt in Austauschbeziehungen stehen.[7] Sie sind dynamisch, weil die Systemelemente durch ihr Zusammenwirken Aktivitäten zeigen, durch die sich der Zustand des Systems im Zeitverlauf ändert.[8] Es handelt sich um sozio-

[1] Vgl. [Kappel 1996: S. VIII], [Quibeldey-Cirkel 1994].

[2] Hier existieren zwei Ausnahmen: 1) Bei Wartungsprojekten im Großrechnerumfeld werden auch heute noch strukturierte Ansätze verwendet. 2) Bei technischen Steuersystemen wird auf den Einsatz objektorientierter Technologien verzichtet, um Speicherplatz einzusparen und eine höhere Performanz zu erreichen. Deshalb werden teilweise noch Assemblersprachen oder mindestens sehr maschinennahe Sprachen wie C oder PERL (vgl. [Pratt 1997], [Louden 1994]) verwendet. Hinzu kommen als weitere Möglichkeit außerdem noch die Projekte, die überhaupt keinen methodischen Ansatz (als *ad hoc* Vorgehen bezeichnet) verwenden.

[3] [Baetge 1974: S. 11], [Schmidt 1988: S. 19], [Schulte-Zurhausen 1995: S. 27]. Diese Sichtweise, dass Systeme ein Netz von Beziehungen darstellen, findet sich bereits in der Kybernetik. Vgl. [Cube 1970: S. 161].

[4] Vgl. [Schmidt 1988: S. 21].

[5] Vgl. [Heinen 1985: S. 51, 53, 62, 72], [Kappler 1991: S. 76], [Bleicher 1991: S. 35], [Ferstl 1998: S. 59, 65], [Striening 1988: S. 6, 150], [Probst 1992: S. 27], [Schulte-Zurhausen 1999: S. 1 f., 36], [Hill 1994: S. 17 – 26], [Völkner 1998: S. 17], [Wild 1966: S. 30].

[6] Vgl. [Hill 1994: S. 22], [Heinen 1985: S. 53], [Kosiol 1962: S. 76].

[7] Vgl. [Heinen 1985: S. 72].

[8] Vgl. [Hill 1994: S. 23]. Nach MILLER wird dies als *lebendes System* bezeichnet. Vgl. [Miller 1978].

technische Systeme, weil mehrere Personen arbeitsteilig zusammenarbeiten und zur Erreichung der Ziele Aufgaben erfüllen und dabei Werkzeuge verwenden.[1]

Aus diesen Gründen wird für die Abbildung von Geschäftsprozessen eine objektorientierte Sichtweise verwendet und eine *objektorientierte Geschäftsprozessmodellierung*[2] (ooGPM) angestrebt. Es wird also im weiteren Verlauf der Arbeit davon ausgegangen, dass ein Unternehmen ein System von Objekten darstellt, die untereinander in Beziehungen stehen und durch Nachrichtenaustausch miteinander und mit ihrer Umwelt interagieren.[3] Dies wird in Anlehnung an die objektorientierten Ansätze des Software Engineerings als *objektorientiertes Modell der Unternehmung* bezeichnet.[4]

Prozesse bringen das „*im Zeitablauf sich vollziehende Verhalten der Organisation bzw. der in ihr tätigen Menschen zum Ausdruck.*“[5] „*Prozesse [...] vollziehen sich im Rahmen ihrer verhältnismäßig unveränderlichen Grundstruktur.*“[6] Ein Prozess wird als eine inhaltlich abgeschlossene Folge von logisch zusammenhängenden Aktivitäten verstanden, die nach

---

[1] Dieser Organisationsbegriff umfasst Unternehmen (Betriebswirtschaften), öffentliche Dienstleistungsbetriebe, Verwaltungen, Spitäler, Schulen etc. Vgl. [Bleicher 1991: S. 35], [Hill 1994: S. 17 ff.]. Verwaltungen im Allgemeinen und Behörden im Besonderen, Vereine, Schulen, Kirchen, Militär, Parteien, kriminelle Vereinigungen, Betriebe – ihnen allen ist bei allen Unterschieden im Detail gemeinsam, dass sie auf Dauer angelegt sind und aus Gruppen und Individuen bestehen, die gemeinsam in Arbeitsteilung ein Ziel anstreben. Vgl. [Schulte-Zurhausen 1995: S. 1]. Eine ähnliche Sichtweise findet sich schon bei WEBER, der Gemeinsamkeiten in der Struktur von Militär, Staat und Betrieb sah. Vgl. [Weber 1980: S. 825]; ähnlich auch in [Bleicher 1991: S. 34], [Bleicher 1993], [Wild 1966: S. 42]. Es ist jedoch unterschiedlich, inwieweit sie dem Wirtschaftlichkeitsprinzip folgen, unter alternativen Verwendungsweisen knappe Mittel so einzusetzen, dass das Verhältnis des Mitteleinsatzes und der Bedürfnisbefriedigung einen bestmöglichen Wert annimmt. Vgl. [Zelewski 1994: S. 19 f.]. Siehe auch im Glossar die Definition des Begriffs *Betriebswirtschaft*.

[2] Vgl. [Chonoles 1995: S. 54].

[3] Diese Definition folgt der in der Kybernetik verwendeten, nach der ein *System* aus einer Menge von Elementen besteht, die Eigenschaften haben und zwischen denen Beziehungen existieren. Ein System ist von seiner Umwelt abgrenzbar. Ein Element eines Systems kann selbst ein System sein. Vgl. [Klaus 1979: S. 800, 806, 807].

[4] Ähnlich auch in [Ferstl 1998: S. 38]. Vermutlich war GRAHAM [Graham 1991] die erste größere Veröffentlichung, die die objektorientierte Modellierung von Unternehmen erwähnte. Weitere Veröffentlichungen, die sich mindestens am Rand mit dem Thema befassen, sind [Frank 1994], [Fahrwinkel 1995], [Jacobson 1995], [Ferstl 1994], [Ferstl 1995], [Taylor 1995], [Hubert 2002]. Ein guter Überblick über das Spannungsverhältnis zwischen Objektorientierung und BPR findet sich in [Fowler 2000].

[5] [Heinen 1985: S. 53].

[6] [Heinen 1985: S. 62].

bestimmten Regeln durchgeführt wird.[1] Diese – zum Teil geordnete – Menge von Prozessschritten dient in ihrer Gesamtheit der Erreichung eines Systemziels. Durch Prozesse werden Leistungen erstellt oder Objekte verändert.[2] *Prozesse* stellen in der hier verwendeten objektorientierten Sichtweise Abfolgen von Objektinteraktionen dar, durch die die beteiligten Objekte Materie, Energie und Informationen[3] austauschen und Verhalten zeigen.[4] *Geschäftsprozesse* sind dadurch gekennzeichnet, dass im Rahmen der Interaktionsbeziehungen hauptsächlich Informationen ausgetauscht werden.[5] Diese besondere Bedeutung der Interaktionsbeziehung für die Modellierung von Prozessen im objektorientierten Ansatz wird im weiteren Verlauf als das *Primat der Interaktionsbeziehung* bezeichnet.

1.5 Evaluierung objektorientierter Modellierungsmethoden

Objektorientierte Modellierungsmethoden basieren auf Konzepten der konzeptuellen und semantischen Datenmodellierung, Semantischen Netzen, Zustandsübergangsdiagrammen und der objektorientierten Programmierung. Seriöse Arbeiten gehen von mindestens 50 unterschiedlichen Ansätzen aus, wobei die tatsächliche Anzahl aufgrund nicht allgemein zugänglicher Arbeiten aus Forschung und Praxis erheblich höher sein dürfte.[6] Als Ergebnis konvergierender Entwicklungen entstanden zwei Notationen, die aktuell als die am weitesten entwickelten angesehen werden:

---

[1] [Schulte-Zurhausen 1995: S. 41].

[2] [Schulte-Zurhausen 1999: S. 49].

[3] Unter *Information* wird zweckorientiertes, auf den Vollzug von Aktionen zur Erreichung von Zielen bezogenes Wissen verstanden. Vgl. [Wild 1966: S. 97], [Wild 1970: S. 52], [Heinen 1985: S. 62].

[4] VÖLKNER bezeichnet Geschäftsprozesse als Folge von Zuständen. Vgl. [Völkner 1998: S. 28]. Bei einer objektorientierten Sichtweise ist jedoch der Nachrichtenaustausch durch Interaktion maßgebend. Die Zustandsveränderungen sind demgegenüber nachrangig, da sie sich immer nur auf einzelne Objekte beziehen. An Prozessen sind aber in der Regel immer mehrere Objekte beteiligt.

[5] Daraus ergibt sich, dass es sich bei Geschäftsprozessen in der Regel um Informationsprozesse (vgl. das Stichwort *Informationsprozess* auf S. 2) handelt.

[6] Folgende Ansätze gehören zu den bekannteren und wurden vom Verfasser gesichtet: Object Oriented Analysis (OOA) von COAD und YOURDDON [Coad 1991], IDEF4 [IDEF4 1995b], Object Modeling Technique (OMT) von RUMBAUGH, BLAHA und PREMERLANI [Rumbaugh 1993], Object Oriented Analysis and Design (OOAD) von BOOCH [Booch 1995], Object Oriented Software Engineering (OOSE) von JACOBSON [Jacobson 1994], FUSION von COLEMAN [Coleman 1994], SYNTROPY von COOK und DANIELS [Cook 1994], das Semantische Objektmodell (SOM) von FERSTL und SINZ [Ferstl 1998], Business Object Notation (BON) von NERSON und WALDÉN [Waldén 1995], Object Oriented Systems Analysis (OOSA) von SHLAER und MELLOR [Shlaer 1988], Object Oriented Analysis & Design (OOA&D) von MARTIN und ODELL [Martin 1992], Object Oriented Diagram Technique (OODT) von KAPPEL [Kappel

- Die *OPEN Modeling Language* (OML) von FIRESMITH, HENDERSON-SELLERS, GRAHAM et. al.[1]
- Die *Unified Modeling Language* (UML) der Object Management Group (OMG).[2]

Beide Sprachen sind ähnlich mächtig und umfangreich dokumentiert. Die UML wird als Grundlage dieser Arbeit verwendet, weil sie eine breitere Akzeptanz in Wissenschaft und Praxis findet.[3] Sie stellt heute den Industriestandard zur objektorientierten Modellierung dar.[4] Das bedeutet für diese Arbeit im Einzelnen:

- Zum einen wird der hier entwickelte Modellierungsansatz selbst mit Hilfe der UML beschrieben. Dies geschieht durch die Definition von Metamodellen, welche die Form statischer Strukturdiagramme (so genannte Klassendiagramme) der UML besitzen.
- Zum anderen stellt der hier entwickelte Modellierungsansatz eine Erweiterung der UML zur Beschreibung von Geschäftsprozessen dar. Das bedeutet, dass die Konstrukte der UML verwendet und durch zusätzliche Konstrukte erweitert werden.

1.6 Ziele der Arbeit

Die Arbeit befasst sich mit der Konzeption eines Modellierungsansatzes und gehört damit sowohl zu einem Teilbereich des Software Engineerings wie auch zur Wirtschaftsinformatik. Das Ziel ist

---

1996], Object-Oriented Systems Analysis (OSA) von EMBLEY, KURTZ und WOODFIELD [Embley 1992], Responsibility-Driven-Design (RDD) von WIRFS-BROCK, WILKERSON und WIENER [Wirfs-Brock 1993], [Wilkinson 1995], der Werkzeug-und-Material-Ansatz (WAM) von ZÜLLIGHOVEN [Züllighoven 1998], und der Ansatz des OOTC der Firma IBM [IBM 1997]. In der Literatur finden sich umfangreiche Evaluierungen dieser Methoden und Notationen. Vgl. [Stein 1997], [Hutt 1994], [Frank 1997d], [Frank 1997e]. Aus diesem Grund wird hier darauf verzichtet, die Methoden darzustellen oder zu vergleichen.

[1] Vgl. [Firesmith 1997], [Henderson-Sellers 1998], [Graham 1997], [Henderson-Sellers 2000]. OPEN steht für Object-Oriented Process, Environment and Notation.

[2] Die UML geht im Wesentlichen auf RUMBAUGH, BOOCH und JACOBSON zurück und ist seit November 1997 von der OMG standardisiert. Vgl. [Booch1999], [Rumbaugh 1999], [Jacobson 1999], [OMG 1999], [OMG 2001].

[3] Ein Vergleich von OML und UML findet sich in [Frank 1997d], [Frank 1998d].

[4] Vgl. [Kobryn 1999]. Dieser Industriestandard stammt von der OBJECT MANAGEMENT GROUP (OMG), einer Vereinigung, der eine Vielzahl bekannter Softwarefirmen (u. a. Hewlett-Packard, IBM, Microsoft, Oracle, Rational, SAP) angehören. Vgl. [OMG 1999].

die Entwicklung eines semiformalen[1] objektorientierten Modellierungsansatzes, der im Rahmen der objektorientierten Softwareentwicklung (ooSe) zur GPM verwendet wird. Die Arbeit verfolgt einen interdisziplinären Ansatz, der sowohl den aktuellen Stand der Forschung im Bereich der angewandten Organisationslehre (Geschäftsprozessmanagement) wie auch den der angewandten Informatik (objektorientierte Softwareentwicklung) berücksichtigt.

Die Geschäftsprozesse werden unabhängig davon beschrieben, ob und wie sie durch ein CIS automatisiert werden. Auf diese Weise soll die GPM in das methodische Vorgehen zur Entwicklung von CIS integriert werden. Im Gegensatz zur UML basiert der Modellierungsansatz auf dem Konzept der *Skripte*, einer tabellarischen Darstellungsform,[2] die primär zur Abbildung von Objektinteraktionen verwendet wird.

Wie wird nun die Entwicklung eines weiteren Ansatzes gerechtfertigt, wenn schon eine ganze Reihe von objektorientierten Ansätzen und auch eine Vielzahl von Techniken zur GPM existieren? Folgende Überlegungen und Beobachtungen führten zur Entwicklung des hier dargestellten Ansatzes: Es wird in der Regel ohne weitere Prüfung postuliert, dass der OA auch zur GPM geeignet ist. Die objektorientierten Ansätze sind aber zur Modellierung von Softwaresystemen, nicht jedoch zur Modellierung von Geschäftsprozessen entwickelt worden. Es handelt sich in erster

[1] Als *semiformaler Ansatz* wird nach FENSEL ein Ansatz bezeichnet, der eine definierte Syntax und eine Menge vordefinierter Primitive verwendet. Die Verwendung natürlicher Sprache ist nur in eingeschränkter Weise zugelassen. Semiformale Ansätze verfügen über keine definierte Semantik. Vgl. [Fensel 1994: S. 4]. Formale Ansätze sind Spezifikationssprachen wie VDM oder Z. Diese verfügen über eine definierte Semantik und basieren in der Regel auf mathematischen und logischen Kalkülen. Vgl. [Jones 1990], [Wordsworth 1992], [Andrews 1991], [Andrews 1997], [Lano 1995], [Sheppard 1994], [Turner 1994]. Die Verwendung formaler und semiformaler Ansätze in der Softwareentwicklung wird in der Praxis sehr kritisch betrachtet, da sie stark mit Vorurteilen behaftet ist. Ähnliche Vorbehalte sind auch im Bereich der Geschäftsprozessmodellierung zu erwarten. In [Meyer 1985] und [Hall 1990] werden diese Vorurteile untersucht und widerlegt. Die dort genannten Argumente können hier direkt übernommen werden.

[2] Die Wurzeln des Konzepts *Skript* finden sich in der Kognitionspsychologie und im Bereich der Wissensrepräsentation, wo man sich mit der Untersuchung und Repräsentation menschlicher Wissens- und Denkstrukturen auseinandersetzt. Vgl. [Gerstenmeier 1995], [Reimer 1991: S. 209], [Bobrow 1975], [Mandler 1984], [Helbig 1996: S. 124 ff.], [Smith 1981], [Schank 1972], [Schank 1975], [Schank 1977], [Schank 1981]. Der Begriff wird dort als Schema für bestimmte Ereignisabfolgen verwendet. Vgl. [Banyard 1995]. In der Informatik versteht man unter Skript die prototyphaft beschriebene Folge von Ereignissen und Aktionen in einem bestimmten Kontext. Vgl. [Rechenberg 1999: S. 986]. In der Methode OBJECT BEHAVIOR ANALYSIS schließlich wurden Tabellen als Repräsentationsform für Skripte entwickelt. Vgl. [Gibson 1990], [Rubin 1992], [Rubin 1994b]. Diese tabellarische Struktur wird als Ausgangspunkt des hier entwickelten Ansatzes verwendet und weiterentwickelt.

Linie um ein Maschinenmodell.[1] Für die Aufgaben der GPM fehlen häufig entsprechende Konstrukte. Viele Sachverhalte lassen sich deshalb nur mit großer Mühe dokumentieren. Ein weiteres Problem wird darin gesehen, dass die meisten Ansätze komplexe graphische Notationen verwenden. Dadurch ist die Verwendung entsprechender automatisierter Werkzeuge nahezu zwingend erforderlich. Ebenfalls als Problem wird die Verwendung unterschiedlicher Sichten angesehen. Dabei wird in der Regel die Struktur getrennt von den innerhalb dieser Struktur ablaufenden Prozessen abgebildet. So sinnvoll diese Trennung zur Komplexitätsbewältigung und zur Bildung isolierender Abstraktionen auch ist, erweist sie sich gerade in den frühen Phasen der Modellierung häufig als hinderlich, da sie die parallele Erstellung und Pflege mehrerer Modelle bzw. Diagramme erfordert. Dies erschwert die praktische Arbeit, weil diese Sichten ständig abgeglichen werden müssen.

Vor diesem Hintergrund wird ein Ansatz entwickelt, der folgenden Ansprüchen genügt: Der Ansatz basiert auf den Prinzipien der Objektorientierung, weil diese in der Softwareentwicklung maßgeblich sind. Er beinhaltet zusätzliche Konstrukte, die bei der GPM benötigt werden. Dabei wird überprüft, welche Unterschiede bei einer Geschäftsprozessmodellierung, die auf der Grundlage des Objektansatzes vorgenommen wird, zu beachten sind. Prozesse werden in dem zu entwickelnden Ansatz als Abfolgen von Objektinteraktionen interpretiert.[2] Es wird mit den Skripten keine graphische Notation verwendet, sondern eine tabellarische Darstellung. Diese Form wird gewählt, weil sie eine kompaktere Darstellung von Prozessen ermöglicht, als dies mit den bisherigen Ausdrucksmitteln des Objektansatzes möglich ist. Die Skripte sollen die simultane Modellierung der Prozesse und der Struktur ermöglichen, wobei der Schwerpunkt der Betrachtung auf den Prozessen liegt. Der Ansatz muss schließlich kompatibel zur UML sein, weil die UML in Forschung und Praxis weitestgehend als Standard akzeptiert ist.

Aus dieser Zielsetzung ergeben sich für die Arbeit die folgenden Aufgaben:

1. Es wird geklärt, was unter einem Modell verstanden wird und welche Bestandteile der Modellierungsansatz umfassen wird.
2. Der Modellierungsansatz basiert auf dem im Software Engineering verwendeten OA. Da die Terminologie im Schrifttum nicht einheitlich ist, wird – auf der Grundlage des verwendeten Quellenmaterials – ein Objektansatz entwickelt.
3. Die UML als standardisierte Repräsentationsform des Objektansatzes wird beschrieben, weil sie im Folgenden zur Spezifikation des Modellierungsansatzes dient.

---

[1] Vgl. Abschnitt 2.2.

[2] Dies wurde als das *Primat der Interaktionsbeziehung* bezeichnet.

4. Um Verbesserungspotentiale aufzuzeigen, werden die Möglichkeiten existierender objektorientierter Modellierungsansätze zur GPM dargestellt.
5. Als Ausgangspunkt für die eigene Entwicklung wird ein Modellierungsansatz vorgestellt, der als Repräsentationsform Skripte verwendet. Im Weiteren wird auf die in diesem Ansatz enthaltenen Schwächen eingegangen.
6. Es wird für den OA ein Begriffssystem in Form eines Metamodells entwickelt.
7. Für den Aufbau der Skripte wird ein weiteres Metamodell entwickelt.
8. Die äußere Repräsentationsform der Skripte soll weitgehend vom verwendeten Begriffsystem getrennt werden. Hierfür wird eine entsprechende Architektur entwickelt.
9. Durch die Architektur werden die Beziehungen zwischen den Skripten und dem Begriffssystem festgelegt.
10. Es wird gezeigt, dass auch andere Repräsentationsformen (d. h. Diagrammarten) sowohl auf die verwendeten Skripte als auch auf das Begriffssystem abgebildet werden können.
11. Die Anwendung des Modellierungsansatzes wird anhand von Beispielen demonstriert.
12. Die Möglichkeiten der UML werden durch zusätzliche Konstrukte erweitert.
13. Es werden die Phänomene dargestellt, die bei der Modellierung von Geschäftsprozessen zu berücksichtigen sind.
14. Es wird gezeigt, wie diese Phänomene mit Skripten objektorientiert modelliert werden können.
15. Es werden die Unterschiede erarbeitet, die bei einer objektorientierten GPM zu beachten sind.

1.7 Abgrenzung

Das Thema der vorliegenden Arbeit ist die Entwicklung eines objektorientierten Ansatzes zur GPM. Folgende Themen sind nicht Inhalt dieser Arbeit:

- Die Arbeit befasst sich nicht mit der *strategischen Geschäftsprozessanalyse und Soll-konzeption*[1], in deren Rahmen Ziele, kritische Erfolgsfaktoren, Strategien usw. entwickelt werden, welche die Randbedingungen der Prozesse darstellen. Es wird davon ausgegangen, dass das Zielsystem, interne Rahmenbedingungen, kritische Prozesse und Wertketten, die

[1] Vgl. [Scheer 1998b: S. 41]. Dies wird von GEHRING als *Strategieentwicklung* bezeichnet, deren Ergebnis die *Geschäftsfeldstrategie* ist. Vgl. [Gehring 1998: KE2, S. 31]. Zum Thema strategische Unternehmens-führung vergleiche [Staehle 1994: S. 573 ff.].

Kernkompetenzen, Geschäftsfelder, der Gestaltungsbereich und andere strategische Entscheidungen feststehen, bevor die *objektorientierte Geschäftsprozessmodellierung* stattfindet.[1]

- In der Arbeit wird die ooSe als gegeben angesehen.[2] Die Diskussion über ihre Vorteile ist in den letzten 15 Jahren hinreichend geführt worden und wird als abgeschlossen angesehen. Die Arbeit befasst sich auch nicht mit dem Vorgehen der ooSe, da hierzu eine Fülle von Veröffentlichungen existieren.[3]
- Die Arbeit befasst sich nur insoweit mit anderen Ansätzen zur GPM, wie dies dem Fortgang der Arbeit dient.[4] Auch Werkzeuge zur GPM werden nicht untersucht oder dargestellt. Evaluierungen bestehender Ansätze und am Markt erhältlicher Werkzeuge finden sich in anderen Veröffentlichungen.[5]
- In der Arbeit wird das Konzept der Geschäftsprozesse als gegeben angesehen. Sie befasst sich nicht mit der Fragestellung des innovativen Gehalts dieses Ansatzes. Es wird auch nicht der Fragestellung nachgegangen, inwiefern sich Aspekte dieses Ansatzes in der betriebswirtschaftlichen Literatur und insbesondere im Schrifttum der Organisationslehre nachweisen lassen.[6]
- Die Arbeit befasst sich nicht mit Fragestellungen und Problemen der Arbeitsteilung, Mitarbeitermotivation, Interessenkonflikten oder Auswirkungen des Geschäftsprozesskonzepts

---

[1] Auch etablierte Ansätze zur Geschäftsprozessmodellierung wie ARIS [Scheer 1998: S. 7 ff.] [Scheer 1998b: S. 41] und SOM [Ferstl 1998] setzen dies – eher implizit – voraus, da sie sich nur am Rand mit diesen Themen auseinandersetzen. Nur wenige Methoden – wie PROMET von ÖSTERLE [Österle 1995] und neuere Arbeiten von BECKER et. al. [Becker 2000] und FISCHERMANNS [Fischermanns 1997] – integrieren die Ebene der Strategieentwicklung in das Prozessmanagement. Vorgehensmodelle der Prozessgestaltung, die auch die Ebene der Strategieentwicklung beinhalten, finden sich in [Schulte-Zurhausen 1999: S. 74] und [Krüger 1994: S. 121].

[2] Beispielhaft findet sich eine Übersicht über die Vorteile des Objektansatzes in [Meyer 1995].

[3] Vgl. [Noack 1999], [Oestereich 1999], [Müller-Ettrich 1999], [Jacobson 1999], [Dröschel 1998], [Linssen 1999], [Kruchten 1998], [Balzert 1998], [Royce 1998].

[4] Vgl. [Fahrwinkel 1995], [Hess 1996], [Mattheis 1993]. Im Rahmen der Vorarbeiten wurden eine Reihe von Ansätzen auf ihre Verwendungsfähigkeit untersucht. Der Entity-Relationship-Ansatz wurde in [Linssen 1999e] untersucht, die UML in [Linssen 1999] und [Linssen 1999f], Szenarien, Sequenzdiagramme und Use Cases in [Linssen 1998], [Linssen 1999b] und [Linssen 1999f], die OBA in [Linssen 1996] und [Linssen 1998b] sowie der Werkzeug & Material-Ansatz in [Linssen 1999g].

[5] Die umfassendste und aktuellste Studie stammt vom Fraunhofer-Institut für Arbeitswissenschaft und Organisation IAO und vergleicht und bewertet insgesamt 31 Visualisierungs-, Modellierungs-, Simulations-, Workflow-Management- und CASE-Werkzeuge. Vgl. [Bullinger 2001b].

[6] Eine Betrachtung dieser Thematik findet sich in [Wolff 1998].

auf die Aufbauorganisation. Dadurch wird nicht bestritten, dass solche Auswirkungen existieren. Sie werden hier jedoch nicht behandelt.

- Der Modellierungsansatz erzwingt keinen organisatorischen Neuentwurf des Betrachtungsgegenstandes Unternehmen, wie dies in den Ansätzen des *Business Reengineering* oder *Business Process Reengineering* der Fall ist.[1]
- Der hauptsächliche Anwendungsbereich des entwickelten Modellierungsansatzes sind informationsverarbeitende Prozesse mit einem repetitiven, administrativen oder deterministischen Charakter,[2] da der Ansatz keine Modellprimitive für *fallweise Regelungen*[3] beinhaltet. Sie sind plan- und strukturierbar, werden nach Regeln ausgeführt und treten wiederholt auf.[4] Sie besitzen eine formale Struktur und einen bestimmten Informationsbedarf. Die Abbildung *informaler Regelungen* (die auf persönlichen Einstellungen und Motiven sowie auf persönlicher Sympathie beruhen[5]) wird vom Modellierungsansatz ebenfalls nicht unterstützt. Schließlich bestehen die Prozesse aus gleichbleibenden Elementen der Organisation und verfügen über einen bekannten und festgelegten Lösungsweg.[6]
- Der Ansatz dient der Entwicklung von Informationssystemen, die Geschäftsprozesse unterstützen. Er dient nicht zur Entwicklung von organisatorischen Lösungen und beinhaltet insbesondere keine Gestaltungsempfehlungen wie die Ansätze der Prozessorganisation oder der *Prozessorientierten Unternehmensorganisation*.[7]
- Der Modellierungsansatz dient zur GPM auf der Basis von Konzepten aus dem Software Engineering. Er stellt keinen Ansatz dar, die dort verwendeten Methoden, Formalismen und die

---

[1] Vgl. [Hammer 1994].

[2] Vgl. die Differenzierung unterschiedlicher Prozesstypen von SCHULTE-ZURHAUSEN und bei STRIENING. [Schulte-Zurhausen 1999: S. 123], [Striening 1988: S: 61]. Prozesse haben einen *repetitiven Charakter*, wenn sie strukturiert, wiederholbar und nach bestimmten Regeln ablaufen. Sie haben einen *administrativen Charakter*, wenn die Kommunikation schriftlich über einen vorgeschriebenen Dienstweg abläuft. Sie haben einen *deterministischen Charakter*, wenn sie durch Mensch-Maschine-Kommunikation geprägt sind.

[3] *Fallweise Regelung* beinhaltet *Disposition* (fallweise, punktuelle Einzelentscheidungen) und *Improvisation* (kurzfristig punktuelle Einzelentscheidungen). Vgl. [Schulte-Zurhausen 1999: S. 3], [Krüger 1994: S. 19]. Zur Bedeutung genereller und fallweiser Regelungen für die Organisation siehe auch [Gutenberg 1973: S. 238 ff.].

[4] Vgl. [Striening 1988: S. 61], [Schulte-Zurhausen 1999: S. 57], [Kosiol 1962: S. 31]. Nach dem *Substitutionsprinzip* der Organisation [Gutenberg 1973: S. 240] handelt es sich also um Prozesse mit sehr geringer Variabilität.

[5] Vgl. [Krüger 1994: S. 19].

[6] Vgl. [Schulte-Zurhausen 1999: S. 121].

[7] Vgl. [Engelmann 1995: S. 42 ff.], [Gaitanides 1983], [Striening 1988].

übliche Terminologie in die Fachsprache der Betriebswirtschaft, insbesondere die der Organisationslehre, zu übersetzen. In diesem Sinne wird die Betrachtung von Geschäftsprozessen in die Objektorientierung integriert. Dadurch grenzt er sich von den Ansätzen ab, die den umgekehrten Weg gehen.[1] Das bedeutet, dass für die Modellierung der Geschäftsprozesse die Denkweise und Terminologie des Objektansatzes maßgeblich ist, wie er in der praktischen und angewandten Informatik verwendet wird.[2] Die Zielgruppe des Modellierungsansatzes sind Software-Entwickler, die im Rahmen einer ooSe Geschäftsprozesse modellieren, und nicht Organisatoren, die im Rahmen von (Re-)Organisationsprojekten objektorientierte Software entwickeln sollen.

- Der Modellierungsansatz integriert nicht das Prinzip der Sichtenbildung des betriebswirtschaftlich orientierten Geschäftsprozessmanagements.[3] Da im objektorientierten Ansatz die Bildung von Sichten – also die getrennte Modellierung von Daten, Funktionen, der Aufbauorganisation und der Prozesse – abgelehnt wird, finden sich keine Teilmodelle für Daten, Organisationseinheiten und Funktionen wie in ARIS oder PROMET.[4]
- Viele der bisherigen Arbeiten befassten sich hauptsächlich damit, wie Geschäftsprozesse durch Standardsoftwaresysteme unterstützt werden können bzw. wie die von Standardsoftwaresystemen unterstützten Geschäftsprozesse dokumentiert werden.[5] Dieser Aspekt wird hier nicht betrachtet. Noch heute ist es so, dass Unternehmen ihre Prozesse an vorhandenen Standard-Informationssystemen ausrichten.[6] Wesentlich sinnvoller wäre es, wenn die Informationssysteme passend zum Geschäftsprozess (GP) entworfen wären bzw. Standardsysteme so angepasst werden könnten, dass sie zu den GP eines Unternehmens passen. Deshalb wird ein Ansatz entwickelt, Geschäftsprozesse zu beschreiben, bevor und unabhängig davon, ob sie durch eine Standard- oder Individualsoftware unterstützt werden. Es werden auch keine

---

[1] Vgl. [ZimmermannV 1999], [Bungert 1995], [Nüttgens 1998].

[2] Dies sei an drei Beispielen erläutert: (1) Bei einem *Objekt* handelt es sich in dieser Arbeit nicht um ein betriebswirtschaftliches Arbeitsobjekt, sondern um ein Exemplar einer Klasse. (2) Bei einem *dynamischen Modell* handelt es sich um die Darstellung des funktionalen Verhaltens und der Kontrollaspekte eines Systems und nicht um ein Verfahren der dynamischen Optimierung (vgl. [Hillier 1988]). (3) Bei einer *Transaktion* handelt es sich nicht um den Prozess der Klärung und Vereinbarung eines Leistungsaustauschs (vgl. [Picot 1982: S. 269]), sondern um eine unteilbare Ausführung einer Aktivität (vgl. [Lockemann 1993: S. 415 ff.]).

[3] Vgl. [Scheer 1992], [Becker 2000], [Scheer 1998b: S. 33 ff.].

[4] Vgl. [Scheer 1998], [Österle 1995].

[5] Vgl. [HMD 1998], [Brenner 1995].

[6] Immerhin ist es möglich, die Standardsoftware innerhalb bestimmter Bereiche anzupassen, was als *Customizing* oder *Redlining* bezeichnet wird. Vgl. [Scheer 1998b], [Scheer 1998], [Wenzel 1999].

Branchenreferenzprozesse entwickelt, die eine so genannte „*optimale Lösung*“ für betriebliche Abläufe darstellen wollen. Der Verfasser ist vom didaktischen Wert solcher idealisierten Abläufe überzeugt, hält aber die bisher zu diesem Thema veröffentlichte Literatur aktuell für erschöpfend.[1]

- Die Arbeit befasst sich nicht mit den Problemen der Prozessplanung.[2] Es werden keine quantitativen Modelle und Methoden des Operations Research verwendet. Die Arbeit stellt keinen Optimierungsansatz für betriebswirtschaftlich orientierte Fragestellungen der Ablauf-, Ressourcen- oder Terminplanung dar.[3]
- Der Modellierungsansatz soll in den frühen Phasen des objektorientierten Entwicklungsprozesses, der Anforderungs- bzw. Systemanalyse, eingesetzt werden, um Anforderungen an ein zu entwickelndes System zu untersuchen und zu dokumentieren. Eine automatische Generierung lauffähiger Softwaresysteme ist ohne manuelle Programmierung aus diesen Modellen zum heutigen Zeitpunkt nicht möglich bzw. befindet sich noch im Forschungsstadium. Auf die Phasen der Anforderungsanalyse und Systemspezifikation folgen die Phasen des Entwurfs und der Implementierung, deren Ergebnis ein lauffähiges System ist. Hierfür ist der Einsatz einer Reihe anderer Techniken und Methoden und eine Reihe von Entscheidungen notwendig, die nicht Inhalt dieser Arbeit sind. Aus diesem Grund wird von implementierungstechnischen Details, Fragen des Datenbankeinsatzes, der Benutzerschnittstelle oder der Netzwerkarchitektur abstrahiert. Das Ergebnis der Modellierung ist kein lauffähiges Programm.

---

[1] Vgl. [Scheer 1995], [Scheer 1998], [Scheer 1998b], [Kruse 1996], SPECK und SCHNETGÖKE in [Becker 2000: S. 179], [Schmelzer 2002: S. 58 ff.]. Ein wichtiger Vorläufer dieses Ansatzes ist das vom Betriebswirtschaftlichen Institut für Organisation und Automation an der Universität zu Köln (BIFOA) entwickelte Kölner Integrationsmodell (KIM), einem integrierten Gesamtmodell der Informationsverarbeitungsaufgaben industrieller Unternehmen. Vgl. [Grochla 1974], [Scheer 1995: S. 8 ff.]. Es soll nicht unerwähnt bleiben, dass zum Beispiel COOK und DANIELS den Wert solcher Modelle bestreiten. Vgl. [Cook 1994: S. 5, 14]. Vgl. auch die kritischen Anmerkungen zur Machbarkeit von Unternehmensdatenmodellen in [Picot 1994: S. 120] und in [Frank 1994].

[2] Vgl. [Gaitanides 1992: Sp. 7 - 10], [Schmidt 1997], [Jablonski 1997]. Unter Planung wird die gedankliche Vorbereitung von Entscheidungen verstanden.

[3] Eine Auseinandersetzung mit dem Problem der Planung von Prozessen aus Vorgängen gegebener Dauer unter Ressourcenbeschränkung mit alternativen Teilabläufen ist in der Arbeit von RITTGEN zu finden. Vgl. [Rittgen 1998]. Mit den Problemen der Reihenfolge- und Zuordnungsplanung auf Prozess- und Auftragsebene befasst sich die Veröffentlichung von SCHMIDT. Vgl. [Schmidt 1997].

- Die Modelle der UML sind zum heutigen Zeitpunkt nicht ausführbar.[1] Aus diesem Grund kann das Ergebnis der Modellierung nicht zur Simulation der abgebildeten Geschäftsprozesse verwendet werden. Hierfür bieten sich andere Ansätze an, wie sie zum Beispiel in den Arbeiten von MIELKE, VÖLKNER und MÜHLPFORDT zu finden sind.[2]
- Geschäftsprozesse unterscheiden sich von Workflows in der Weise, dass Workflows die automatisierten Teile eines Geschäftsprozesses darstellen.[3] Die GPM wird dabei als Vorstufe zur Modellierung von Workflows verwendet, wobei die GPM fachlich-konzeptuell orientiert ist, die Workflowmodellierung dagegen operativ ausgerichtet ist.[4] Das Ziel der Workflow-modellierung ist die Ausführung von Prozessen durch ein Workflow-Managementsystem (WfMS) oder durch eine entsprechende Standardsoftware.[5] Die Ausführung von Geschäfts-prozessen durch ein WfMS stellt aber nur eine Möglichkeit der technischen Realisierung von Geschäftsprozessen dar.[6] Aus diesem Grund werden eventuell vorhandene Anforderungen, die sich aus der Verwendung eines WfMS ergeben, nicht berücksichtigt.
- Eine andere Möglichkeit der Modellierung besteht darin, Geschäftsprozesse mit Hilfe von Petri-Netzen abzubilden.[7] Die Petri-Netze werden zur Modellierung der Prozesse,[1] Entity-

---

[1] Vgl. Abschnitt 2.4.1. Maschinell ausführbar wird die UML eventuell in naher Zukunft durch die *Action Semantics*, die Anfang 2002 von der OMG verabschiedet wurde. Vgl. [OMG 2002]. Es existieren aber zu diesem Zeitpunkt keinerlei Erfahrungen und insbesondere keine Werkzeuge, die diese Erweiterung verwenden.

[2] Vgl. [Mielke 2002], [Völkner 1998], [Mühlpfordt 1999].

[3] Vgl. [WfMC 1996]. Ein *Workflow* ist nach JABLONSKI ein Vorgang, der von einem Workflow-Managementsystem (WfMS) ausgeführt wird. Vgl. [Jablonski 1997: S. 24]. Es handelt sich also um den automatisierten Teil eines Geschäftsprozesses. Vgl. [WfMC 1996: S. 7, 9]. Ein WfMS dient der Funkti-onsintegration, so wie ein Datenbankmanagementsystem (DBMS) der Datenintegration dient. [Jablonski 2001]. Daraus folgt, dass bestimmte Fragestellungen, die im Bereich der WfMS untersucht werden (Beispiel *Transaktionsproblematik:* Prozesse wie *Mahnung versenden* oder *Bypassoperation durchführen* sind nicht rücksetzbar.), im Bereich der Geschäftsprozessmodellierung keine Rolle spielen und auch hier nicht betrachtet werden. Zur Transaktionsproblematik in WfMS vgl. [Leymann 1996]. Zum Workflow-Ansatz vgl. [Herrmann 1998], [Herrmann 1998b], [Jablonski 1995], [Jablonski 1995b], [Jablonski 1997], [Weikum 1997], [Götze 1995], [Oberweis 1996], [Raufer 1995], [Raufer 1997], [Rathgeb 1994], [Heilmann 1994], [Gruhn 1996], [Frank 1995], [Frank 1995b].

[4] Vgl. [Gehring 1998: KE 1, S. 4].

[5] Vgl. [Becker 1996b].

[6] Aktuelle Entwicklungen lassen die Vermutung zu, dass Geschäftsprozesse in Zukunft eher durch modular aufgebaute betriebliche Informationssysteme unterstützt werden.

[7] Vgl. [Jaeschke 1996], [Oberweis 1996], [Dinkhoff 1996]. Einführungen in Petri-Netze finden sich in [Peterson 1981], [Schnieder 1993], [Rosenstengel 1982] und in [Balzert 1996].

Relationship-Modelle werden zur Abbildung der im Prozess verwendeten Daten eingesetzt. Der Wert von Petri-Netzen wird nicht bestritten. Es existieren auch eine Reihe von Ansätzen, die eine Integration von Petri-Netzen und objektorientierten Konzepten versuchen.[2] Aber da in der ooSe Petri-Netze keine Verwendung finden und sie kein Bestandteil der UML sind, werden sie hier nicht weiter berücksichtigt.

1.8 Aufbau der Arbeit

Abgesehen vom Resümee in Kapitel 8 lassen sich die folgenden sieben Kapitel der Arbeit in zwei Abschnitte zerlegen. Im zweiten bis fünften Kapitel werden die Grundlagen geschaffen, ohne die objektorientierte GPM in der hier beabsichtigten Form nicht möglich ist. Das sechste und siebte Kapitel stellen den Kern der Arbeit dar, weil sie sich – im sechsten Kapitel – mit der objektorientierten Modellierung mit Skripten und – im siebten Kapitel – speziell mit der objektorientierten GPM befassen.

Das zweite Kapitel beschäftigt sich zuerst mit dem Modellbegriff. Es wird gezeigt, was unter Modell bzw. Modellierung verstanden wird. Das Ziel ist, die notwendigen Bestandteile eines Modellierungsansatzes herzuleiten. Dies sind seine *Metapher*, ein *Metamodell*, eine *Repräsentation*, eine *Architektur* und *Anwendungsbeispiele*. Die *Metapher* ist die Sichtweise, die einem Modellierungsansatz zu Grunde liegt. Die hier zu Grunde liegende Metapher ist der OA, der im weiteren Verlauf des Kapitels dargestellt wird. Anschließend wird die UML als Notation zur graphischen Modellierung von Systemen auf der Grundlage des Objektansatzes umrissen. Das Kapitel schließt mit der Darstellung der vorhandenen objektorientierten Ansätze zur Geschäftsprozessmodellierung und den Anforderungen an die GPM.

Ausgangspunkt des Ansatzes ist die Anfang der neunziger Jahre von RUBIN und GOLDBERG entwickelte Object Behavior Analysis (OBA), die zu Beginn des dritten Kapitels dargestellt wird. Es wird aufgezeigt, welche Schwächen der OBA durch eine Weiterentwicklung behoben werden müssen. Anschließend wird eine aus drei Ebenen bestehende *Architektur* verwendet, die aus einem *Externen*, einem *Konzeptuellen* und einem *Internen Schema* besteht.[3] Das Externe Schema stellt die

---

[1] Vgl. [Lausen 1988], [Schnieder 1993].

[2] Vgl. [Zapf 2000].

[3] Bei den Bezeichnungen *Externes*, *Konzeptuelles* und *Internes Schema* und dem später noch eingeführten Begriff *Semantisches Netz* werden die Adjektive groß geschrieben, auch wenn sie nicht zu den in § 64 der amtlichen Regelungen der deutschen Sprache aufgeführten Ausnahmen gehören (vgl. [Duden 2000: S. 1138]). Damit wird den sogenannten *Agenturschreibungen* gefolgt (vgl. [Duden 2000: S. 120]). In diesen wird festgelegt, dass bei fest gebrauchten Bezeichnungen aus Adjektiv und Substantiv in bestimmten Fällen das Adjektiv auch dann groß geschrieben wird, wenn es nicht zu den in § 64 genannten Gruppen gehört.

Repräsentation des Modellierungsansatzes dar. Das Konzeptuelle Schema ist das Gesamtmodell aller Externen Schemata. Das Interne Schema stellt die softwaretechnische Realisierung des Konzeptuellen Schemas dar. Im weiteren Verlauf des Kapitels wird an einem vereinfachten Beispiel exemplarisch gezeigt, wie der Inhalt des Externen auf das Konzeptuelle und das Interne Schema abgebildet wird.

Im vierten Kapitel wird durch das *Metamodell* des Konzeptuellen Schemas ein Begriffssystem für die verwendete Metapher geschaffen. Auf diese Weise wird unabhängig von einer Repräsentation festgelegt, wie die Begriffe der Metapher verwendet werden. Das Metamodell besteht aus Knoten, Kanten und *Prädikaten* und wird in Form von UML-Klassendiagrammen festgelegt.

Die Modellierung verwendet zur Repräsentation Skripte, eine tabellarische Darstellungsform. Sie stellen eine Sicht (ein Externes Schema) auf das Konzeptuelle Schema dar. Für ihren Aufbau wird im fünften Kapitel ein weiteres Metamodell erstellt. Auch dieses Metamodell wird in Form von UML-Klassendiagrammen und durch eine Grammatik in Form einer erweiterten BACKUS-NAUR-Form (eBNF) festgelegt. Anschließend werden die Verfahren entwickelt, mit denen der Inhalt der Skripte auf das Konzeptuelle Schema abgebildet wird. Anhand mehrerer Beispiele wird gezeigt, dass – neben den Skripten – auch andere Sichten verwendet werden können.

Nachdem durch das Konzeptuelle Schema ein Metamodell der Metapher und durch ein weiteres Metamodell der Aufbau der Repräsentation festgelegt wurde, ist das Thema des sechsten Kapitels die Anwendung (im Sinne von *Anwendungsbeispielen*) der Skripte. Dabei wird ausführlich auf die im fünften Kapitel definierten Prädikate eingegangen. Außerdem werden eine Reihe von Erweiterungen der UML entwickelt. Das Ziel dieses Kapitels ist es darzustellen, wie man mit Skripten modelliert.

Der Inhalt des siebten Kapitels ist die objektorientierte GPM. Dieses Kapitel zeigt, wie Geschäftsprozesse mit Skripten modelliert werden. Zunächst wird anhand der relevanten Literatur zum Thema GPM untersucht, welche Phänomene (im Sinne von Anforderungen an den Modellierungsansatz) modellierbar sein müssen. Danach wird auf der Grundlage der Verfahren aus dem sechsten Kapitel gezeigt, wie diese Anforderungen mit Hilfe des hier entwickelten Modellierungsansatzes erfüllt werden. Dabei wird auch deutlich gemacht, welche Unterschiede zwischen der Objektorientierung und der Geschäftsprozessorientierung bestehen.

Das achte Kapitel fasst im Resümee die Ergebnisse der Arbeit zusammen.

2 Der Objektansatz

„Die Monaden, von denen meine Schrift handeln wird, sind nichts weiter als einfache Substanzen, welche in dem Zusammengesetzten enthalten sind. Einfach heißt, was ohne Teile ist. Einfache Substanzen muss es geben, weil es Zusammengesetztes gibt; denn das Zusammengesetzte ist nicht anderes als eine Anhäufung oder ein Aggregat von Einfachem.“
[Leibnitz 1979: S. 13]

2.1 Modelle

2.1.1 Modellbegriff

GROCHLA versteht unter einem Modell[1] ein abstraktes, vereinfachendes Abbild eines Systems, welches zu einem bestimmten erkenntnistheoretischen oder gestaltungsspezifischen Zweck entwickelt wird.[2] Abstrakter ist die Definition von KÖHLER:[3] Er definiert Modell als die Relation zwischen zwei Systemen, dem Original und dem Abbild. Das Original ist das, was abgebildet wird, dass Modell ist das Abbild des Originals. Ein Modell ist daher immer ein Modell von etwas anderem.[4] Welcher Art die Relation ist, ist von Fall zu Fall unterschiedlich und wird von einem Subjekt durch dessen Zielsetzungen, Absichten usw. festgelegt. Die Definition BECKERs, ein Modell repräsentiere ein immaterielles und abstraktes Abbild der Realwelt zum Zwecke eines Subjekts, ist prägnant, schließt aber materielle Modelle aus.[5]

---

[1] Der Begriff Modell wird in vielerlei Bedeutung verwendet. Der Duden definiert Modell u. a. als *„innere Beziehungen und Funktionen von etwas abbildendes bzw. [schematisch] veranschaulichendes [und vereinfachendes, idealisierendes] Objekt, Gebilde.“* [Duden 1993].

[2] [Grochla 1974: S. 22]. Siehe auch [Jablonski 1997: S. 35 ff.], [Kruse 1996: S. 13].

[3] Vgl. [Köhler 1975: Sp. 2701 ff.]. Ähnlich auch in [Ferstl 1998 S. 18].

[4] Dies bezeichnet STACHOWIAK als *Abbildungsmerkmal*. Vgl. [Stachowiak 1973: S. 131].

[5] Vgl. [Becker 1996: S. 19].

Ein Modell abstrahiert[1] vom Original, da bestimmte Dinge, die für die Betrachtung als nicht relevant angesehen werden, weggelassen werden.[2] Ein Modell ist also in der Regel auch eine Interpretation des Originals bzw. der Realität. Durch das Modell wird eine Vorstellung abgebildet, die man sich von einem Gegenstand oder Vorgang der Umwelt macht.[3] In die Erstellung des Modells fließen also subjektspezifische Wertungen und Zielinhalte ein.

Bei *Isomorphie* liegt eine umkehrbar eindeutige Zuordnung der Elemente des Modells und des Originals vor. Jedem Element des Originals ist genau ein Element des Modells zugeordnet und umgekehrt. Bei *Homomorphie* liegt zwar eine eindeutige, aber keine umkehrbare Zuordnung vor.[4] Es gibt Elemente des Originals, denen kein Element im Modell zugeordnet ist. Man spricht auch von Ähnlichkeit oder Analogie. Üblicherweise werden homomorphe Modelle verwendet, da eine Weltverdopplung als unfruchtbar angesehen wird.[5]

Zusammengefasst kann man Modelle als vereinfachende Abbildungen eines Ausschnitts der Wirklichkeit unter Anwendung einer bestimmten Sichtweise bezeichnen.[6] Ein Modell kann eine Darstellung eines Teils der vergangenen, gegenwärtigen oder zukünftigen Wirklichkeit darstellen.[7] Ziel der Modellbildung ist es, das Abbild der Realwelt so zu vereinfachen, dass es für die vom

---

[1] Unter *abstrahieren* wird verallgemeinern, aus dem Besonderen das Allgemeine entnehmen verstanden. Unter *Abstraktion* wird ein verallgemeinerter, unanschaulicher Begriff verstanden. Vgl. [Duden 1993]. PARTSCH definiert Abstraktion wie folgt: „*Unterdrückung von Details und Konzentration auf die wesentlichen Eigenschaften.*“ [Partsch 1998: S. 42]. Vgl. auch [Brockhaus 2000].

[2] Dies bezeichnet STACHOWIAK als *Verkürzungsmerkmal*. Vgl. [Stachowiak 1973: S. 132]. BAETGE betrachtet Modelle als abstrakte Systeme, welche andere Systeme in vereinfachter Form abbilden. Vgl. [Baetge 1974: S. 47]. Vgl. auch [Henderson-Sellers 2000: S. 54].

[3] Vgl. [Jablonski 1997: S. 35], [Schmidt 1985b: S. 17].

[4] Vgl. [Milling 1981: S. 97 ff.], [Köhler 1975: Sp. 2706], [Stachowiak 1973: S. 94 ff.]. Hier verbirgt sich die Fragestellung, wann ein Modell „richtig“ oder mindestens adäquat für eine bestimmte Fragestellung ist. Diesem Problem der Ähnlichkeit zwischen Original und Modell versucht man aktuell durch die Entwicklung von Grundsätzen ordnungsgemäßer Modellierung beizukommen. Vgl. [Becker 1995], [Becker 1995b], [Scheer 1998b: S. 119 f.], [Becker 2000]. Als generelle Handlungs*richtlinien* sind solche Grundsätze begrüßenswert. Der Beweis, dass sie in realen Problemsituationen eine Entscheidung ermöglichen, steht aber nach Ansicht des Verfassers noch aus. Vgl. zu diesem Thema auch [Frank 1997f].

[5] Vgl. [Corsten 1994: S. 51], [Eichhorn 1979: S. 66]. STACHOWIAK geht davon aus, dass Modelle ihren Originalen nicht eindeutig zuzuordnen sind. Sie erfüllen ihre Ersetzungsfunktion für bestimmte Subjekte innerhalb eines bestimmten Zeitraums und unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen. Dies bezeichnet er als *pragmatisches Merkmal*. Vgl. [Stachowiak 1973: S. 132 ff.].

[6] Vgl. [Rechenberg 1999: S. 1022], [Erichson 1993: S. 190].

[7] Vgl. [Keller 1998: S. 117].

Subjekt verfolgten Zwecke handhabbar ist.[1] Erst dadurch wird die Bildung von Aussagen über komplexe Systeme möglich, was auch die Evaluation und Bildung von Gestaltungsalternativen einschließt.[2] Hier wird unter Modell eine homomorphe Abbildung von *Phänomenen*[3] eines Originals auf der Basis einer Abbildungsvorschrift verstanden:

- Ausgangspunkt sind die Phänomene des Originals.
- Berücksichtigt werden nicht alle Phänomene des Originals.
- Die Zuordnung der Erscheinungen zu Elementen des Modells basiert auf einer bestimmten Sichtweise.

2.1.2 Modelle im Software Engineering, in der Wirtschaftsinformatik und im Prozessmanagement

„Es ist seit langem Konsens, dass die konzeptuelle Modellierung eine wesentliche Voraussetzung für die Entwicklung qualitativ hochwertiger Software darstellt. [...] Anders als es die vermeintliche Anschaulichkeit grafischer Modelle suggerieren mag, ist die konzeptuelle Modellierung mit großen Herausforderungen verbunden."[4]

Das heutige Software Engineering ist ohne seine Vielzahl unterschiedlicher Modellarten undenkbar.[5] Ursache ist die inhärente Komplexität von Softwaresystemen.[6] Modelle werden u. a. dazu verwendet, um die Anforderungen an ein zu realisierendes Softwaresystem zu sammeln, zu verstehen und auf ihre Widerspruchsfreiheit zu prüfen. Diese Modelle dienen als Kommunikationsmedium zwischen den am Entwicklungsprozess beteiligten Personen.[7] Sie sind der

---

[1] Auf die Grundlagen der allgemeinen Theorie für Modelle kann hier nicht eingegangen werden. Der Leser sei hier exemplarisch auf die Arbeit von STACHOWIAK [Stachowiak 1973] verwiesen. Dort findet sich unter anderem eine ausführliche Darstellung der unterschiedlichen Modelltypen. Vgl. [Stachowiak 1973: S. 159 – 303]. Mit der Verwendung von Modellen in der Technik befasst sich u. a. ROPOHL [Ropohl 1979]. Mit dem sozialen Prozess der Modellbildung in der Informatik befasst sich FLOYD in [Floyd 1998].

[2] Vgl. [Hars 1994: S. 8], [Frank 1994: S. 15], [Krallmann 1994: S. 10].

[3] *Phänomen* wird im Sinne von Erscheinung verwendet. Damit sind die äußeren Eigenschaften von Dingen und Prozessen gemeint, die dem Betrachter durch Anschauung bzw. Wahrnehmung oder Erfahrung gegeben sind. Vgl. [Klaus 1972], [Ferber 1999].

[4] [Frank 1997f: S. 1].

[5] Eine kurze Übersicht über die Natur und Aufgaben von Modellen in der Softwareentwicklung findet sich in [Rumbaugh 1999: S. 13 ff.].

[6] Vgl. [Booch 1995], [Quibeldey-Cirkel 1994: S. 20].

[7] [Partsch 1998: S. 11].

Ausgangspunkt der weiteren Entwicklung.[1] In deren weiterem Verlauf werden Modelle entwickelt, mit denen die Architektur der Softwarekomponenten beschrieben wird, die Benutzerschnittstelle, die Speicherung von Daten usw. Der heutige Stand der Technik ist, dass zumindest Teile eines Softwaresystems direkt generiert werden können, wenn die Modelle einen entsprechenden Grad der Formalisierung besitzen. JACOBSON hat auch den gesamten Entwicklungsprozess von Software als Abfolge von Modellerstellungsprozessen bezeichnet.[2] In der Wirtschaftsinformatik stellen Modelle von Informationssystemen das wichtigste Hilfsmittel zur Analyse und Gestaltung dar:[3] „*Die Modellierung betrieblicher Systeme – der Begriff betriebliches System wird im folgenden als Oberbegriff für Unternehmen, Unternehmensverbunde oder Geschäftsbereiche von Unternehmen verwendet – ist eine zentrale Aufgabe der Wirtschaftsinformatik und besitzt dort eine lange Tradition.*“[4]

Die Bedeutung von Modellen in der Betriebswirtschaftslehre braucht hier nicht weiter vertieft zu werden. Als „Denken an Modellen“ wird es in der Wirtschaftswissenschaft nach EICHHORN allgemein anerkannt.[5] Bei PICOT liest man: „*Zur Gestaltung betrieblicher Informationssysteme und deren Einbindung in den Gesamtzusammenhang einer Unternehmung bedarf es prinzipiell einer modellhaften Abstraktion.*“[6] Die Verwendung von Modellen ist Usus: *„Modelle von Unternehmen sind seit langem ein bedeutendes Instrument der betriebswirtschaftlichen Forschung.“*[7] Allerdings ist zu beachten, dass es „*kaum möglich* [ist]*, die gesamte Komplexität eines Gebildes, das wir als Unternehmen bezeichnen, zu erfassen und systematisch zu beschreiben.*“ [8] „*Kein Modell kann die Vielfalt der im Betriebe wirkenden Prozesse, Vorgänge, Handlungen und Abläufe wiedergeben. Jedes Modell muss also mit Abstraktionen arbeiten.*“[9] Die Darstellung der Modelle erfolgt sowohl in verbaler, graphischer wie auch in analytischer Form.[10]

---

[1] [Partsch 1998: S. 29].

[2] [Jacobson 1994]. Andere Veröffentlichungen, die diese Position unterstreichen, sind [Embley 1992: S. 5] und [McMenamin 1988: S. 36 ff.].

[3] Vgl. [Ferstl 1994: S. 3], [Ferstl 1998: S. 117], [Frank 1994]. Vgl. hierzu auch die jährlichen Tagungsbände über die Modellierung betrieblicher Informationssysteme. Vgl. [MobIS 1996], [MobIS 1997], [MobIS 1998], [MobIS 1999], [MobIS 2000].

[4] [Ferstl 1995: S. 209 f.].

[5] [Eichhorn 1979: S. 65].

[6] [Picot 1994: S. 107].

[7] [Frank 1994: S. 11].

[8] [Hars 1993: S. 8].

[9] [Wöhe 1986: S. 37].

[10] Vgl. [Eichhorn 1979: S. 66 - 74]. Mit *analytischer Form* sind mathematische Formeln gemeint.

Auch das Gebiet des Prozessmanagements bzw. Geschäftsprozessmanagements basiert in starkem Maße auf der Entwicklung und Verwendung von Modellen: *„Um Geschäftsprozesse zu verstehen und auf sie einwirken zu können, ist es notwendig, sie angemessen zu modellieren, d. h. geeignete Modelle für Geschäftsprozesse zu bilden.“* [1] Modelle sollen dabei

- die Transparenz über die Elemente und deren Beziehungen innerhalb eines Systems schaffen,
- der Erklärung der Funktionsweise eines Systems dienen,
- die Kommunikation durch eine konsistente Formalisierung unterstützen.[2]

Die in der Betriebswirtschaft übliche Unterscheidung von Beschreibungsmodellen, Erklärungsmodellen, Prognosemodellen und Entscheidungsmodellen[3] ist allerdings bei den im Software Engineering verwendeten Modellen nicht bekannt. Es überwiegt der beschreibende Charakter der Modelle, da man versucht, die Phänomene der realen Welt abzubilden. Die Entwicklung eines Softwaresystems beinhaltet aber auch einen erklärenden Charakter, wenn durch Modelle für bestimmte Phänomene (Probleme) Lösungen in Form von Algorithmen entwickelt werden. In diesem Fall werden durch den Algorithmus Aussagen über Wirkungszusammenhänge der realen Welt im Sinne nomologischer Hypothesen getätigt.[4] Dass solche Modelle den Charakter von Entscheidungsmodellen besitzen können, also die Bestimmung optimaler Handlungsmöglichkeiten erleichtern,[5] mag vielleicht im Einzelfall zutreffen. Die Aussage aber, dass durch die Modelle des Software Engineerings eine optimale Lösung bestimmt werden kann, wird man kaum treffen wollen, wenn man sich in der Praxis mit den Vor- und Nachteilen unterschiedlicher Modelle für das gleiche Phänomen beschäftigt hat.

[1] [Becker 1996: S. 19]. Vgl. auch [Becker 1995: S. 134], [Fischermanns 1997: S. 113].

[2] [Keller 1998: S. 118].

[3] Vgl. [Wöhe 1986: S. 39 ff.], [Eichhorn 1979], [Grochla 1975c], [Corsten 1994: S. 55 f.], [Köhler 1975], [Krallmann 1994: S. 16]. Mit den Grundlagen der Bildung von Entscheidungsmodellen im Bereich der Unternehmensplanung auf der Basis der Systemtheorie befassen sich beispielsweise MILLING [Milling 1981] oder BAETGE [Baetge 1974].

[4] Als *nomologische Hypothesen* werden allgemein gültige Zusammenhänge im Sinne von gesetzesmäßigen Eigenschaften, Zustands- und Ereigniszusammenhängen bezeichnet. Vgl. [Wild 1966: S. 56]. In diesem Zusammenhang kann das Testen eines Programms als Versuch der Falsifizierung verstanden werden. Scheitert die Falsifizierung, gilt das Programm bis auf weiteres als richtig.

[5] Vgl. [Wöhe 1986: S. 40], [Eichhorn 1979: S. 95].

2.1.3 Aufbau des Modellierungsansatzes

Voraussetzung für die Modellierung ist nach FERSTL und SINZ ein Beschreibungsrahmen, durch den die bei der Modellierung verwendete Sichtweise auf Original und Modell sowie das bei der Modellierung verwendete Begriffssystem festgelegt wird. Dies wird hier als *Modellierungsansatz* bezeichnet. Demnach besteht ein Modellierungsansatz aus folgenden Teilen: [1]

- Einer *Metapher* zur Beschreibung der Sichtweise.
- Einem *Metamodell* zur Definition eines mit der Metapher abgestimmten Begriffssystems.[2]

Diese Bestandteile entsprechen der im Abschnitt 2.1.1 erwähnten Abbildungsvorschrift, durch die festgelegt wird, welche Phänomene der Wirklichkeit auf welche Weise Eingang in das Modell finden. Darüber hinaus sollen hier drei weitere Teile den Modellierungsansatz ergänzen:

- Die *Repräsentation* legt fest, welche äußere Form die Modelle besitzen.
- Die *Architektur des Modellierungsansatzes* definiert die Beziehungen zwischen dem Metamodell des Begriffssystems und seiner Repräsentation.
- Die *Anwendungsbeispiele* erläutern, wie der Modellierungsansatz verwendet wird.

Der Ausgangspunkt ist die Metapher. Die Metapher legt eine bestimmte Sichtweise auf ein Original fest. Der im Rahmen dieser Arbeit entwickelte Modellierungsansatz verwendet als Metapher den OA. Darauf aufbauend wird durch das *Metamodell* für diese Metapher ein Begriffssystem gebildet. Das Metamodell definiert nach FERSTL und SINZ „*die verfügbaren Arten von Modellbausteinen, die Regeln für die Verwendung von Modellbausteinen durch Beziehungen sowie die Bedeutung (Semantik) der Modellbausteine und Beziehungen*.“[3] Vereinfacht ausgedrückt

---

[1] [Ferstl 1998: S. 119].

[2] ZÜLLIGHOVEN spricht von Begriffsgerüst. Vgl. [Züllighoven 1998: S. 20].

[3] [Ferstl 1998: S. 120]. Wenn ein Objektsystem OS durch ein Modellsystem MS abgebildet wird und ein Modell M2 für das Modellsystem MS existiert, so stellt M2 das Metamodell in Bezug auf das OS dar. Vgl. [Hars 1993: S. 11]. Eine der ersten Veröffentlichungen, die sich mit der Metamodellierung im Rahmen des Objektansatzes auseinandergesetzt hat, war [Blaha 1992]. Das Metamodell der UML ist in [OMG 1999] definiert. Der Abschnitt *2.2.1 Four-Layer Metamodell Architecture* setzt sich mit den Beziehungen zwischen dem Meta-Metamodell (M3), dem Metamodell (M2), Modellen (M1) und Daten bzw. Objekten (M0) auseinander. Diese Modell-Architektur basiert zum Teil auf der META OBJECT FACILITY (MOF) SPECIFICATION der OMG, die gleichzeitig die Ebene M3 darstellt. Vgl. [OMG 1999b]. STRAHRINGER setzt sich auf allgemeiner Ebene mit dem Begriff des Metamodells in der Informatik auseinander. Vgl. [Strahringer 1998]. Eine knappe Darstellung des Meta-Metamodells, des Metamodells, der Modelle und Objekte der Methode ARIS findet sich in [Scheer 1998b: S. 119]. Eine ausführliche Darstellung des Meta-

handelt es sich bei einem Metamodell um ein Modell eines Modells.[1] Da für die im Metamodell definierten Inhalte prinzipiell unterschiedliche Darstellungen verwendet werden können, wird durch eine *Repräsentation* festgelegt, welche konkrete Form der Darstellung verwendet wird. Die Repräsentation legt dadurch die äußere Form des Modellierungsansatzes fest. Auch der Aufbau der Repräsentation wird hier durch ein Metamodell festgelegt. Die Beziehungen zwischen dem Metamodell des Begriffssystems und dem Metamodell der Repräsentation werden hier durch die *Architektur des Modellierungsansatzes* festgelegt. Die Anwendung des Modellierungsansatzes wird schließlich durch Beispiele erläutert.

Ein *Vorgehensmodell* wird für den hier entwickelten Modellierungsansatz nicht entwickelt, da er im Rahmen des Entwicklungsprozesses der Softwareentwicklung eingesetzt werden soll.[2] Etablierte Methoden besitzen dafür eigene Vorgehensmodelle:

- Im UNIFIED SOFTWARE DEVELOPMENT PROCESS wird im Rahmen des Arbeitsablaufs der Anforderungsanalyse u. a. festgestellt, welche Abläufe ein zu entwickelndes Softwaresystem unterstützen muss und in welchem Kontext dieses System eingesetzt wird.[3]
- Im RATIONAL UNIFIED PROCESS (RUP) existiert der Arbeitsablauf (engl. *workflow*) zur Geschäftsmodellierung. In diesem Prozess werden Geschäftsprozesse abgegrenzt, dokumentiert und überarbeitet.[4]
- Auch die OPEN-Prozessspezifikation sieht die GPM vor.[5]
- Im V-MODELL 97 gehört zum Submodell Systemerstellung (SE) eine integrative Behandlung von Ist-Systemen inklusive einer ganzheitlichen Betrachtung des organisatorischen Einsatzbereichs und der Geschäftsprozesse. Dabei sind auch die Teile der Geschäftsprozesse zu dokumentieren, die nicht durch Softwaresysteme unterstützt werden.[6]

Metamodells der Methode MEMO findet sich in [Frank 1998]. Zu den Begriffen Methode, Methodologie, Modell, Metamodell und Modellierungsmethode siehe auch [Frank 1997e: S. 8 ff.].

[1] Vgl. [Martin 1999: S. 307].

[2] Aus diesem Grund wird hier auch nicht von der Entwicklung einer *Methode* gesprochen. Wie auch in der Betriebswirtschaftslehre (vgl. [Kosiol 1962: S. 34], [Wild 1966: S. 43]) sind Methoden planmäßige Verfahren zur Erreichung eines Zieles. Sie liefern Vorgaben für ein systematisches Vorgehen bei der Softwareentwicklung. Vgl. [Rechenberg 1999: S. 772]. Eine Methode stellt im Software Engineering eine Einheit dar, die mindestens aus einem Modellierungsansatz und einem Vorgehensmodell besteht.

[3] Vgl. [Jacobson 1999: S. 111].

[4] Vgl. [Rational 1999], [Kruchten 1999: S. 133 – 140].

[5] Vgl. [Graham 1997: S. 85 ff., 231 ff.], [Henderson-Sellers 1998: S. 117 f.], [Fowler 2000], [Henderson-Sellers 2000: S. 196]

[6] Vgl. [Dröschel 1998: S. 43, 48], [V-Modell 1997: SE 1.1, SE 1.5, GPO].

Durch den folgenden Abschnitt wird mit der Objektorientierung das Fundament des Modellierungsansatzes – seine Metapher – eingeführt.

2.2 Die Metapher ‚Objektorientierung'

Der OA ist in erster Linie ein Maschinenmodell, welches einen Alternativentwurf zur VON-NEUMANN-Organisation von Rechnern darstellt.[1] Er entstand in den 60er-Jahren, um Systeme zu simulieren, deren Verhalten sich nicht algorithmisch beschreiben ließ. Dies geschah zuerst durch die in Norwegen entwickelte Programmiersprache SIMULA.[2] Die Ideen von SIMULA wurden durch verschiedene Programmiersprachen weiterentwickelt, wobei insbesondere FLAVOURS, ACTOR, SMALLTALK, OBJECTIVE-C, C++, EIFFEL und JAVA zu erwähnen sind.

Was heute als OA bezeichnet wird, basiert also in erster Linie auf Programmierkonzepten. Darüber hinaus wurden Anleihen in anderen Disziplinen getätigt. Der Gedanke, die Beziehungen zwischen den Elementen eines Systems graphisch zu beschreiben, stammt aus der Entity-Relationship-Modellierung bzw. aus der semantischen Datenmodellierung.[3] Gewisse Bezüge zum Forschungsgebiet der Künstlichen Intelligenz sind ebenfalls nachzuweisen, wobei hier insbesondere *Semantische Netze* und *Frames*[4] zu erwähnen wären. Außerdem wurden mit den Abstrakten Datentypen (ADT) und den Zustandsmaschinen Konzepte aus der praktischen und theoretischen Informatik übernommen.[5] Darüber hinaus wird der OA gerne als allgemeiner Denkansatz bezeichnet, dessen Fundament sich in der Philosophie, Kognitionsforschung, Systemtheorie und Biologie finden lasse.[6]

---

[1] Vgl. [Stoyan 1991]. Die sieben Prinzipien der Rechnerorganisation nach VON NEUMANN finden sich in [Rechenberg 1999: S. 297].

[2] [Nygaard 1986], [Louden 1994: S. 365], [Rechenberg 1999: S. 544].

[3] Vgl. [Peckham 1988], [Mylopoulos 1999].

[4] Vgl. [Reimer 1991], [Helbig 1996], [Winston 1992].

[5] Vgl. [Pomberger 1993], [Quibeldey-Cirkel 1994], [Meyer 1997], [Isernhagen 2000]. Abstrakte Datentypen werden ausschließlich über ihre Operationen definiert. Die interne Repräsentation der Daten und die verwendeten Algorithmen der Operationen sind von außen nicht sichtbar. Vgl. [Balzert 1999: S. 23]. Zu den Konzepten abstrakter Datentypen und Zustandsmaschinen vgl. [Liskov 1974], [Guttag 1977], [Guttag 1978], [Liskov 1986], [Floyd 1996], [Ehrich 1989]. Siehe auch die Gegenposition von COOK, der ADTs und Objektorientierung als getrennte Konzepte ansieht. Vgl. [Cook 1990].

[6] Vgl. [Booch 1995: S. 54 ff.], [Partridge 1994], [Miller 1978], [Leibnitz 1979], [Hirschberger 2000], [Quibeldey-Cirkel 1994: S. 152 ff.].

Auf diese Weise hat sich der OA in den letzten 25 Jahren zu einem umfangreichen Begriffssystem entwickelt.[1] Tatsächlich weicht aber Aufbau und Inhalt dieses Systems von Autor zu Autor, von Methode zu Methode und von Programmiersprache zu Programmiersprache voneinander ab.[2] Die führt dazu, dass die Terminologie unterschiedlich verwendet wird.[3] PRASSE schreibt zu dieser Problematik: *„Die Konzepte Objekt, Klasse und Vererbung bilden die Eckpfeiler im objektorientierten Paradigma. Trotzdem gibt es keine einheitliche Interpretation dieser zentralen Begriffe."*[4] Dies kann man auf nahezu alle Begriffe des Objektansatzes ausdehnen. Es gibt nicht *das* objektorientierte Paradigma im Sinne eines geschlossenen Begriffssystems. Jeder Verfasser eines Modellierungsansatzes definiert seinen eigenen OA, welcher auch seine persönlichen Ansichten widerspiegelt. Im weiteren Verlauf des Kapitels werden zunächst die wesentlichen Begriffe des Objektansatzes dargestellt, ohne dass dabei auf Konzepte von Programmiersprachen zurückgegriffen wird. Der Inhalt orientiert sich an dem, was als *Common Sense* anzusehen ist.

2.2.1 Der Objektansatz als Weltmodell

Der OA stellt als Weltmodell bzw. *Systemparadigma*[5] eine bestimmte Sichtweise dar, die Umwelt zu betrachten. Er basiert auf der Sichtweise, dass die Welt aus interagierenden Objekten besteht, die voneinander Leistungen in Form von Operationen anfordern.[6] Systeme sind innerhalb dieser Welt eine geordnete Menge (Gemeinschaft bzw. Gesellschaft[7]) von interagierenden Objekten,[8] für die man entscheidet, dass sie als ein Ganzes angesehen werden sollen.[9] Dadurch wird das System gegenüber der Umwelt und anderen Systemen abgegrenzt. Aus dem Systemgedanken ist abzuleiten, dass die Objekte eines Systems in Beziehung zueinander stehen. Objekte können selbst

---

[1] Pointiert könnte man auch von einem ausufernden Begriffs*wirrwarr* sprechen. Aber dies stellt (m)eine persönliche Wertung dar.

[2] Vgl. [Rauschecker 2000: S. 118], [Firesmith 1995], [Saake 1993: S. 18], [Züllighoven 1998: S. 19].

[3] Vgl. [Booch 1995: S. 45, 56], [Henderson-Sellers 1994: S. 46], [Meyer 1997: S. vi].

[4] [Prasse 1998: S. 23]. Frühe Darstellungen (bezogen auf die Zeitdimensionen des objektorientierten Software Engineerings, sind damit die späten 80er-Jahre gemeint) sind noch kurz und prägnant. Vgl. [Stefik 1986], [Wegner 1987]. Spätere Darstellungen wachsen zu umfangreichen Sammlungen heran, wie an [Firesmith 1995] zu sehen ist.

[5] [Saake 1995: S. 18].

[6] Vgl. [Coleman 1992: S. 14]. Eine einführende Darstellung in die objektorientierte Denkweise findet sich in [Stoyan 1991: S. 181 ff.].

[7] Man spricht auch von Objekt*gesellschaften*. Vgl. [Saake 1995: S. 19].

[8] Vgl. [Embley 1992], [Stoyan 1991: S. 184], [Saake 1993], [Berard 1993: S. 166, 175].

[9] Vgl. [Reenskaug 1996: S. 36].

Systeme sein,[1] die sich aus Objekten zusammensetzen. Was als Objekt und was als System betrachtet wird, ist von der aktuell gewählten Betrachtungsebene abhängig. MILLER hat dieses Prinzip verwendet, um die hierarchische Struktur von Zellen, Organen, Organismen, Gruppen, Organisationen, Gesellschaften und supranationalen Systemen zu untersuchen.[2] Der objektorientierte Ansatz ist also ein bestimmter Blickwinkel, die aus Objekten bestehende Welt zu betrachten.

2.2.2 Objekte

Objekte sind nach BOOCH greifbare und/oder sichtbare Dinge, etwas, was intellektuell wahrnehmbar ist oder etwas, worauf sich unser Denken und Handeln bezieht.[3] Im OA gilt das *Vollständigkeitsprinzip*, nach dem alles ein Objekt sein kann. Damit können Objekte gemäß dem Objektansatz sowohl Dinge wie auch Personen, Ereignisse, Beziehungen oder abstrakte Konzepte sein.[4]

Alle Objekte werden (a) durch ihre Eigenschaften und (b) die Dinge, die sie tun, beschrieben. Drittes Merkmal von Objekten ist (c) ihre Einzigartigkeit. In der Literatur werden diese drei Eigenschaften als (a) ihr Zustand bzw. ihr Status, (b) ihr Verhalten und (c) die Objektidentität bezeichnet.[5]

Zu (a): Objekte haben beobachtbare Eigenschaften,[6] durch die sie beschrieben werden können. Die Gesamtheit der Eigenschaften bezeichnet man als Struktur des Objekts.[7] Die einzelnen Eigenschaften heißen *Attribute*.[8] Der Zustand bzw. Status eines Objekts umfasst die (normaler-

---

[1] Vgl. [Embley 1992: S. 1].

[2] Vgl. [Miller 1978].

[3] [Booch 1995: S.109], [D'Souza 1998: S. 2-78]. Diese Definition findet sich in Variationen in nahezu allen Veröffentlichungen. Vgl. exemplarisch [Englmeier 1997: S. 9] oder [Martin 1993]. Davon abweichend definiert LUTZ, dass alles ein Objekt sein kann, was sich durch Attribute (dort als Komponenten bezeichnet) und Operationen beschreiben lässt. Vgl. [Lutz 1997: S. 9].

[4] Vgl. [Vetter 1995: S. 28 f.].

[5] Vgl. [Booch 1995: S.109 ff.], [Reenskaug 1996: S. 36], [de Champeaux 1993: S. 19], [Züllighoven 1998: S. 24], [Cook 1994: S. 29 f.], [Vetter 1995: S. 26].

[6] Vgl. [Saake 1995: S. 19], [Englmeier 1997: S. 10].

[7] [Kappel 1996: S. VII], [Booch 1995: S. 111].

[8] Die Attribute einzelner Objekte bezeichnet man in der objektorientierten Programmierung als *Instanzvariablen*. Sie besitzen einen Namen und einen Wertebereich, der die möglichen Werte der Instanzvariable festlegt. [Kappel 1996: S. 12]. Davon abweichend bezeichnet man Attribute, deren Werte für alle Objekte einer Klasse als identisch anzusehen sind, als *Klassenvariablen*.

weise statische Struktur der) Attribute[1] des Objekts und die aktuellen (normalerweise dynamischen) Werte dieser Eigenschaften sowie die Beziehungen, die das Objekt zu anderen Objekten besitzt.[2] Dieser Zustand eines Objekts ist veränderlich. Man unterscheidet außerdem den *konkreten* und den *abstrakten* Zustand. Der konkrete Zustand wird durch die Attribute, deren aktuelle Werte und die aktuellen Beziehungen beschrieben.[3] Daneben verwendet man den abstrakten Zustand, der einem bestimmten konkreten Zustand einen Namen gibt.[4]

Zu (b): Der Begriff *Verhalten*[5] drückt umgangssprachlich aus, dass man Objekte verändern kann, ihnen etwas übergeben kann, etwas von ihnen erhalten kann, etwas mit ihnen tun kann oder sie auffordern kann, etwas zu tun bzw. sich zu verändern. Das Verhalten eines Objekts im Sinne des Objektansatzes stellt die Art und Weise dar, wie es (1) in Form von Zustandsveränderungen reagiert und (2) durch Übergabe von Nachrichten an andere Objekte agiert.[6]

Zu (1): Eine Form des Verhaltens eines Objekts stellen Zustandsveränderungen dar. Eine Zustandsveränderung ist allgemein eine Veränderung eines Objekts[7] beziehungsweise die Modifikation des Wertes eines oder mehrerer Attribute oder Beziehungen. Jede Zustandsveränderung eines Objekts wird durch eine Operation – die das Objekt zu diesem Zweck zur Verfügung

[1] Das bedeutet, dass sich der Aufbau der Attribute eines Objekts nicht ändern kann. Hiervon sind die sog. *statischen Attribute* zu unterscheiden, die zum Beispiel in der Programmiersprache C++ als Klassenattribute verwendet werden. Bei Klassenattributen teilen sich alle Objekte der Klasse einen Wert.

[2] Vgl. [Booch 1995: S. 112], [OMG 1992: S. 36]. COLEMAN definiert ein Attribut als ein `<Identifizierer,Wert>`-Paar. Vgl. [Coleman 1992: S. 9]. Ähnlich definiert ENGLMEIER den Zustand eines Objekts als die Abstraktion seiner Attributwerte. Vgl. [Englmeier 1997: S. 34]. MARTIN und ODELL dagegen definieren den Zustand eines Objekts als die Menge der Beziehungen, die ein Objekt mit anderen Objekten besitzt. Vgl. [Martin 1995: S. 87].

[3] Vgl. [Kappel 1996: S. 10, 45], Booch 1995: S. 112 f.], [Henderson-Sellers 2000: S. 95].

[4] Vgl. [Kappel 1996: S. 10, 45].

[5] Verhalten bzw. Verhaltensweise: „*Gesamtheit der möglichen Reaktionen eines dynamischen Systems (Systemverhalten) oder des Elements eines solchen Systems (Verhalten eines Elements) auf äußere Einwirkungen.*" [Klaus 1979: S. 888].

[6] Ähnlich auch in [Booch 1995: S. 115], [Wirfs-Brock 1993: S. 20] zu finden. Etwas problematisch ist die Situation, in der ein Objekt Informationen über seinen Zustand liefern soll. Dadurch verändert sich in der Regel sein Zustand nicht. Dieses Problem löst man durch sogenannte *Selbstübergänge*. Ein Selbstübergang ist ein Übergang in denselben Zustand. Wenn das Objekt Informationen über seinen Zustand liefern soll, erhält man einen Rückgabewert. Das Objekt „geht" dabei in denselben Zustand „über". Das würde der Möglichkeit (1) entsprechen. Man könnte aber auch argumentieren, dass der Rückgabewert einer Nachricht an den Anfrager entspricht. Deshalb kann man diese Situation auch unter die Möglichkeit (2) subsumieren.

[7] Das hat Meyer in folgender Form ausgedrückt: *„Ask not first what the system does: Ask what it does it to!"* [Meyer 1997: S. 116].

stellt – herbeigeführt. In diesem Sinne stellen Operationen Abstraktionen der Zustandsveränderungen der Objekte dar. Sie stehen für die Veränderungen, die ein Objekt vollzieht, bzw. für die Verrichtungen, die man an einem Objekt durchführt. Die Operationen werden als Reaktion auf *Ereignisse*[1] ausgeführt. Dies entspricht in objektorientierten Programmiersprachen dem Aufruf einer Methode.[2] Der Umstand, dass ein Formular ausgefüllt wird und dadurch seinen Zustand von „`nicht ausgefüllt`" zu „`ausgefüllt`" verändert, wird durch eine Operation „`ausfüllen`" des Formulars ausgedrückt.[3] Der Zusammenhang zwischen Zustand und Verhalten besteht darin, dass der aktuelle Zustand des Objekts das kumulierte Ergebnis seines bisherigen Verhaltens darstellt.[4]

Zu (2): Die andere Form des Verhaltens von Objekten besteht darin, dass ein Objekt Aktionen ausführt. Durch Aktionen werden Ereignisse – hier in Form von Nachrichten – für andere Objekte erzeugt, auf die diese mit der Ausführung von Operationen reagieren.[5] Die Reaktion dieser Objekte kann darin bestehen, dass sie Aktionen ausführen oder[6] wiederum andere Objekte auffordern, Operationen auszuführen. Der Aktivierung von Objekten durch Ereignisse setzt sich also fort, bis ein Objekt bei keinem anderen Objekt mehr Verhalten bewirkt. Im Rahmen der Aktionen kann jedes involvierte Objekt seinen Zustand verändern. Der Zusammenhang zwischen Aktionen und Operationen besteht darin, dass die Aktionen des Objekts in der Regel im Rahmen einer Operation stattfinden.[7]

---

[1] Ein Ereignis wird als Zwischenfall definiert, auf den es einer Reaktion bedarf [Hutt 1994: S. 169], bzw. das Eintreten eines zu beachtenden Umstandes. Diese Definition wird im Laufe der Arbeit noch präzisiert.

[2] [Saake 1995: S. 19]. Methoden stellen die Realisierung bzw. Implementierung einer Operation dar. Vgl. [OMG 1992: S. 36].

[3] Der übliche Erklärungsversuch, Operationen würden die Funktionen darstellen, die Objekte ausführen können, ist regelmäßig dann zum Scheitern verurteilt, wenn die Autoren zu erklären versuchen, warum sich ein Rechnungsformular selbst ausfüllen können soll. LUTZ ist ein typisches Beispiel dafür, wie versucht wird, Beobachtungen der Realität und den objektorientierten Ansatz in Einklang zu bringen. Er muss selbst zugeben, dass seine Sichtweise kontraintuitiv ist. Vgl. [Lutz 1997: S. 10].

[4] [Booch 1995: S. 116], [Rumbaugh 1993: S. 107].

[5] Auf das erzeugte Ereignis kann auch das Objekt selbst reagieren, d. h., es fordert sich selbst auf, eine Operation auszuführen.

[6] Damit ist kein exklusives Oder gemeint.

[7] Die Ausnahme von dieser Regel sind so genannte *aktive Objekte*, die ohne äußeren Einfluss aktiv werden. Darauf wird im weiteren Verlauf noch eingegangen.

Zu (c): Durch die Identität lässt sich jedes Objekt von allen anderen Objekten unterscheiden.[1] Es ist eindeutig identifizierbar, selbst wenn alle seine Attributwerte und Beziehungen zu anderen Objekten mit denen eines anderen Objekts übereinstimmen.

2.2.3 Klassen

Der Begriff des Objekts ist untrennbar mit dem der Klasse verbunden, da es keine Objekte ohne Klasse gibt. Danach sind die Struktur und das Verhalten gleichartiger Objekte als ihre Klasse definiert. Objekte sind immer Exemplare eines (Objekt-)Typs, der als Klasse bezeichnet wird.[2] Die Menge der aus einer Klasse gebildeten Objekte bezeichnet man als ihre Extension.[3] Man bezeichnet Klassen auch als Muster, Schablone, Bauplan oder Blaupausen für Objekte. Diese Dichotomie zwischen Klasse und Objekt ist nahezu allen objektorientierten Modellierungsansätzen gemein.

Mehr (programmier-)technisch gesehen, stellen Klassen ein Modell dar, in dem eine Datenstruktur und die darauf zugreifenden Operationen zusammen definiert werden. Die Elemente der Datenstruktur sind die Attribute bzw. die Struktur der Klasse. Klassen sind also eine Menge von Objekten, die eine gleiche Struktur und das gleiche Verhalten aufweisen.[4] Die Klasse ist dabei als eine Art Repository für die Operationen ihrer Extension anzusehen, da alle Objekte den gleichen Satz von Operationen verwenden. Als Folge ist festzuhalten, dass eine Operation eines Objekts auch allen anderen Objekten der Extension zur Verfügung steht. Das Verhalten der Klasse stellt das Muster des Verhaltens ihrer Exemplare (Objekte) dar.[5] Die Werte der Attribute können sich verändern, wogegen sich die Struktur selbst nicht verändern kann, da alle Objekte einer Klasse die gleiche Struktur besitzen.[6]

Durch Klassenbildung bzw. *Klassifikation* erfolgt die Zusammenfassung ähnlicher Dinge (Objekte) zu Klassen oder Kategorien.[7] Durch die Bildung von *Taxonomien* lassen sich darauf aufbauende Klassen in eine Beziehung setzen, die als *Vererbung* bezeichnet wird. Vererbung

---

[1] [Booch 1995: S.115 ff.], [Kappel 1996: S. 10].

[2] Vgl. [Meyer 1997: S. 171], [Budd 1997: S. 13], [Kappel 1996: S. 11], [Partridge 1994: S. 44]. Der Begriff Objekttyp wird häufig als Synonym für den Begriff Klasse verwendet.

[3] [Burkhard 1994: S. 48], [Quibeldey-Cirkel 1994: S. 84]. Hier ist der Sonderfall der Klassen zu betrachten, die keine Extension besitzen. Eine solche Klasse bezeichnet man als *abstrakte Klasse*. Vgl. [BalzertH 1999: S. 533].

[4] [Booch 1995: S. 136], [de Champeaux 1993: S. 26].

[5] Vgl. [Coleman 1992: S. 15].

[6] [Booch 1995: S. 136].

[7] [Taivalsaari 1996: S. 441].

bedeutet, dass die erbende Klasse eine Spezialisierung der vererbenden Klasse darstellt. Die erbenden Klassen erhalten die Struktur und die Operationen der übergeordneten Klasse. Sie können zusätzliche Attribute und Operationen hinzufügen oder ererbte Attribute und Operationen nach bestimmten Regeln verändern.[1]

So wie Objekte Exemplare von Klassen sind, können Klassen selbst als Exemplare einer übergeordneten Abstraktionsebene betrachtet werden. In diesem Fall spricht man von *Metaklassen*[2], deren Exemplare Klassen sind. Dadurch wird die Struktur von Klassen durch Metaklassen dokumentiert.[3] Ein Objekt ist ein Exemplar einer Klasse, die Klasse kann Exemplar einer Metaklasse sein.[4] Auf diese Weise wird der Aufbau von Objekten durch Klassen, Klassen durch Metaklassen und Metaklassen durch Meta-Metaklassen definiert.[5]

2.2.4 Geheimnisprinzip und Kapselung

Die Eigenschaften von Objekten sind gegenüber ihrer Umwelt verborgen. Das bedeutet auch, dass diese Interna von außen nicht verändert werden können. Die Werte der Attribute und die Beziehungen eines Objekts sind von außen weder sichtbar noch veränderbar.[6] Nur das Objekt selbst kann durch seine Operationen seinen Zustand verändern. Wie das durch Operationen ausgedrückte Verhalten den Zustand eines Objekts verändert, ist seine Angelegenheit. Wenn ein Objekt eine Operation „`Erstelle Dokument`“ anbietet, so ist es dem Objekt überlassen, wie dieses Dokument im Einzelnen erstellt wird. Dieses Verbergen der Interna eines Objekts wird als Geheimnisprinzip bzw. *Information Hiding* bezeichnet. Hier ist anzumerken, dass der Begriff Information Hiding kein originärer Begriff des Objektansatzes ist, sondern aus dem modularen Entwurf übernommen wurde.[7]

Durch Information Hiding wird festgelegt, dass die Interna eines Objekts gegenüber seiner Umwelt verborgen sind. Alle Veränderungen des Objekts finden durch Operationen statt. Das *Kapselungsprinzip* steht damit im engen Zusammenhang. Eine Kapsel bildet eine Hülle um eine

---

[1] Auf eine umfassende Definition der Semantik der Vererbung wird noch eingegangen.

[2] Vgl. [Booch 1995: S. 173], [Martin 1999: S. 307], [Alhir 1998: S. 112 ff.].

[3] [de Champeaux 1993: S. 128].

[4] Vgl. [Atkinson 2000: S. 33].

[5] Das Metamodell der UML ist ein Beispiel für einen solchen Aufbau und besteht tatsächlich aus vier Ebenen. Vgl. [OMG 1999: S. 2-4], [Diaz 1999].

[6] Vgl. [Coleman 1992: S. 9], [Quibeldey-Cirkel 1994: S. 220].

[7] [Parnas 1972]. Information Hiding, Kapselung und Abstraktion stehen im engen Zusammenhang miteinander. Diese Zusammenhänge werden in [Berard 1993: S. 63 ff.] dargestellt.

Anzahl von Dingen.[1] Im objektorientierten Ansatz bilden die Interna eines Objekts und die auf diese Interna zugreifenden Operationen eine Kapsel. Nur die Operationen der Klasse können den Inhalt dieser Kapsel verändern. Durch die Kombination von Information Hiding und Kapselung[2] wird die Bildung von Abstraktionen im Sinne von „*blackboxes*" möglich, deren Verhalten man beobachten und beschreiben kann, deren Innenleben und die damit verborgene Komplexität man aber nicht kennen muss. Dies ermöglicht die Beschreibung komplexer Systeme, ohne sich mit der Fülle der Details der einzelnen Elemente des Systems auseinandersetzen zu müssen. Auf diese Weise unterscheidet man zwischen der Schnittstelle und den Interna eines Objekts. Die Gesamtheit der von einem Objekt angebotenen Operationen stellen die von außen sichtbare *Schnittstelle* des Objekts dar. Die Realisierungen der Operationen stellen dagegen die Interna des Objekts dar und sind außerhalb des Objekts nicht sichtbar.[3] Das bedeutet, dass jedes Objekt in Eigenverantwortung entscheidet, wie es seine Aufgaben (Operationen) realisiert. Objekte können ausschließlich durch den Aufruf von Operationen auf andere Objekte zugreifen.[4] Daten gelangen nur durch eine Operation in ein Objekt hinein und Daten gelangen auch nur durch Operationen aus einem Objekt heraus.

2.2.5 Systeme als Objektstrukturen

Wie schon Eingangs erwähnt, können mehrere Objekte zu komplexen Objekten bzw. Objektstrukturen zusammengesetzt werden.[5] Systeme werden im OA als eine geordnete Menge von Objekten interpretiert, zwischen denen Beziehungen bestehen.[6] Die Beziehungen zwischen den

---

[1] Vgl. [Wirfs-Brock 1993].

[2] Vgl. [Kappel 1996: S. 20], [Booch 1995: S. 69 ff.].

[3] [Kappel 1996: S. 10]. Nicht nur die Schnittstellen von Klassen, sondern auch Komponenten, aus denen betriebliche Anwendungssysteme zusammengefügt werden, werden über ihre Schnittstellen definiert (vgl. [Turowski 2001]). Insofern können die hier verwendeten Techniken auch skaliert zur Spezifikation von Komponenten verwendet werden. Manchmal findet sich auch der Begriff *Protokoll*. Dieser wird im objektorientierten Ansatz eher unscharf verwendet, wie [Firesmith 1995: S. 358 ff.] zeigt. Als Protokoll einer Klasse werden hier ihre Schnittstelle und die Bedingungen bezeichnet, unter denen die in der Schnittstelle dokumentierten Operationen verwendet werden können. Diese Definition basiert auf [Züllighoven 1998: S. 28]. Das Protokoll ist also die Dokumentation der Schnittstelle inklusiver ihrer Semantik.

[4] Vgl. [Kappel 1996: S. 123].

[5] Vgl. [Saake 1995: S. 20], [Embley 1992].

[6] Dies deckt sich mit der Systemdefinition der Kybernetik, in der ein System als „[*G*]*eordnete Gesamtheit von materiellen oder geistigen Objekten*" betrachtet wird. [Klaus 1979: S. 800]. „*Danach ist unter einem System eine Menge von Objekten zu verstehen, zwischen denen gewisse Relationen bestehen. ... Die*

Objekten – als die im Zeitablauf relativ konstante Konfiguration der Objekte – repräsentieren die Struktur des Systems. Objekte gelten auf der einen Seite als unteilbare Einheiten, können aber auf einer weiteren Stufe der Zerlegung in Untersysteme (Subsysteme) zerlegt werden. Diese rekursive Definition des System- und des Objektbegriffs bedeutet, dass jedes System aus Objekten besteht und gleichzeitig Objekte als Systeme von Objekten aufgefasst werden können.[1] Kein Objekt existiert in Isolation.[2] Systeme sind damit als Strukturen von Objekten definiert,

- ... die einander *Nachrichten* zusenden,
- ... die auf *Ereignisse* reagieren,
- ... die in *Beziehungen* zueinander stehen,
- ... die von anderen Objekten *Dienste* in Form von Operationen anfordern
- ... und durch die Ausführung von Diensten ihren *Zustand* und den des Systems verändern.

Das allgemeine Verfahren zur Konstituierung von Systemen besteht darin, die Objekte als Elemente oder Teilsysteme mit genau definierten Eigenschaften aufzufassen. Die durch die Gesamtheit von Objekten bestehende Ordnung bildet die Struktur des Systems.[3]

2.2.6 Nachricht und Ereignis

Objekte kommunizieren miteinander durch das Versenden von Nachrichten. Eine Nachricht ist die Spezifikation einer Kommunikation zwischen Objekten, durch die Informationen mit der Absicht übertragen werden, dass Aktivitäten folgen.[4] Zur Bezeichnung des Vorgangs der Nachrichtenübertragung wird im Folgenden der Terminus *Interaktion* bzw. Interaktionsbeziehung verwendet. Das Senden einer Nachricht ist die einzige Möglichkeit, mit einem Objekt zu kommunizieren.[5] Auf diese Weise werden auch Berechnungen durchgeführt[6] und Kontrollstrukturen realisiert.[1] Die

---

Gesamtheit der Relationen eines solchen Systems bezeichnet man als dessen Struktur.“ [Klaus 1979: S. 806].

[1] Vgl. [Embley 1992: S.1]. Dies deckt sich mit der in der Kybernetik üblichen Sichtweise: „*Was z. B. innerhalb eines Systems als Element auftritt, d. h. als nicht weiter aufteilbare Einheit, kann innerhalb eines anderen Systems den Charakter eines komplexen Teilsystems haben.*“ [Klaus 1979: S. 807].

[2] [Booch 1995: S. 115].

[3] Vgl. [Klaus 1979: S. 800].

[4] [Booch 1999: S. 210].

[5] [Kappel 1996: S. VIII].

[6] [Budd 1997: S. 13].

Aufgaben eines Systems werden durch Nachrichtenaustausch zwischen Objekten realisiert.[2] Für die Entwicklung des Modellierungsansatzes ist besonders wichtig, dass ein Prozess aus objektorientierter Sicht eine Abfolge von Objektinteraktionen darstellt.[3]

Der Inhalt einer Nachricht wird als Botschaft bezeichnet. Auch die Übertragung eines Objekts an ein anderes Objekt oder die Übertragung einer Anweisung wird als Nachricht verstanden. Durch den Austausch von Nachrichten wird sowohl der Datenfluss wie auch der *Kontrollstrang*[4] realisiert.[5] Der Datenfluss findet durch die Übertragung von Informationen in Form von Nachrichten statt. Der Kontrollstrang wird realisiert, in dem der Sender die Kontrolle an den Empfänger übergibt und wartet, bis dieser die angeforderte Operation ausgeführt hat.

Durch das Senden einer Nachricht fordert ein Objekt ein oder mehrere Objekte auf, Verhalten zu zeigen. Das Verhalten des Empfängers besteht darin, eine Operation auszuführen.[6] Für den Empfänger der Nachricht stellt das Eintreffen der Nachricht ein *Ereignis* dar, auf das es zu reagieren hat.[7] *„Ein Ereignis tritt zu einem bestimmten Zeitpunkt ein, wird von einem Objekt innerhalb oder außerhalb des Systems ausgelöst und ruft bei einem oder mehreren Objekten eine Reaktion hervor."*[8] Ein Ereignis stellt einen einzelnen Reiz eines Objekts für ein oder mehrere andere Objekte dar.[9] Sie sind atomar und diskret, lassen sich nicht unterbrechen und haben keine Zeitdauer.[10]

Die Sender einer Nachricht erzeugen durch das Senden einer Nachricht Ereignisse. Die Empfänger von Nachrichten reagieren auf diese Ereignisse. Das Versenden einer Nachricht und das

---

[1] Siehe die konsequenteste Umsetzung des Objektansatzes in der Programmiersprache SMALLTALK. Vgl. [Wallrabe 1997: S. 98]. Die Idee, alle Aktivitäten in einem Computerprogramm durch Nachrichten zu realisieren, geht im Wesentlichen auf die von HEWITT entwickelte experimentelle Programmiersprache ACTOR zurück. Vgl. [Hewitt 1977].

[2] Vgl. [Rechenberg 1999: S. 519, 1055].

[3] Vgl. Abschnitt 1.4. Die besondere Bedeutung der Interaktion wurde dort als das *Primat der Interaktionsbeziehung* bezeichnet.

[4] Als Kontrollstrang wird hier die Kontrolle über einen Prozess bezeichnet. Übliche Synonyme sind Faden, Handlungsfaden oder *thread of control*. Vgl. [Rechenberg 1999: S. 635].

[5] Vgl [Rumbaugh 1996b].

[6] Vgl. [Kappel 1996: S. 21].

[7] RUMBAUGH formuliert dies so, dass durch Ereignisse Informationen von einem Objekt zu einem anderen Objekt transportiert werden. Vgl. [Rumbaugh 1993: S. 104].

[8] [Kappel 1996: S. 76]. Vgl. auch [OMG 1992: S. 36]. Der Duden definiert *Ereignis* als besonderer, nicht alltäglicher Vorgang, Vorfall, Geschehnis. [Duden 1993].

[9] Vgl. [Rumbaugh 1994: S. 103].

[10] [Schienmann 1997: S. 259].

Eintreffen eines Ereignisses bilden den gleichen Sachverhalt aus unterschiedlichen Gesichtspunkten ab.[1] Bezogen auf die Sicht der Ereignisse bedeutet *Information Hiding*, dass Objekte Ereignisse erzeugen und auf Ereignisse reagieren. Was in ihrem Inneren passiert, bleibt dem außenstehenden Beobachter verborgen. Das Ereignis stellt einen *Stimulus* für das Objekt dar, welches die Nachricht erhält. Auf diese Weise lassen sich Objekte als Stimulus-Reaktions-Mechanismen verstehen.[2] Man spricht in diesem Zusammenhang auch von *ereignisgesteuerten Systemen*, wenn der Kontrollfluss eines Systems durch Ereignisse kontrolliert wird.[3]

Auch Ereignisse selbst können als Objekt aufgefasst werden.[4] Da durch Ereignisse Operationen ausgelöst werden und diese den Zustand eines Objekts verändern, beeinflussen Ereignisse den Zustand von Objekten.[5] Aus der Sicht der reagierenden Objekte treten die Ereignisse nichtdeterministisch ein. Objekte wissen weder, wann Ereignisse eintreffen noch in welcher Reihenfolge sie eintreffen. Auf diese Weise werden alle Veränderungen in einem objektorientierten System durch Ereignisse ausgelöst. Dies unterscheidet die objektorientierte Sichtweise von der strukturierten, in der nur die Umwelt eines Systems Ereignisse erzeugt.[6] Der objektorientierte Ansatz verwendet hier eine andere Granularität, da jedes Objekt separat betrachtet werden kann. Dadurch ergibt sich, dass Prozesse in objektorientierten Systemen auch Abfolgen von Ereignissen sind. Eine mögliche Abfolge von Ereignissen bezeichnet man als ein *Szenario*.[7]

2.2.7 Polymorphismus

Unter *Polymorphismus* oder *Polymorphie*[8] wird die Eigenschaft verstanden, verschiedene Formen annehmen zu können.[9] Der Begriff wird hier eingeführt, obwohl er in der Modellierung von untergeordneter Bedeutung ist und erst in der objektorientierten Programmierung richtig genutzt

---

[1] Vgl. [Alhir 1998: S. 63], [Lutz 1997: S. 13], [de Champeaux 1993: S. 81].

[2] Vgl. [Cook 1994: S. 17].

[3] [Vlissides 1999: S. 129]. Vgl. auch [Lutz 1997: S. 13]. Alternativ wird auch der Begriff *reaktives System* verwendet. Vgl. [Olsen 1996: S. 22].

[4] Vgl. [Embley 1992: S. 68], [Alhir 1998: S. 63], [Firesmith 1995: S. 162].

[5] MARTIN und ODELL drehen diesen Zusammenhang um. Bei ihnen ist die Zustandsveränderung das Ereignis. Vgl. [Martin 1993: S. 103], [Martin 1995: S. 87].

[6] Vgl. [McMenamin 1988: S. 47 ff.].

[7] SELIC definiert Szenario als eine Serie von kausal verbundenen Ereignissen, die typischerweise mehrere parallel laufende Prozesse (*tasks*) eines Systems umfassen und eine High-Level-Operation darstellen. [Selic 1994: S. 25]. Ähnlich auch in [Rumbaugh 1996b] und [Jarke 1999] zu finden. Vgl. auch den Übersichtsartikel [Linssen 1999b].

[8] Vielgestaltigkeit.

[9] Vgl. [Meyer 1997: S. 467].

wird. Er ist aber deshalb so wichtig, weil Polymorphie eine Ursache der Flexibilität objektorientierter Systeme ist.[1]

Polymorphismus ermöglicht es, dass die Objekte verschiedener Klassen Operationen mit gleichem Namen, aber unterschiedlicher Semantik besitzen können. Das hat zur Folge, dass unterschiedliche Objekte auf den Erhalt der gleichen Nachricht unterschiedlich reagieren. Der Sender einer Nachricht muss nur wissen, dass ein Empfängerobjekt eine Nachricht versteht (das bedeutet in den meisten Programmiersprachen: das Objekt besitzt die entsprechende Operation); er muss aber nicht wissen, zu welcher Klasse das Objekt gehört.[2] In der Regel wird für Polymorphie vorausgesetzt, dass Klassen über eine Vererbungsbeziehung verbunden sind:[3] Die Objekte der erbenden Klasse sind gleichzeitig Objekte der vererbenden Klasse; sie erben deren Verhalten und damit die Fähigkeit, auf eine Nachricht mit der Ausführung einer Operation reagieren zu können. Man bezeichnet die vererbende Klasse auch als *polymorphe Entität*.[4]

Die tatsächlich verwendete Operation wird erst zum Zeitpunkt des Aufrufs bestimmt. Im engen Zusammenhang zum Polymorphismus stehen die Konzepte des dynamischen oder späten Bindens (*dynamic* oder *late binding*) und des Überladens (*overloading*) von Operationen.

Spätes bzw. dynamisches Binden bedeutet, dass erst in dem Moment, in dem die Ausführung der Operation von einem anderen Objekt angefordert wird, darüber entschieden wird, welche Operation ausgeführt wird.[5]

Das Überladen von Operationen bedeutet, dass eine Klasse unterschiedliche Operationen mit dem gleichem Namen besitzt.[6] Die Operationen werden durch die Anzahl und den Typ ihrer Argumente unterschieden.[7] Der Unterschied zwischen Polymorphismus und überladenen Operationen besteht darin, dass beim Polymorphismus Objekte unterschiedlicher Klassen Operationen mit gleichem

---

[1] Kaum ein Konzept der Objektorientierung sorgt für mehr Verwirrung als das des Polymorphismus. In [BalzertH 1999: S. 256 ff.] findet sich eine gute Darstellung dieses Konzepts. Auf Details wird hier nicht eingegangen, da sie nur auf der Ebene von Programmiersprachen und Objekttypen verständlich wären. Vgl. hierzu die umfangreiche Darstellung in [Meyer 1997] sowie [Firesmith1995: S. 332].

[2] [BalzertH 1999: S. 256]. Vgl. auch [Booch 1995], [Wirfs-Brock 1993], [Blair 1989].

[3] Dies ist eine Regelung typisierter Programmiersprachen. [Rechenberg 1999: S: 535]. Man spricht dann von der *eingeschränkten Polymorphie*. Dynamisch typisierte Programmiersprachen wie SMALLTALK lassen eine *uneingeschränkte Polymorphie* zu. Vgl. [Züllighoven 1998: S. 27, 41 ff.].

[4] Vgl. [Meyer 1997: S. 469].

[5] Vgl. [Pomberger 1993: S. 191]. Irrtümlicherweise werden die Konzepte des späten Bindens mit Polymorphie häufig gleichgesetzt. Tatsächlich sind diese Konzepte aber zu trennen, da späte Bindung auch ohne Polymorphie auftritt. Vgl. [Pratt 1997], [Louden 1994], [Budd 1997].

[6] Vgl. [Kappel 1996: S. 21 f.].

[7] Vgl. [Meyer 1997: S. 93].

Namen besitzen können. Eine überladene Operation liegt dagegen vor, wenn ein Objekt mehrere Operationen mit dem gleichen Namen, aber unterschiedlichen Argumentsignaturen besitzt.

2.2.8 Dienstleistungen und Vertragsmodell

Die in der Schnittstelle einer Klasse definierten Operationen werden als Leistungen, Dienste[1] bzw. *Dienstleistungen*[2] angesehen, die die Objekte der Klasse anderen Objekten zur Verfügung stellen. Nur durch die Dienste kann ein Objekt den Zustand eines anderen Objekts ändern. Das die Dienstleistung zur Verfügung stellende Objekt wird als Anbieter, Lieferant oder Dienstleister bezeichnet, das die Dienstleistung anfordernde Objekt wird als Kunde oder Klient[3] bezeichnet. Dieses Dienstleistungs- oder *Client/Supplier-Prinzip* hat sich in den letzten Jahren zu einer verbreiteten Metapher für den Entwurf objektorientierter Systeme entwickelt.[4]

Fordert ein Objekt im Rahmen einer Interaktion von einem anderen Objekt eine Dienstleistung an, so stellt es bestimmte Anforderungen an dieses Objekt. Das Objekt, welches die Aufgabe ausführen soll, geht auf der anderen Seite von bestimmten Erwartungen aus, damit es die Aufgaben ausführen kann. Diese Erwartungen werden als Rechte bezeichnet, das zu erbringende Ergebnis als Pflichten des Lieferanten. Dieses Verhältnis von Rechten und Pflichten zwischen Objekten bezeichnet man als *Vertrag*.[5] Der Vertrag spezifiziert die Bedingungen, unter denen ein Objekt eine Operation ausführt. Eine Dienstleistung, für die ein Objekt zuständig ist, wird in der objektorientierten Literatur auch als *Verantwortlichkeit* des Objekts bezeichnet. *„Verantwortlichkeiten sind Dienstleistungen, die ein Objekt aufgrund seiner Verträge bewerkstelligen kann. [...] Eine*

---

[1] Vgl. [Quibeldey-Cirkel 1994: S. 220].

[2] Diese Arbeit setzt sich nicht mit dem Begriff der Dienstleistung in der Betriebswirtschaftslehre auseinander. Charakteristische Merkmale einer Dienstleistung sind nach CORSTEN ihre Immaterialität und der unmittelbare Kontakt zwischen Anbieter und Nachfrager. Dabei spielt es keine Rolle, ob die Dienstleistung ein materielles Trägermedium besitzt. Vgl. [Corsten 1990: S. 19, 23], [Striening 1988: S. 43]. Im Unterschied zur betriebswirtschaftlichen Definition von CORSTEN (vgl. Glossareintrag) wird im OA nicht davon ausgegangen, dass die Dienstleistung für einen externen Kunden erbracht wird. Vgl. [Corsten 1990: S. 23].

[3] [Züllighoven 1998: S. 26], [Quibeldey-Cirkel 1994: S. 100 ff.].

[4] Vgl. [Waldén 1995], [Wirfs-Brock 1993], [Wilkinson 1995], [Züllighoven 1998], [Quibeldey-Cirkel 1994: S. 99 ff.], [Rumbaugh 1993: S. 243], [Booch 1995: S. 227], [Firesmith 1995: S. 91], [Englmeier 1997: S. 21].

[5] [Quibeldey-Cirkel 1994: S. 102 ff.], [Henderson-Sellers 2000: S. 86].

Dienstleistung kann entweder die Ausführung einer Aktion oder die Lieferung von Informationen sein."[1]

Verträge regeln die im Rahmen der Aufteilung von Verantwortlichkeiten an Objekte delegierten Aufgaben. Der Vertrag legt die Bedingungen fest, unter denen der Lieferant eine Leistung für einen Kunden erbringt. Der Kunde darf nur Operationen anfordern, wenn er seine Vertragsbestandteile einhält. Erfüllt der Kunde seinen Vertrag, muss der Lieferant die angeforderte Operation erfüllen. Der Vertrag zwischen Kunde und Lieferant spezifiziert also nicht, wie eine Operation ausgeführt wird, sondern unter welchen Bedingungen eine Leistung erbracht wird. Objekte organisieren also in gewisser Weise selbständig ihre Arbeit, denn jedes Objekt entscheidet in eigener Verantwortung, wie es auf eine eintreffende Nachricht reagiert und wie es eine geforderte Aufgabe erfüllt.

Als *Delegation* wird bezeichnet, wenn ein Objekt im Rahmen der Erfüllung seiner Aufgabe andere Objekte mit Teilaufgaben beauftragt.[2] Der zugehörige Vertrag, der Rechte und Pflichten festlegt, heißt *Unterauftrag*.

Die Verträge werden durch logische Prädikate ausgedrückt, die die Bedingungen für die durch Operationen ausgedrückten Verantwortlichkeiten darstellen. Die Bedingungen müssen vor und nach der Ausführung einer Operation erfüllt sein.[3] Kern dieses Prinzips ist das Konzept der *Invarianz*, welches bei BOOCH und WALDÉN und unter dem Begriff *Design-by-Contract*[4] bei MEYER zu finden ist.[5] Invariante Bedingungen finden sich im objektorientierten Ansatz in drei Ausprägungen: Invarianten, Vor- und Nachbedingungen.[6]

„Eine Invariante ist eine boolesche Bedingung (True oder False), deren Zutreffen nicht verletzt werden darf. Für jede Operation, die einem Objekt zugeordnet ist, können wir Vorbedingungen definieren (Invarianten, die von der Operation vorausgesetzt werden) sowie Nachbedingungen (Invarianten, die von der Operation erfüllt werden sollen). Die Verletzung einer Invariante bricht die Vereinbarung, die mit einer Abstraktion verbunden ist. Wenn eine Vorbedingung verletzt wird, bedeutet das, dass ein Client seinen Teil des Abkommens nicht erfüllt ha, und dass der Server nicht korrekt weiterarbeiten kann. Analog dazu hat bei der Verletzung einer Nachbedingung ein Server

---

[1] [Wirfs-Brock 1993: S. 63].

[2] Vgl. [Firesmith 1995: S. 126]. In der Organisationslehre ist Delegation die dauerhafte Übertragung von Entscheidungsaufgaben sowie der zugehörigen Kompetenzen und Verantwortung an hierarchisch nachgeordnete Stellen. [Schulte-Zurhausen 1999: S. 192].

[3] [Kappel 1996: S. 13].

[4] Das Prinzip wird noch näher erläutert.

[5] [Booch 1995], [Waldén 1995], [Meyer 1997].

[6] Invariante Bedingungen werden von Henderson-Sellers als *Assertions* (*Zusicherungen*) bezeichnet. Vgl. [Henderson-Sellers 2000: S. 85].

seinen Teil des Abkommens nicht erfüllt, und seine Clients können sich auf das Verhalten des Servers nicht verlassen.“[1]

Die Vor- und Nachbedingungen einer angeforderten Operation legen fest, was eine Operation vor ihrer Ausführung erwartet bzw. was sie nach ihrer erfolgreichen Ausführung sicherstellt.[2] Vor- und Nachbedingungen beschreiben Operationen isoliert von anderen Operationen des Objekttyps. Hierbei ist zu beachten, dass Operationen teilweise nur in einer bestimmten Reihenfolge aufgerufen werden können. Diese Ausführungsfolgen werden als gültiger Lebenszyklus der Klasse bezeichnet.[3]

Invarianten sind im Gegensatz zu Vor- und Nachbedingungen logische Prädikate, die – unabhängig von der Ausführung einer Operation – Aussagen über den Zusammenhang zwischen Attributen beinhalten.[4] Scheitert die Erbringung einer Leistung, bezeichnet man dies als Ausnahme bzw. als *Exception*:[5] *„Eine* Ausnahme *ist ein Anzeichen dafür, dass eine Invariante nicht erfüllt wurde oder erfüllt werden kann.“*[6] Das Objekt, dessen Operation scheitert, erzeugt die Ausnahme. Ausnahmen werden in der Regel als besondere Objekte interpretiert.[7]

2.3 Projektionen und Ebenen in der Modellierung

2.3.1 Differenzierung von Projektionen und Ebenen

Bei der Modellierung von Systemen werden Projektionen[8] im Sinne isolierender Abstraktionen verwendet. Dabei wird versucht, möglichst orthogonale Sichtweisen zu verwenden, um die Anzahl der Querbezüge zu anderen Projektionen möglichst gering zu halten. In der Literatur finden sich eine Vielzahl möglicher Sichtweisen, die bei der Modellierung von Systemen verwendet werden können.[9] Im Software Engineering werden üblicherweise die Projektionen

---

[1] [Booch 1995: S.63]. Hervorhebung im Original.

[2] [Kappel 1996: S. 43].

[3] [Kappel 1996: S. 45]. Vgl. auch [Martin 1995: S. 90]. Der Begriff Lebenszyklus geht im Objektansatz auf SHLAER und MELLOR zurück. Vgl. [Shlaer 1992: S. 33].

[4] Teilweise ist auch von *Guards* oder Zusicherungen die Rede. Vgl. [Frank 1998b: S. 28].

[5] Vgl. [de Champeaux 1993: S. 70], [Frank 1998b: S. 28].

[6] [Booch 1995: S.63]. Vgl. [Oestereich 1998: S. 241].

[7] [Embley 1992: S. 82].

[8] Projektion: „*Betrachtung eines Systems unter verschiedenen Gesichtspunkten, jeweils hinsichtlich einer Teilmenge seiner Eigenschaften.*“ [Partsch 1998: S. 43]. Vgl. auch [Davis 1993: S. 51].

[9] Vgl. [Partsch 1998: S. 43].

- System- und Komponentenstrukturen,
- Abläufe und funktionales Verhalten,
- Kontrollaspekte und Steuerung,

verwendet,[1] da auf diese Weise die Informationen gesammelt werden, die zur Realisierung eines Softwaresystems notwendig sind. Die *Strukturen* beschreiben die Elemente des Systems und die Beziehungen zwischen diesen Elementen im Sinne der zeitinvarianten Anordnung der Elemente. Die *Abläufe und funktionales Verhalten* beschreiben den Aufbau funktionaler Abstraktionen im Sinne *„transformationsbezogener zeitinvarianter Verrichtungen der einzelnen Bausteine und des ganzen Anwendungssystems“*.[2] Die *Steuerung und Kontrolle* beschreibt schließlich die zeitlichen Veränderungen im Sinne kausaler Abhängigkeiten von Ereignissen, Zuständen und den daraus resultierenden Aktivitäten. Auch in der Praxis sind diese drei Sichtweisen in vielen Methoden wie MSA[3], OMT[4] oder ARIS[5] mehr oder weniger ausgeprägt vorzufinden. Zur Modellierung der Projektionen finden sich hierfür in der Literatur eine Vielzahl von Techniken.[6]

Neben diesen Projektionen werden drei unterschiedliche Betrachtungs*ebenen* verwendet. Die *Systemebene* betrachtet ein System als Ganzes und stellt die Außensicht (*blackbox*) auf ein System dar. Die *Elementebene* betrachtet die in einem System vorzufindenden Strukturen, Funktionen und Zustandsveränderungen und stellt die Innensicht (*whitebox*[7]) eines Systems dar. Die *Intra-*

[1] [Partsch 1998: S. 44, 54, 58, 81, 97]. Ähnlich auch in [Davis 1993: S. XXI] und [Ferstl 1998: S. 16] zu finden. Man beachte, dass in der Wirtschaftsinformatik teilweise eine andere Unterscheidung vorgenommen wird, die sich nicht konsequent an dieses Projektionen orientiert. Dort wird zwischen funktionaler Zerlegung, dem Datenflussansatz, der Datenmodellierung, dem objektorientierten Ansatz und dem geschäftsprozessorientierten Ansatz unterschieden. [Rechenberg 1999: S. 1054 f.]

[2] [Schienmann 1997: S. 51].

[3] Vgl. [Yourdon 1992].

[4] Vgl. [Rumbaugh 1993].

[5] Vgl. [Scheer 1998], [Scheer 1998b].

[6] Übersichten hierzu finden sich in [Schönthaler 1992], [Davis 1993], [Partsch 1998] und [V-Modell 1997]. Eine umfassende Darstellung der in der Softwareentwicklung üblichen Techniken – mit Ausnahme der formalen Spezifikation – findet sich in [Balzert 1996].

[7] Da *blackbox* nach der neuen deutschen Rechtschreibung in einem Wort geschrieben wird, wird hier auch *whitebox* in einem Wort geschrieben.

Elementebene betrachtet die Struktur, Funktion und Zustandsveränderungen eines einzelnen Elements und stellt somit die Sicht auf das Innenleben einzelner „Moleküle“ des Systems dar.[1]

2.3.2 Anwendung in der objektorientierten Modellierung

Die Projektionen Strukturen, Abläufe und Funktionen sowie Kontrolle und Steuerung werden im objektorientierten Ansatz durch unterschiedliche Darstellungsformen abgedeckt. Der wesentliche Unterschied des Objektansatzes zu anderen Ansätzen besteht darin, dass keine isolierte Repräsentation von Datenstrukturen und Funktionen stattfindet. Durch das Kapselungsprinzip sind in den Klassen die Datenstruktur und die Funktionen vereint. Die Kontrollaspekte werden durch Veränderungen des Zustands einzelner Klassen abgebildet. Die Untersuchung eines Systems bezieht sich also immer auf den Betrachtungsgegenstand Objekt. Dadurch werden typische Modellierungsfehler vermieden, die bei der isolierten Erstellung von Daten-, Funktions- und Steuerungsmodellen entstehen. Es ist beispielsweise nicht möglich, eine Funktionalität zu untersuchen, ohne dass diese Funktionalität irgendeinem Element des Systems zugeordnet ist. Darüber hinaus entfallen die aufwändigen Abgleichungsregeln, die in strukturierten Methoden zur Konsistenzerhaltung zwischen einzelnen Modellen notwendig sind.[2]

Aufgrund der gemeinsamen Betrachtung von Datenstruktur und Funktion bezeichnet PARTSCH die objektorientierten Verfahren als kombinierte Verfahren.[3] Die gemeinsame Betrachtung hat allerdings zur Folge, dass die idealtypische Trennung in drei Projektionen auf den OA nur bedingt übertragbar ist. Im objektorientierten Ansatz wird die System- und Komponentenstruktur in der sogenannten *statischen Sicht* abgebildet. *Partitionen*[4] werden gebildet, indem das System in seine Elemente (Objekte bzw. Klassen) zerlegt wird. Abstraktionen werden gebildet, indem Vererbungsbeziehungen zwischen generellen und spezielleren Elementen gebildet werden.

In der *dynamischen* oder *Verhaltenssicht* werden gemeinsam funktionales Verhalten und Kontrollaspekte betrachtet.[5] Kontrollaspekte werden durch Zustände und Zustandsübergänge

---

[1] Vgl. hierzu den Ansatz von SCHIENMANN, der zwei Ebenen (Intra-Objektsicht und Inter-Objektsicht) unterscheidet. Eine Ebene, auf der ein System als Ganzes angesehen wird, betrachtet er nicht. [Schienmann 1997: S. 53 ff.].

[2] Vgl. [Yourdon 1992], [Raasch 1993: S. 111].

[3] Vgl. [Partsch 1998: S. 130].

[4] Partition: „*Zerlegung in einzelne Teile und sukzessive Konzentration auf einzelne individuelle Komponenten und Teilsysteme.*“ [Partsch 1998: S. 43]. Verbreiteter ist im Software Engineering allerdings der Begriff *Dekomposition* (s.u.). Zu den Begriffen Partition und Abstraktion siehe auch [Davis 1993: S. 48 ff.].

[5] Vgl. [Martin 1995: S. 10], [Cook 1994], [Booch 1995: S. 222], [Booch 1999], [Behringer 1997: S. 25 ff.]. Offensichtlich stellt ein Geschäftsprozessmodell im objektorientierten Ansatz ein dynamisches Modell dar.

abgebildet. Partitionen werden gebildet, indem Zustände weiter in Subzustände zerlegt werden. Funktionale Aspekte werden durch die Operationen der Klassen in der statischen Sicht betrachtet. Abläufe werden in Form von Interaktionen zwischen Objekten und Klassen dargestellt.

Im OA findet bei der Modellierung funktionaler Aspekte weder die Bildung von Partitionen noch die von Abstraktionen statt. Dies stellt einen wesentlichen Unterschied zwischen den objektorientierten und den strukturierten Ansätzen dar. Während in den strukturierten Ansätzen[1] die Modellierung auf der *Dekomposition*[2] von Funktionen basiert, findet in den objektorientierten Ansätzen die Dekomposition von Objekten statt. Die Ursache hierfür besteht darin, dass aus objektorientierter Sicht komplexe Systeme keine übergeordnete („Top-Level-“)Funktion besitzen, die sich zerlegen lässt.[3] Allgemein wird die Ansicht vertreten, dass eine separate Modellierung funktionalen Verhaltens und insbesondere die Bildung von Partitionen (funktionale Dekomposition) nicht mit dem OA in Einklang zu bringen ist.[4] Die Betrachtungsebenen finden im OA in der Form Anwendung, dass entweder

- ein einzelnes Objekt bzw. eine einzelne Klasse (Intra-Elementebene) oder
- eine Gruppe bzw. eine Struktur von Objekten und Klassen (Elementebene) betrachtet wird.

Die Struktur einzelner Elemente sind die Attribute der Klassen. Die Struktur einer Gruppe sind die Beziehungen zwischen den Klassen. Das funktionale Verhalten einzelner Elemente sind die Operationen der Objekte einer Klasse. Funktionales Verhalten, welches aus mehr als einer Operation eines Objekts besteht, wird durch Abläufe in Form von Interaktionssequenzen zwischen Objekten abgebildet. Kontrollaspekte einzelner Elemente werden durch die möglichen Zustandsübergänge der Objekte einer Klasse abgebildet. Diese Zustandsübergänge werden durch ein Ereignis initiiert, welches in der folgenden Tabelle in der mittleren Spalte aufgeführt ist. Für ein Objekt Brief könnten dies sein:

---

[1] Exemplarisch ist hier SADT [Ross 1977], [Ross 1977b], [IDEF0 1993] oder MSA [Yourdon 1992] zu nennen.

[2] *Dekomposition* ist der im Software Engineering übliche Begriff für eine systematische Aufgliederung (in elementare Komponenten). Vgl. [Duden 1993]. Üblicherweise ist damit eine Dekomposition von Funktionen gemeint. Bei PARTSCH würde dies als Partition(-sbildung) bezeichnet (s. o.).

[3] Vgl. [Meyer 1997: S. 103].

[4] Hierbei ist anzumerken, dass eine Reihe von Methoden die funktionale Modellierung unterstützen. Die OMT [Rumbaugh 1993] beinhaltet ein funktionales Modell und ist deshalb in der Vergangenheit stark kritisiert worden.

Zustand	Ereignis	Folgezustand
(kein Brief)[1]	erzeugen	‚Brief erzeugt'
‚Brief kuvertiert'	öffnen	‚Brief offen'
‚Brief unfrankiert'	frankieren	‚Brief frankiert'
‚Brief versandfertig'	versenden	‚Brief versendet'
‚Brief offen'	Ablage zuordnen	‚Brief zugeordnet'
‚Brief offen'	schließen	‚Brief kuvertiert'
jeder Zustand außer (kein Brief)	vernichten	‚Brief vernichtet'
‚Brief zugeordnet'	Brief an Empfänger übergeben	‚Brief übergeben'

Tabelle 1: Zustandsübergänge in einer tabellarischen Notation[2]

Man vermeidet üblicherweise in objektorientierten Ansätzen aus Gründen der Komplexitätsreduktion, Kontrollaspekte einer Gruppe von Objekten abzubilden. Wenn es notwendig ist, wird die gegenseitige Beeinflussung von Zuständen durch Interaktionen abgebildet, auf die die beteiligten Objekte durch Zustandsübergänge reagieren. Das bedeutet, dass auch eine Betrachtung von Kontrollaspekten mehrerer Objekte immer auf die Betrachtung der Kontrollaspekte einzelner Objekte „heruntergebrochen" wird. Die Betrachtung von Kontrollaspekten auf der Systemebene findet – wenn überhaupt als sinnvoll angesehen – dadurch statt, dass das System wie ein einzelnes Objekt betrachtet wird. Das funktionale Verhalten eines Systems wird durch ein Diagramm abgebildet, durch welches ersichtlich wird, welche Funktionen ein System erfüllt. Die Struktur stellt auf dieser Betrachtungsebene die Interaktion eines Systems mit anderen Systemen dar.

[1] Natürlich kann ein Brief nicht den Zustand besitzen, dass kein Brief existiert. Hierbei handelt es sich um einen sogenannten *Initialzustand*, d. h. den Zustand, den ein Objekt unmittelbar zum Zeitpunkt seiner Entstehung besitzt. Vgl. [Rumbaugh 1999: S. 76, 300].

[2] Bei dieser Darstellung handelt es sich um eine abgewandelte Form der von BALZERT verwendeten Zustandstabelle. Vgl. [Balzert 1996: S. 274 ff.].

2.4 Repräsentation des Objektansatzes in der UML

Bei allen bestehenden Unterschieden in Terminologie, Notation und Anwendung ist festzustellen, dass die Majorität aller Modellierungsansätze eine nichtleere Schnittmenge gemeinsamer Eigenschaften und Konzepte besitzt. In nahezu allen Ansätzen wird das Ergebnis der Anforderungsanalyse in Form von Klassen dokumentiert, die miteinander durch unterschiedliche Beziehungstypen verbunden sind. Klassendiagramme werden zur Darstellung der Struktur der Klassen und des durch sie konstituierten Systems verwendet. Die einzelnen Klassen werden durch ihre Eigenschaften und die von ihnen ausgeführten Aktivitäten dokumentiert. Die Darstellung der Ergebnisse der Modellierung erfolgt in der Regel graphisch.

In den Details finden sich jedoch viele Unterschiede in den verwendeten Diagrammen der einzelnen Modellierungsansätze. So verwendet die OMT und die OSA zur Abbildung von Abläufen die Datenflussdiagramme der strukturierten Analyse.[1] In der Methode OOA werden hierfür Flussdiagramme verwendet,[2] MARTIN und ODELL verwenden Ereignisdiagramme.[3] Im OOSE werden die *Stimulus-Response-Diagramme* der SDL verwendet, um Kontrollaspekte darzustellen.[4] Eine allgemein gültige Zuordnung, welche Diagramme zur Abbildung der einzelnen Projektionen und Ebenen in der objektorientierten Modellierung verwendet werden, ist deshalb kaum möglich. Aus diesem Grund wird die UML verwendet, da sie als umfassender Industriestandard Referenzcharakter besitzt.[5] Im Standard der OMG sind neun unterschiedliche Diagrammtypen definiert, die in den folgenden Abschnitten kurz vorgestellt werden.[6]

---

[1] Vgl. [Rumbaugh 1993], [Shlaer 1992].

[2] Vgl. [Coad 1991].

[3] Vgl. [Martin 1999], [Martin 1995].

[4] Vgl. [Jacobson 1994], [Jacobson 2000: S. xvii]. Die *Stimulus-Response-Diagramme* der SDL sind erweiterte Zustandsdiagramme, die zusätzlich Entscheidungs-, Eingabe- und Ausgabeknoten beinhalten. Sie sind in [Olsen 1996] beschrieben. Eine kurze Übersicht findet sich in [Partsch 1998: S. 122]. Sie werden mittlerweile auch als objektorientierte Technik verwendet. Vgl. [Ellsberger 1997]. Bestimmte Elemente der Stimulus-Response-Diagramme sind schon in die UML übernommen worden. Vgl. [OMG 1999: S. 3-149].

[5] Eine umfassende Darstellung der UML erfolgt hier nicht, da hierzu schon Veröffentlichungen in großer Zahl vorliegen. Vgl. [Booch 1999], [Jacobson 1999], [Rumbaugh 1999], [Hitz 1999]. Im Rahmen der Vorarbeiten zu dieser Arbeit befassten sich [Linssen 1999] und [Linssen 1999f] mit der UML.

[6] Vgl. [OMG 1999: S. 1-3]. Das Paketdiagramm dient nicht der Modellierung, sondern zur Organisation von Modellen (vgl. [OMG 1999: S. 2-161]) und wird aus diesem Grund hier nicht aufgeführt. Eine der wenigen Veröffentlichungen, die sich mit diesem Thema auseinandersetzt, ist [Kocher 1998].

2.4.1 Klassendiagramm

Das *Klassendiagramm*[1] ist das zentrale Element der objektorientierten Modellierung und stellt den Kern der statischen Modellierung dar. Es dokumentiert die Struktur einzelner Klassen und aus Klassen bestehende Strukturen. Außerdem wird den Klassen durch Operationen funktionales Verhalten zugeordnet. Das Klassendiagramm ist von zentraler Bedeutung im Rahmen des Softwareentwicklungsprozesses, weil aus ihm durch entsprechende Generatoren automatisch Programmcode erzeugt werden kann.[2]

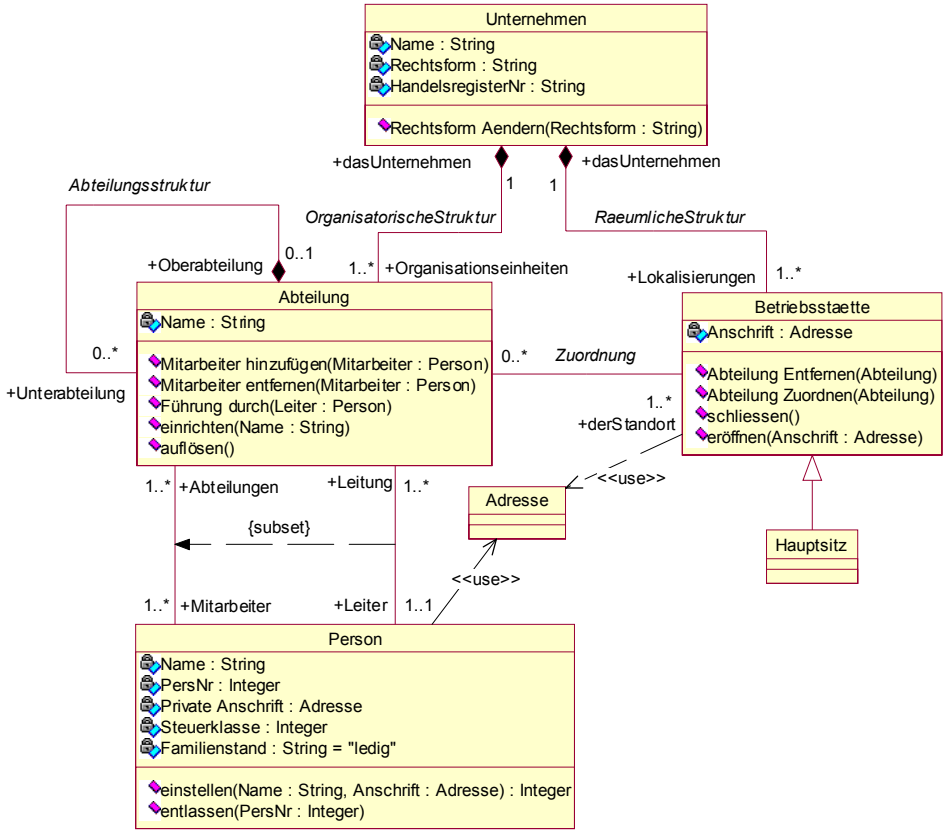


Abbildung 1: Klassendiagramm der UML

[1] Die Bezeichnung *Klassendiagramm* ist üblich, auch wenn es eigentlich *statisches Strukturdiagramm* heißt.

[2] Bei den übrigen Diagrammen ist diese Umsetzung aktuell noch Gegenstand der Forschung. Auf die Bedeutung von Generatoren im Softwareentwicklungsprozess wird hier nicht im Einzelnen eingegangen. Der interessierte Leser sei auf [Linssen 1996] verwiesen.

Die Darstellung des Klassendiagramms ähnelt der semantischen Datenmodellierung.[1] Zu beachten ist aber, dass hier nicht nur eine Datenstruktur dokumentiert wird, sondern die Struktur eines Systems inklusive der auf die einzelnen Klassen des Systems verteilten Funktionalität. Darüber werden durch die Klassendiagramme Aspekte wie Klassifikationen, referentielle Integrität und das „Voneinander-Wissen“ von Konzepten abgebildet. Die Konzepte zur statischen Modellierung können insoweit als ausgereift und konsolidiert angesehen werden, als hier in den letzten Jahren keine nennenswerte Weiterentwicklung stattgefunden hat.

2.4.2 Objektdiagramm

Ein Objektdiagramm stellt die Struktur eines Systems oder eines einzelnen Objekts zu einem bestimmten Zeitpunkt dar. Es besteht aus Objekten und Objektverbindungen (sog. *Links*). Die aktuellen Werte von Attributen können abgebildet werden.

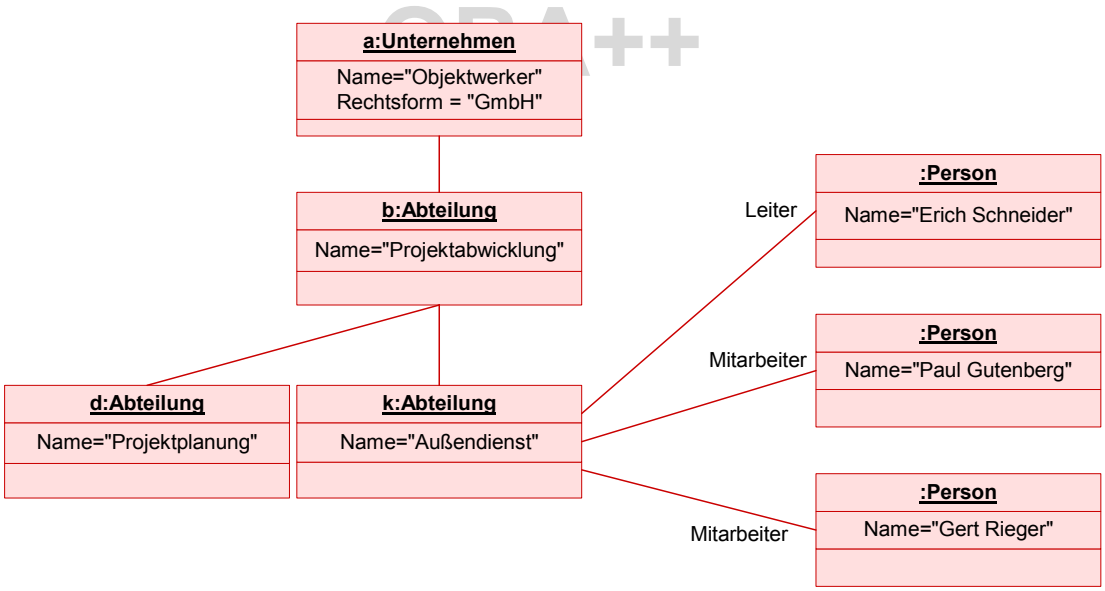


Abbildung 2: Objektdiagramm der UML

Der Unterschied zwischen Klassen- und Objektdiagramm besteht darin, dass das Objektdiagramm die Struktur des Systems zu einem bestimmten Zeitpunkt anzeigt[2], während das Klassendiagramm die möglichen Konfigurationen des Systems anzeigt. Das Objektdiagramm verwendet die gleichen

[1] [Embley 1992: S. 56], [Peckham 1988].

[2] Dies wird als *Konfiguration* bezeichnet.

Symbole wie das Klassendiagramm. An Stelle der Klassenbezeichnungen werden Klassen- und Objektnamen verwendet. Die aktuellen Werte von Attributen und die aktuellen Beziehungen zwischen Objekten werden angezeigt. Die Operationen von Objekten werden hier nicht abgebildet, da sie zur Klasse und nicht zu den Objekten gehören.

2.4.3 Anwendungsfalldiagramm

Das *Anwendungsfalldiagramm* (*use case diagram*) dokumentiert – unter Verbergung interner Details – die Funktionalität eines Systems, einer Gruppe von Objekten oder eines einzelnen Objekts sowie die Interaktionen zwischen dem betrachteten Gegenstand und seiner Umwelt.[1] Für ein fiktives Finanzbuchhaltungssystem könnte dies – stark vereinfacht – wie folgt aussehen:[2]

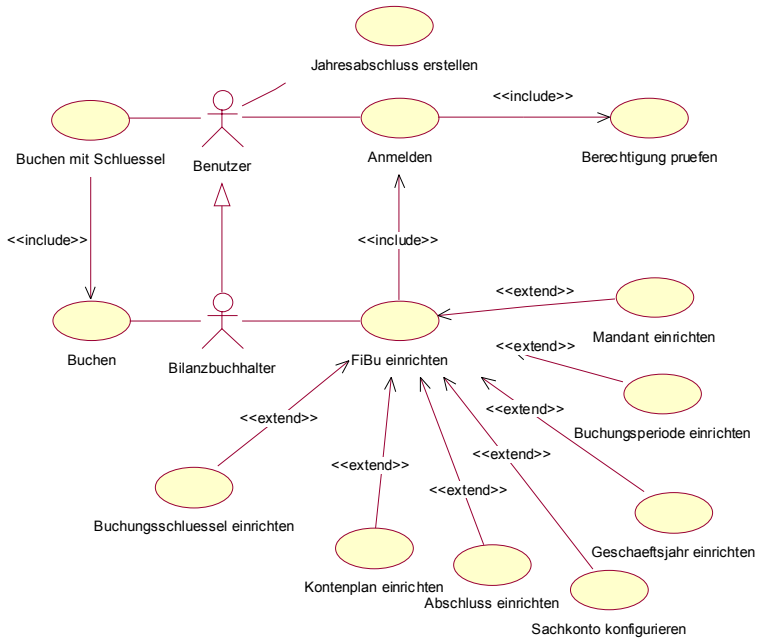


Abbildung 3: Anwendungsfalldiagramm der UML

[1] Die detaillierteste Darstellung über die Konzepte und Prinzipien aus der Feder ihres geistigen Vaters findet sich in [Jacobson 1999]. Diese Darstellung verdeutlicht – im Gegensatz zu [Jacobson 1994], der am häufigsten angegebenen Quelle – wesentlich besser die Konzepte und Prinzipien, die sich hinter Anwendungsfällen (*Use Cases*) verbergen.

[2] Die in Anwendungsfalldiagrammen verwendeten Beziehungen sind wie folgt zu interpretieren: die Beziehung «extend» bedeutet, dass der Anwendungsfall am Ausgangspunkt der Beziehung die Funktio-

Anwendungsfalldiagramme sind keine spezifisch objektorientierte Technik. Das Anwendungsfalldiagramm stellt auf höchster Abstraktionsebene die Funktionalität dar, die ein betrachteter Gegenstand (System, Teilsystem oder Objekt) einem Benutzer bzw. seiner Umwelt (den Akteuren[1]) zur Verfügung stellt (hier: Buchen, Kontenplan einrichten, ...). Dabei wird der Gegenstand als Blackbox angesehen. Alle Interna der Realisierung der Funktionalität sind verborgen. Darüber hinaus werden Interaktionsbeziehungen des Gegenstands mit externen Systemen und Benutzern dokumentiert (hier: Benutzer, Bilanzbuchhalter). Der betrachtete Gegenstand reagiert auf die vom Akteur erzeugten Ereignisse. Wesentlich wichtiger als diese globale Sicht ist die *Anwendungsfallbeschreibung*, in der – in der Regel formlos – zu den einzelnen Anwendungsfällen die Einzelheiten des Verlaufs dokumentiert werden.[2]

2.4.4 Kollaborationsdiagramm

In der UML werden zwei Formen von so genannten *Interaktionsdiagrammen* verwendet, Kollaborations- und Sequenzdiagramme.[3] Eine *Kollaboration* ist in der UML ein Ausschnitt aus der statischen Modellstruktur, die genau jene Elemente enthält, die zur Erreichung eines Ziels kooperieren.[4] Das *Kollaborationsdiagramm* dokumentiert Abläufe und funktionales Verhalten durch Interaktionen mehrerer Objekte. Es kann als Objektdiagramm angesehen werden, in dem der Nachrichtenaustausch hinzugefügt wird und dafür die Attributwerte weggelassen werden. Auch die Spezifikation des funktionalen Verhaltens einzelner Operationen wird durch Kollaborationsdiagramme dokumentiert.

---

nalität des anderen Anwendungsfalls optional erweitert. Die Beziehung «include» bedeutet, dass der Anwendungsfall am Endpunkt der Beziehung in jedem Fall verwendet wird.

[1] Ein *Akteur* (engl. *actor*) ist eine Entität, die sich außerhalb des aktuell betrachteten Systems befindet. Der Akteur interagiert mit dem System. Diese Interaktion stellt für das System Ereignisse dar, auf die es reagieren muss. Akteure werden in der ooSe u. a. dazu verwendet, Benutzerprofile zu definieren.

[2] Vgl. [Linssen 1999b].

[3] Zurückzuführen sind diese Diagrammtypen auf JACOBSON. Er ergänzte 1985 Klassendiagramme mit den Namen der ausgetauschten Nachrichten. Daraus entstanden Kollaborationsdiagramme. Sequenzdiagramme wurden von ihm ab 1968 im Bereich des objektbasierten Entwurfs und 1987 in der objektorientierten Entwicklung eingeführt. Vgl. [Jacobson 2000: S. 199].

[4] [Hitz 1999: S. 117].

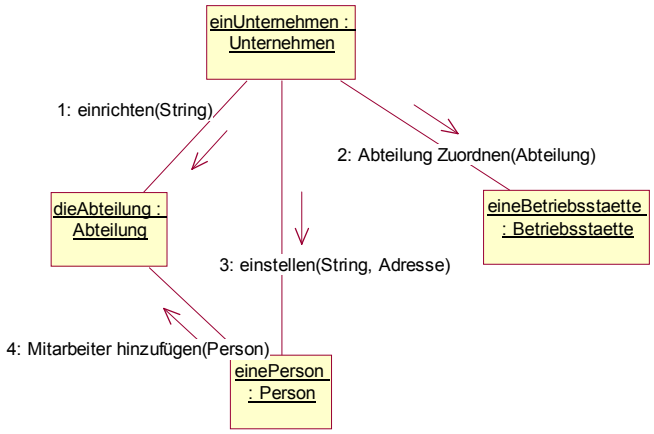


Abbildung 4: Kollaborationsdiagramm der UML

Das Kollaborationsdiagramm stellt die Realisierung einer im Anwendungsfalldiagramm dokumentierten Funktion durch eine Gruppe von Objekten dar. Auf das von außen eintretende Ereignis beginnt der Nachrichtenaustausch der Kollaboration, um die vom System angeforderte Funktion zu realisieren. Die Abfolge der Interaktionen ist in dieser Darstellung durch die Nummerierung der Nachrichten ersichtlich. Das Kollaborationsdiagramm stellt auch Mittel bereit, die Aufrufhierarchie von Operationen abzubilden. Dafür wird die zeitliche Abfolge des Nachrichtenaustauschs weniger deutlich.

2.4.5 Sequenzdiagramm

Das *Sequenzdiagramm* dokumentiert ebenfalls Abläufe und funktionales Verhalten durch Interaktionen mehrerer Objekte.[1] Dabei wird ein möglicher Ablauf eines Anwendungsfalls oder einer Operation dokumentiert.[2] Hervorgehoben wird die zeitliche Anordnung des Nachrichtenaustauschs.

[1] Dies wird von KAPPEL als Interobjektverhalten bezeichnet. Vgl. [Hitz 1999: S. 110].

[2] Dies wird auch als *Szenario* bezeichnet. Vgl. [Linssen 1999], [Linssen 1999b]. Szenarien beschreiben im objektorientierten Ansatz (siehe Abschnitt 2.2.6) mögliche Abfolgen von Ereignissen. Sie finden sich – mit unterschiedlicher Mächtigkeit – in einer Reihe von Modellierungsansätzen. Dabei werden in der Regel Interaktionsdiagramme zur Repräsentation verwendet. Vgl. [Behringer 1997], [Andersson 1995], [IBM 1997], [Selic 1994].

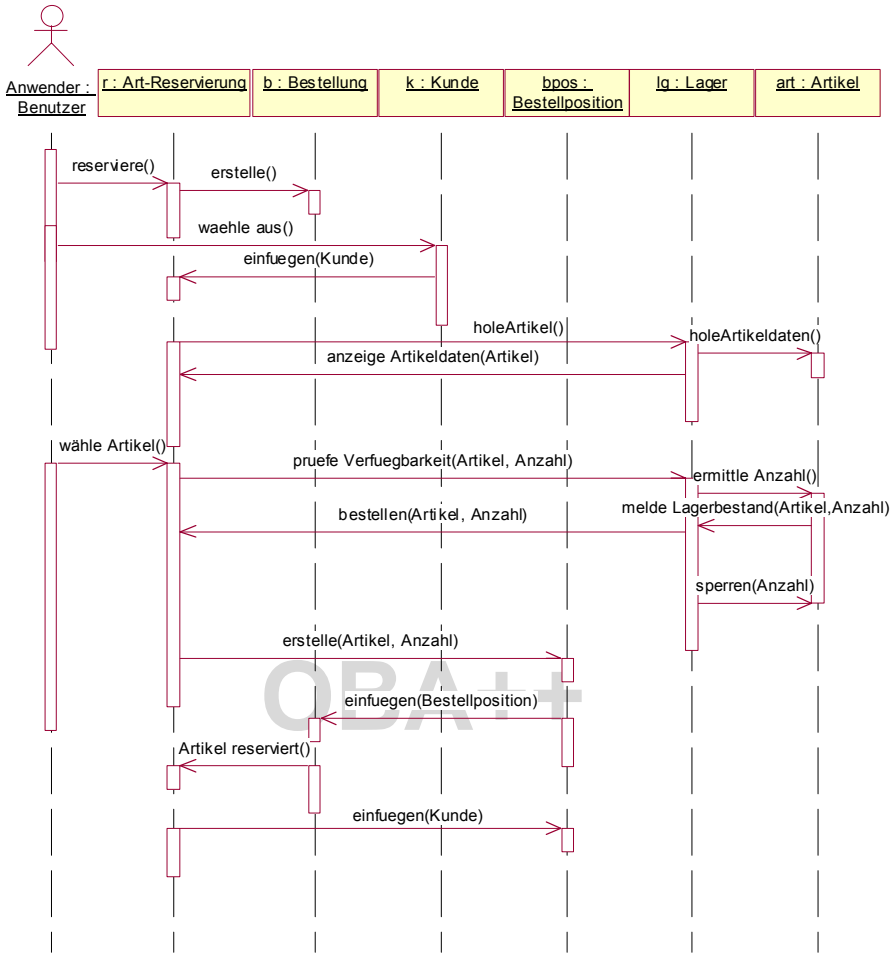


Abbildung 5: Sequenzdiagramm der UML

Die Sequenzdiagramme der UML basieren auf den MESSAGE SEQUENCE CHARTS der SPECIFICATION AND DESIGN LANGUAGE (SDL).[1] „Erfunden“ wurde dieser Diagrammtyp von JACOBSON im Jahr 1968.[2] Das Sequenzdiagramm stellt die Abfolge von Interaktionen dar, die im Rahmen eines Ablaufs stattfinden. Es betrachtet den gleichen Sachverhalt wie das Kollaborationsdiagramm, hebt aber die zeitliche Abfolge der Interaktionen hervor. Im Sequenzdiagramm lassen

[1] Vgl. [ITU-T 1993], [Ellsberger 1997]. Die Diagramme werden allerdings unterschiedlich interpretiert. In der UML kommunizieren Objekte, in der SDL kommunizieren Prozesse. Die MSCs der SDL beinhalten außerdem weitere Ausdrucksmittel, die nicht in die UML übernommen worden sind. Vgl. [Grabowski 1993], [Rudolph 1995], [Rudolph 1995b], [Ben-Abdallah 1997], [Ben-Abdallah 1997b]. Die Integration der Sprachmittel der SDL in die UML ist für zukünftige Versionen gedacht.

[2] Siehe Abschnitt 2.4.4.

sich auch zeitliche Abhängigkeiten dokumentieren.[1] Von RUMBAUGH wurde eine Erweiterung der Notation vorgeschlagen, mit der auch Zustände dokumentiert werden.[2] Diese Variante wurde allerdings nicht von der OMG übernommen.

2.4.6 Zustandsdiagramm

Das *Zustandsdiagramm* dokumentiert die Kontrollaspekte und die Steuerung der Objekte einer Klasse oder einer Methode.[3] Zustandsdiagramme beschreiben die möglichen Abfolgen von Zuständen, die ein Objekt einer Klasse einnehmen kann, sowie die Ereignisse, die die Zustandsübergänge bewirken.[4]

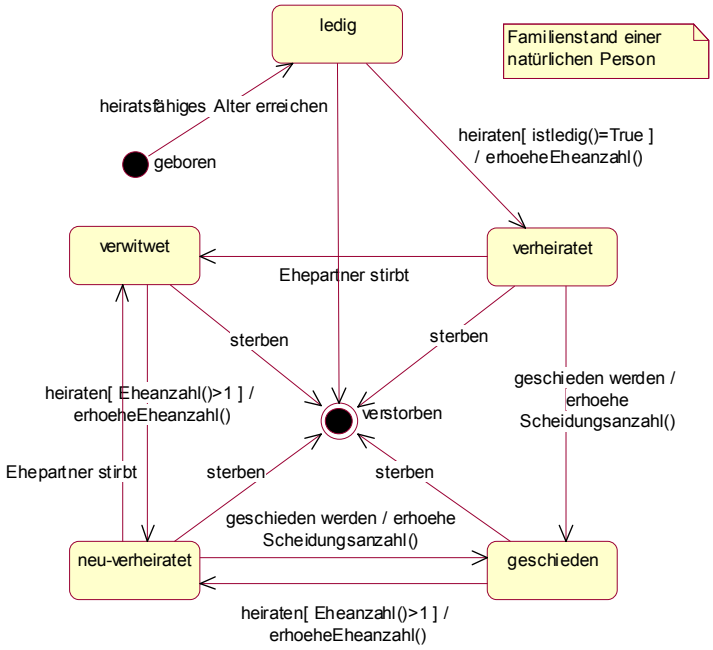


Abbildung 6: Zustandsdiagramm der UML[5]

[1] Darauf wird im weiteren Verlauf noch eingegangen.

[2] Vgl. [Rumbaugh 1999: S. 426].

[3] Vgl. [OMG 1999: S. 3-121].

[4] [Hitz 1999: S. 128, 24 f.].

[5] Die in eckigen Klammern gesetzten Bedingungen dienen nur der Demonstration. Inhaltlich sind sie unnötig.

Zustandsdiagramme sind originär kein objektorientiertes Modell. Sie basieren auf der allgemeinen Automatentheorie der theoretischen Informatik.[1] Die in der UML verwendeten Zustandsdiagramme basieren auf der Notation der HAREL-Statecharts.[2] Diese stellen einen hierarchisch geschachtelten und parallelen Zustandsübergangsautomaten auf der Basis kombinierter MEALY/MOORE-Automaten[3] dar. Die Knoten repräsentieren Zustände, die entweder elementar sind oder durch weitere Statecharts[4] verfeinert werden. Die Kanten stellen die Zustandsübergänge dar. Diese werden durch das Eintreten eines Ereignisses ausgelöst. Mit den Zustandsübergängen können Aktionen oder Operationen der Objekte verbunden sein. Diese greifen auf den Zustand des Objekts zu oder generieren Ereignisse. Auch innerhalb der Zustände können Aktionen ausgeführt werden. Im Gegensatz zur üblichen Interpretation von Statecharts werden in der UML jedoch nicht die Zustände von Prozessen, sondern die Zustände von Klassen modelliert.[5]

Die Masse der objektorientierten Modellierungsansätze verwendet zur dynamischen Modellierung Zustandsdiagramme.[6] Dabei wird häufig auf die von HAREL entwickelte Notation zurückgegriffen. Das Problem besteht darin, dass durch Zustandsdiagramme die Dynamik einer einzelnen Klasse (genauer: eines einzelnen Objekts) dargestellt wird. Ein Geschäftsprozess besteht aber aus einer Mehrzahl von Objekten, die in gegenseitiger Abhängigkeit Verhalten zeigen. Hierfür gelten Zustandsdiagramme als ungeeignet.[7]

2.4.7 Aktivitätendiagramm

Das *Aktivitätendiagramm* stellt eine besondere Form des Zustandsdiagramms dar.[8] Folgt man der Definition des UML Standards der OMG, dokumentiert es die Zustände, die während der einzelnen Schritte eines Ablaufs auftreten, und in welcher Reihenfolge diese Schritte ausgeführt werden.[9] Die

---

[1] Vgl. [Floyd 1998].

[2] Vgl. [Harel 1985], [Harel 1987], [Harel 1987b], [Harel 1988], [Harel 1998], [Coleman 1994], [McGregor 1993].

[3] Das bedeutet, dass sowohl in Zuständen wie auch in Zustandsübergängen Aktivitäten stattfinden.

[4] Wenn hier von Statecharts die Rede ist, sind HAREL-Statecharts gemeint.

[5] Es handelt sich also um so genannte *Objectcharts*, die ursprünglich von COLEMAN entwickelt wurden. Vgl. [Coleman 1994]. Die Unterschiede zwischen den HAREL-Statecharts und UML-Zustandsdiagrammen werden in [OMG 1999: S. 2-150] aufgezählt.

[6] Eine Ausnahme stellt FUSION dar. Dort kommen auch BACKUS-NAUR-Formen zum Einsatz. Vgl. [Coleman 1994: S. 31].

[7] Vgl. [Scheer 1997: S. 15], [Bungert 1995: S. 55].

[8] Vgl. [Rumbaugh 1996b].

[9] Vgl. [OMG 1999: S. 2-151, 3-141].

Knoten in Form von Rechtecken mit konvexen Vertikalen sind keine Operationen, sondern Aktionen bzw. Aktivitäten[1], die in bestimmten Zuständen ausgeführt werden. Die gerichteten Kanten sind Zustandsübergänge, die den Kontrollfluss abbilden. Diese Zustandsübergänge besitzen keine Ereignisse, die sie auslösen. Sie werden durch das Ende der Aktivität am Ausgangspunkt geschaltet und lösen die Aktivität am Endpunkt aus. Synchronisationsbalken sind Pseudozustände zur Synchronisation von folgenden oder vorhergegangenen Aktivitäten.

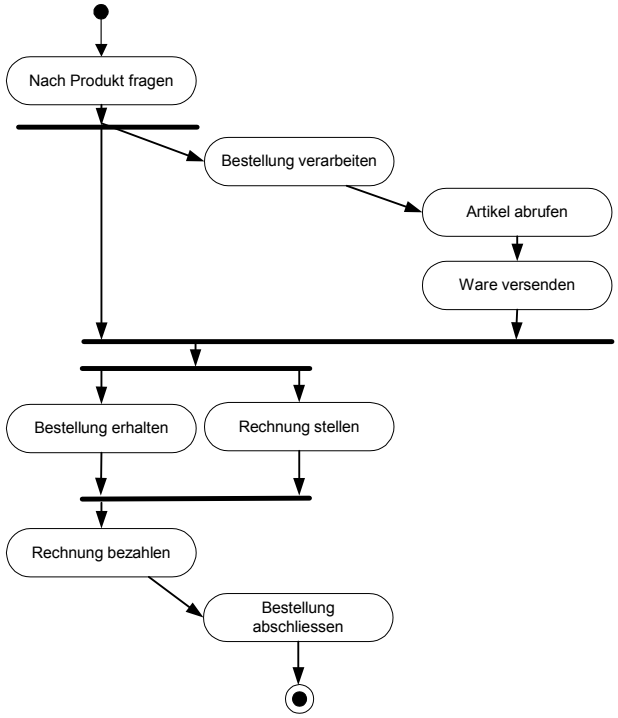


Abbildung 7 Aktivitätendiagramm der UML

Das Aktivitätendiagramm wird verwendet, um die innerhalb einer Operation stattfindende Abfolge von Aktivitäten abzubilden.[2] Es wird aber auch zur Dokumentation globaler Abläufe verwendet, ohne dass die Aktionen oder Aktivitäten Objekten zugeordnet werden.[3] Im Unterschied zu klassischen Flussdiagrammen ist man mit Aktivitätendiagrammen auch in der Lage, parallele

[1] Aktionen und Aktivitäten werden in Zuständen und während Zustandsübergängen ausgeführt. Nach BOOCH sind Aktionen im Gegensatz zu Aktivitäten nicht weiter zerlegbar. Vgl. [Booch 1999: S.261], [Rumbaugh 1999].

[2] Vgl. [OMG 1999: S. 3-141], [Rumbaugh 1996b].

[3] Vgl. [Hitz 1999: S. 152 f.], [Booch 1999: S. 268].

Abläufe zu modellieren. Durch so genannte Schwimmbahnen[1] kann festgelegt werden, welche Objekte für die Ausführung der Aktivität verantwortlich sind. Durch den Objektfluss wird der Datenfluss von Aktivität zu Aktivität abgebildet.

Aktivitätendiagramme wurden nachträglich in die UML aufgenommen. Sie basieren auf den Ereignisschemata von MARTIN, durch die die Abfolge von Operationen und die Ereignisse, durch die Operationen initiiert werden, dokumentiert werden. Diese sind wiederum eine Fortführung der Prozessabhängigkeitendiagramme aus dem *Information Engineering* von MARTIN.[2]

2.4.8 Verteilungsdiagramm

Das *Verteilungsdiagramm* zeigt an, wie die Komponenten eines Softwaresystems physikalisch auf Hardwaresysteme verteilt werden.

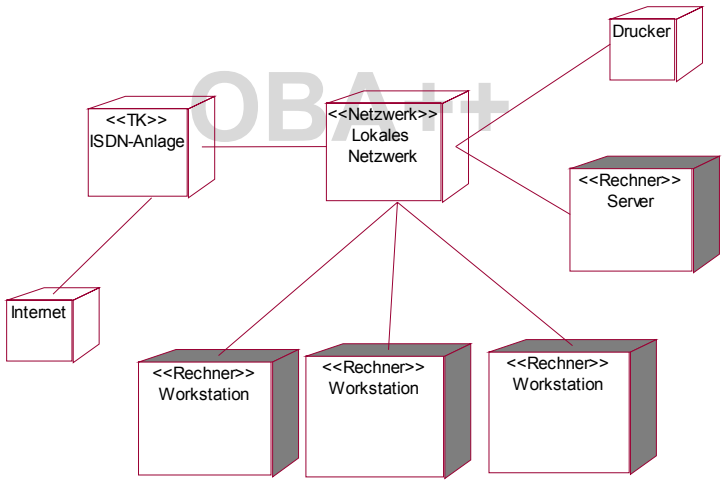


Abbildung 8: Verteilungsdiagramm aus [Booch 1999: S 408]

[1] Vgl. [Booch 1999: S. 265], [OMG 1999: S. 3-145]. Diese Darstellungsform wurde aus der Organisationslehre übernommen. Vgl. [Rummler 1991].

[2] Vgl. [Martin 1990: S. 257 ff.], [Martin 1992: S. 85 ff., 325 ff.]

2.4.9 Komponentendiagramm

Mit Hilfe von *Komponentendiagrammen* wird die Organisation und die Abhängigkeiten zwischen Softwarekomponenten abgebildet.

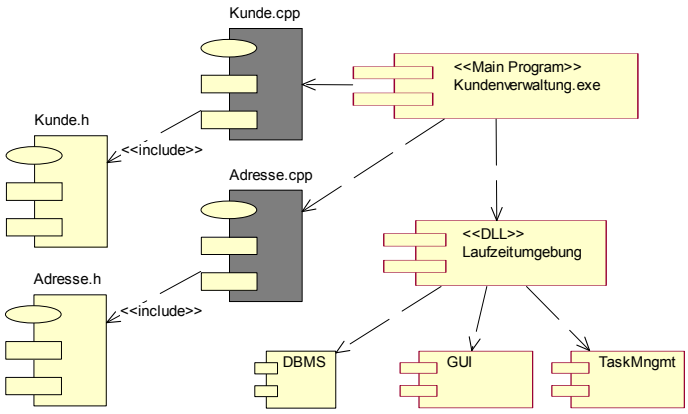


Abbildung 9: UML Komponentendiagramm

Im weiteren Verlauf können Komponenten- und Verteilungsdiagramm außer Acht gelassen werden, da sie die physikalische Organisation von Entwicklungsartefakten[1] (Quellcode, Bibliotheken, Headerdateien usw.) und die physikalische Verteilung lauffähiger Programme dokumentieren. Sie werden auch als Implementierungsdiagramme bzw. als Diagramme zur Beschreibung der Architektur von Softwaresystemen bezeichnet.[1] Darüber hinaus werden das Klassen- und Objektdiagramm zusammen betrachtet, da in beiden die gleiche Notation verwendet wird.

2.5 Zuordnung der Diagramme zu Projektionen und Ebenen

Es stellt sich die Frage, wie die unterschiedlichen Diagramme verwendet werden und welche zur Modellierung von Geschäftsprozessen wichtig sind. Es ergibt sich folgendes Bild, wenn man aus den Projektionen und Ebenen eine Tabelle bildet und die Diagramme zuordnet. Unterstrichene Diagrammtypen zeigen das primäre Verwendungsgebiet eines Diagramms an:

[1] Als Artefakt wird im Software Engineering jedes Zwischenergebnis bezeichnet, welches während des Softwareentwicklungsprozesses erstellt wird.

Projektion	Ebene		
	Intra-Element	**Elementebene**	**Systemebene**
Strukturen	Attribute im Klassend.	Klassend.	Anwendungsfalld., Klassend.
Abläufe und Funktionales Verhalten	Operationen im Klassend., Anwendungsfalld., Aktivitätend.	Kollaborationsd., Sequenzd., Anwendungsfalld., Aktivitätend.	Anwendungsfalld., Aktivitätend.
Kontrollaspekte und Steuerung	Zustandsd., Aktivitätend.	Zustandsd., Aktivitätend.	Zustandsd., Aktivitätend.

Tabelle 2: Zuordnung der Diagramme der UML

Die Tabelle sei zum besseren Verständnis der anschließenden Diskussion erläutert. Dabei werden die Abkürzungen (S)truktur, (A)lauf und funktionales Verhalten, (K)ontrollaspekte und Steuerung, (I)ntra-Element, (E)lementebene und (Sy)stemebene verwendet.

- S/I: Das Klassendiagramm dokumentiert die Struktur einzelner Elemente (= Klassen) durch Attribute.
- S/E: Das Klassendiagramm dokumentiert die Struktur einer Gruppe von Klassen durch die Beziehungen, die zwischen den Klassen existieren.
- S/Sy: Die zwischen mehreren Systemen existierende Struktur wird durch Anwendungsfall-diagramme abgebildet. Wenn Systeme als Klassen abgebildet werden, können hierfür aber auch Klassendiagramme verwendet werden.
- A/I: Das Klassendiagramm dokumentiert das funktionale Verhalten einzelner Elemente (= Klassen) durch Operationen. Das Aktivitätendiagramm dokumentiert den innerhalb einer einzelnen Operation stattfindenden Ablauf. Durch Anwendungsfalldiagramme kann die Funktionalität einzelner Klassen dokumentiert werden.

[1] Vgl. [Hitz 1999: S. 163], [Booch 1999: S. 341], [OMG 1999: S. 1-3]. Auf die Thematik verteilter Objekte wird hier nicht eingegangen. Als Einführung zu diesem Thema vgl. [Orfali 1996].

- A/E: Das Sequenz- und das Kollaborationsdiagramm dokumentieren Abläufe einer Gruppe von interagierenden Objekten unter verschiedenen Gesichtspunkten.[1] Das Aktivitätendiagramm dokumentiert die von einem System unterstützten Abläufe unabhängig von den dabei involvierten Objekten.[2] Das Anwendungsfalldiagramm dokumentiert die Funktionalität einer Gruppe von Objekten.
- A/Sy: Das Anwendungsfalldiagramm dokumentiert die Funktionalität eines Systems – unabhängig von den beteiligten Objekten – durch eine Anzahl von Anwendungsfällen (*Use Cases*). Das Aktivitätendiagramm dokumentiert die von einem System unterstützten Abläufe unabhängig von den dabei involvierten Objekten.[3]
- K/I: Zustandsdiagramme und Aktivitätendiagramme dokumentieren das durch Ereignisse ausgelöste Verhalten einzelner Objekte.
- K/E: Das Zustandsdiagramm dokumentiert durch interagierende Zustandsmaschinen die Veränderungen einer Gruppe von Objekten.[4]
- K/Sy: Die Zustandsveränderungen eines ganzen Systems können durch Zustandsdiagramme und Aktivitätendiagramme dokumentiert werden.

Insgesamt ist festzustellen, dass durch die in der UML definierten Diagrammtypen alle Ebenen und Projektionen abgedeckt werden. Dass für bestimmte Aspekte mehrere Diagramme verwendet werden können, stellt keinen Mangel der Notation dar. Man benötigt in der Praxis unterschiedliche Diagramme, um innerhalb der Projektionen unterschiedliche Gesichtspunkte darzustellen. Ein Beispiel sind hierfür die Interaktionsdiagramme (Sequenz- und Kollaborationsdiagramm): Beide bilden die zwischen Objekten ablaufenden Interaktionen ab. Im Sequenzdiagramm ist die zeitliche Abfolge der Interaktionen deutlicher. Dafür ist die Komplexität der abzubildenden Abläufe durch die gewählte Darstellungsweise beschränkt. Durch Kollaborationsdiagramme lassen sich komplexere Situationen darstellen, dafür wird der zeitliche Verlauf nicht deutlich. Ähnlich ist auch die Verwendung von Anwendungsfalldiagrammen, Sequenzdiagrammen und Kollaborationsdiagrammen zu sehen. Während Anwendungsfalldiagramme Interna ausblenden und die

[1] Vgl. 2.4.4 und 2.4.5.

[2] Auf diese Weise ist eine traditionelle nicht-objektorientierte Technik Bestandteil eines objektorientierten Standards geworden. Hier zeigt sich eine Verwendung in der Praxis, die durch den Standard der OMG in dieser Form nicht vorgesehen, aber auch nicht ausdrücklich verboten ist.

[3] dto.

[4] Vgl. [OMG 1999: S. 3-135]. Hier ist anzumerken, dass seit der *Draft*-Version 1.4 der UML die interagierenden Zustandmaschinen nicht mehr zu finden sind. Vgl. [OMG 2001b]

Funktionalität praktisch als ausdehnungslosen Punkt darstellen, stellen Sequenz- und Kollaborationsdiagramme diese Interna in Form eines Ablaufs dar.

Die Verwendung des gleichen Diagrammtyps auf unterschiedlichen Ebenen ist ebenfalls als Vorteil zu werten, reduziert dies doch die Anzahl der insgesamt notwendigen Diagrammtypen. So kann sowohl die Funktionalität einer einzelnen Klasse, einer Gruppe von Klassen wie auch die eines ganzen Systems durch Anwendungsfalldiagramme dokumentiert werden.[1]

Dagegen ist die mehrfache Verwendung des Aktivitätendiagramms keine Stärke dieses Diagrammtyps, sondern findet ihre Ursache in seiner unpräzise definierten Semantik. Dies führt dazu, dass Aktivitätendiagramme in der Praxis als traditionelle Flussdiagramme oder als Datenflussdiagramme verwendet werden.[2] Mit Aktivitätendiagrammen sollen auch Anwendungsfälle dokumentiert werden.[3] Jeder einzelne Schritt innerhalb des Anwendungsfalls kann dann wieder durch ein Aktivitätendiagramm abgebildet werden. Dieses Verfahren kann rekursiv wiederholt werden, führt aber zu keiner objektorientierten Struktur, sondern zu einer funktional orientierten.[4] Zur Modellierung des funktionalen Verhaltens und der Abläufe auf Intra-Elementebene sind Aktivitätendiagramme sinnvoll. Auf der System- und Elementebene sollten sie nur verwendet werden, um auf einer globalen Ebene Abläufe zu verdeutlichen. Dies stellt dann die Vorstufe einer Modellierung mit Interaktions- und Zustandsdiagrammen dar.

Die Verwendung von Zuständen in Sequenzdiagrammen zur Darstellung systemweiter Kontrollaspekte kann man als Reaktion auf die an bestehenden objektorientierten Ansätzen geübte Kritik verstehen, dass sich objektübergreifende Zustandsübergänge nur schlecht abbilden lassen.[5]

Abgesehen vom Anwendungsfalldiagramm ist die Betrachtung der Systemebene in der objektorientierten Modellierung von geringer Bedeutung. Insofern ist die schwache Abdeckung der entsprechenden Zellen der Tabelle keine Schwäche der UML, sondern hat ihre Ursache im fehlenden Bedarf.

---

[1] Vgl. [OMG 1999: S. 3-83].

[2] Vgl. [Rosenberg 1999: S. 116].

[3] Vgl. [Hitz 1999: S. 152], [Fowler 1998: S. 143].

[4] Aktivitätendiagramme sind selbst aus der Sicht ihrer Erfinder nicht objektorientiert. Vgl. [Rumbaugh 1996b]. SCHNEIDER und WINTERS weisen auf die Lange Tradition dieses Diagrammtyps hin: *„Activity diagrams have been used in many forms under many different names over the last few decades.“* [Schneider 1998: S. 55].

[5] Vgl. [Schienmann 1997: S. 54 f.], [Frank 1994: S. 198], [Popp 1994: S. 18].

2.6 Phasen der objektorientierten Modellierung

2.6.1 Differenzierung von Phasen

Das Ziel der objektorientierten Modellierung ist insgesamt, die gewünschte Systemfunktionalität durch eine Menge interagierender Objekte zu beschreiben und zu realisieren. Die objektorientierten Modelle werden dabei in den unterschiedlichen *Phasen* der Softwareentwicklung mit verschiedenen Zielsetzungen verwendet.[1]

Durch die *objektorientierte Analyse* werden die Objekte des Problembereichs, ihre Struktur, ihr Verhalten und ihre Beziehungen identifiziert.[2] Dies wird auch als *objektorientierte Systemanalyse* bezeichnet.[3] Durch den *objektorientierten Entwurf* werden die Objekte des Lösungsbereichs spezifiziert.[4] COOK und DANIELS, die diese Differenzierung für ungeeignet halten, unterscheiden die folgenden drei Arten von Modellen:[5] Durch das *Essentielle Modell* soll die Mimik des Verhaltens und der Struktur von bestimmten Dingen der realen Welt nachgebildet werden. In anderen Methoden wird dies als *Anforderungsanalyse* oder *Geschäftsmodellierung* bezeichnet.[6] Sehr passend ist der von POPP verwendete Terminus *Modell der Diskurswelt*.[7] Dieses Modell bildet

---

[1] Zu den unterschiedlichen Phasen siehe Abschnitt 1.3.

[2] Vgl. [Kappel 1996: S. 74], [Popp 1994: S. 18].

[3] Als *Systemanalyse* wird ein Verfahren bezeichnet, durch welches die Elemente und Beziehungen von Systemen, die im Anfangsstadium unbekannt sind, durch sukzessive Annäherung ermittelt, Schwachstellen festgestellt und darauf aufbauend ein neues System entworfen und realisiert wird. [Krallmann 1994: S. 4]. Der Begriff Systemanalyse wird schon lange für die frühen Phasen der Softwareentwicklung reklamiert (vgl. [Daniels 1970], [Koreimann 1972], [Wedekind 1973], [Schmidt 1985b], [McMenamin 1988], [Ward 1991], [Yourdon 1992], [Yourdon 1994], [BalzertH 1996], [BalzertH 1999]). Er stammt ursprünglich aus dem Bereich der militärischen Forschung. Es handelt sich um eine Erweiterung des während des Zweiten Weltkriegs entstandenen *Operations Researchs*. Vgl. [Quade 1968], [Optner 1973], [Hillier 1988]. Eine der wenigen Veröffentlichungen, welche die Systemanalyse als ein universell einzusetzendes Verfahren ansieht, ist das Buch von DAENZER und HUBER. Vgl. [Daenzer 1994].

[4] Vgl. [Kappel 1996: S. 74].

[5] Vgl. [Cook 1994: S. 6 ff.]. Eine abweichende, aber kompatible Sicht findet sich in [Züllighoven 1998: S. 147 ff.]. Die Modelle werden dort als *Modell des Anwendungsbereichs*, *Begriffsmodell*, *Entwurfsmodell* und *Implementierungsmodell* bezeichnet. Allerdings ist bei ZÜLLIGHOVEN das Implementierungsmodell schon der Programmcode einer Programmiersprache, während COOK und DANIELS auch hier noch graphische Modelle verwenden. Auch JACKSON verwendet in [Jackson 1995: S. 120 ff.] ähnliche Begriffe, um zwischen Modellen der realen Welt und Modellen für Software zu unterscheiden.

[6] Vgl. Abschnitt 2.1.3.

[7] [Popp 1994: S. 3 f., 19 ff.]

eine Domäne ab, die anschließend unter Umständen nur teilweise als Softwaresystem realisiert wird.

Das *Spezifikationsmodell* bildet den Teil des Essentiellen Modells ab, der durch ein Softwaresystem unterstützt werden soll. Das *Implementierungsmodell* schließlich berücksichtigt Restriktionen, die sich aus der Verwendung von Computern ergeben, und beinhaltet zum Beispiel Strukturen, die sich aus Gründen der Performanz ergeben. Diese Trennung wird als wichtig angesehen, damit man Vorgänge der realen Welt so abbilden kann, wie sie sich dort ereignen, und dabei unberücksichtigt bleiben kann, dass sie unter Umständen bei der Übertragung auf ein Softwaresystem anders realisiert werden müssen.[1]

Unabhängig vom gewählten Begriff ist es den Modellen, die die reale Welt abbilden, gemeinsam, dass es sich um *konzeptuelle Modelle* handelt, d. h., sie bilden eine tatsächliche oder gedachte Domäne unter Vernachlässigung implementierungsrelevanter Aspekte ab.[2] Das bedeutet für die GPM, dass sie unabhängig von der DV-technischen Realisierung beschrieben werden.[3] Bei den Entwurfsmodellen bzw. bei Spezifikationsmodellen muss diese Aussage relativiert werden. Hier findet in der Regel die Berücksichtigung hard- und softwaretechnischer Restriktionen statt.[4] Entwurfsmodelle sind beispielsweise nahezu immer von der Programmiersprache beeinflusst, die in der anschließenden Implementierung verwendet werden soll.

Eine solche Differenzierung in Phasen ist nicht auf objektorientierte Verfahren beschränkt, sondern findet sich auch in nicht-objektorientierten Verfahren und Methoden des Geschäftsprozessmanagements.[5] Was die objektorientierten Verfahren beispielsweise von den strukturierten Methoden oder von ARIS unterscheidet, ist der Umstand, dass durchgängig in allen Phasen die gleichen Modellierungskonstrukte verwendet werden. In der objektorientierten Modellierung werden in allen Phasen Objekte und Klassen, deren Attribute und Operationen sowie die zwischen diesen existierenden Beziehungen verwendet, unabhängig davon, ob es sich um Objekte des Problembereichs, Objekte des Lösungsbereichs oder Objekte einer Programmiersprache handelt. Das bedeutet, dass die erwähnten Diagramme prinzipiell[6] in allen Phasen der Softwareentwicklung zum Einsatz kommen können. Strukturierte Methoden oder Methoden des Geschäftsprozess-

[1] Vgl. [Cook 1994: S. 259 ff.].

[2] Vgl. [Frank 1997f: S. 3].

[3] Vgl. [Müller-Luschnat 1993: S. 75].

[4] Vgl. [Raasch 1993: S. 345].

[5] Vgl. [Scheer 1998b: S. 38 ff.], [Zachman 1987], [Sowa 1992], [Raasch 1993].

[6] Die sog. Implementierungsdiagramme sind hiervon ausgenommen.

managements wie ARIS verwenden dagegen in den unterschiedlichen Phasen der Systementwicklung (dort als Ebenen bezeichnet) unterschiedliche Beschreibungsmittel.[1]

2.6.2 Anwendung in objektorientierten Methoden

Es stellt sich auch bei der objektorientierten Modellierung die Frage, in welcher Reihenfolge die einzelnen Modelle eingesetzt werden. Hier werden wieder die drei Projektionen verwendet, um die einzelnen Modelle in unterschiedlicher Reihenfolge und mit unterschiedlichen Schwerpunkten einzusetzen. Je nachdem, welche Sicht als besonders wichtig angesehen wird, beginnt man entweder mit der Modellierung der Strukturen, der Funktionen oder der Ereignisse.[2]

Strukturzentrierte Verfahren wie die OMT[3] untersuchen primär die Struktur des Systems und erstellen daraus ein Klassendiagramm. Strukturzentrierte Verfahren finden insbesondere bei der Entwicklung betrieblicher Informationssysteme Anwendung, da bei diesen die Datenstrukturen eine große Rolle spielen. Darauf folgt die Untersuchung von Abläufen und der Ereignisse. *Funktionszentrierte Verfahren* wie die Methode OOSE[4] beginnen mit der Untersuchung der Abläufe und Funktionen, die von einem System unterstützt werden sollen. *Ereigniszentrierte Verfahren* beginnen mit der Untersuchung der Ereignisse, auf die ein System reagieren können muss.[5] Funktions- und Ereignisorientierte Verfahren finden bei der Entwicklung von eingebetteten und Echtzeitsystemen Anwendung. Als Beispiel seien hier Steuerungssysteme oder Softwaresysteme im Telekommunikationsbereich genannt.

Zwei Vorgehensweisen sollen hier exemplarisch skizziert werden, um die Verwendung der einzelnen Diagramme zu demonstrieren.[6] Der Ansatz des OOTC zeichnet sich dadurch aus, dass er unterschiedliche Vorgehensweisen beinhaltet. Der UNIFIED SOFTWARE DEVELOPMENT PROCESS (USDP) ist von Interesse, weil er von den ursprünglichen Vordenkern der UML stammt und die Grundlage der heute gängigen Vorgehensmodelle für die UML ist.[7]

---

[1] Vgl. [Scheer 1995: S. 14 - 88], [Raasch 1993: S. 409].

[2] Vgl. [Kappel 1996: S. 56].

[3] Vgl. [Rumbaugh 1993].

[4] Vgl. [Jacobson 1994].

[5] Vgl. [Selic 1994].

[6] Für die Details sei auf die Originalliteratur verwiesen. Eine Übersicht über objektorientierte Vorgehensmodelle findet sich in [Noack 1999] und [Müller-Ettrich 1999].

[7] Vgl. [Jacobson 1999], [Jacobson 2000].

Der Ansatz des OOTC[1] werden drei mögliche Vorgehensweisen unterschieden: Die so genannte szenariengesteuerte Vorgehensweise ist funktionszentriert. Man beginnt mit der Untersuchung der Systemfunktionalität durch Anwendungsfälle. Darauf aufbauend werden mögliche Arbeitsabläufe textuell durch Anwendungsfallbeschreibungen dokumentiert. Die Szenarien werden durch interagierende Objekte in Form von Sequenzdiagrammen dargestellt. Auf der Basis der durch die Sequenzdiagramme gesammelten Informationen erstellt man das Klassendiagramm und die Zustandsdiagramme für die einzelnen Klassen.

Die datengesteuerten Vorgehensweise ist ein strukturzentrierter Ansatz. Man beginnt mit der Erstellung des Klassendiagramms, untersucht die vom System zur Verfügung gestellte Funktion durch Anwendungsfälle, erstellt Szenarien möglicher Abläufe und bildet sie als Folgen von Objektinteraktionen ab. Schließlich werden die Zustände und Zustandsübergänge der Klassen modelliert.

Das zustandsgesteuerte Vorgehen ist ein ereigniszentrierter Ansatz. Man beginnt mit der Erstellung von Zustandsdiagrammen und leitet daraus das Klassenmodell ab. Anschließend werden Anwendungsfälle, Szenarien und Sequenzdiagramme erstellt. Die daraus entstehenden Ergebnisse werden verwendet, um das Klassendiagramm zu verfeinern.

Der USDP definiert ein Vorgehensmodell für die UML.[2] Er basiert in erster Linie auf dem Vorgehensmodell OBJECTORY, welches JACOBSON für die Methode OOSE entwickelt hat.[3] OBJECTORY basiert primär auf der Untersuchung der Abläufe, die ein System unterstützen soll. Aus diesem Grund ist auch der USDP unter die funktionszentrierten Verfahren einzuordnen.

Der USDP beginnt mit der Untersuchung, welche Funktionalität das System zur Verfügung stellt und welche Abläufe es im Rahmen der Funktionalität unterstützt. Die Funktionalität wird durch Anwendungsfalldiagramme dokumentiert, die daraus resultierenden Abläufe durch textuelle Beschreibungen oder Sequenzdiagramme. In geringem Maße kommen auch Aktivitätendiagramme und Zustandsdiagramme zum Einsatz.

Für die Abläufe werden Objekte mit ihren Eigenschaften und Operationen identifiziert, die diesen Ablauf als Abfolge von Interaktionen realisieren. Hierbei werden primär Kollaborationsdiagramme und Objektdiagramme verwendet. Es können aber auch Aktivitätendiagramme, Sequenzdiagramme und Zustandsdiagramme verwendet werden. Die hierbei gewonnenen Ergebnisse werden in das Klassendiagramm übertragen.

---

[1] Vgl. [IBM 1997: S. 39].

[2] Vgl. [Jacobson 1999].

[3] Vgl. [Jacobson 1994].

Die Steuerungsaspekte der einzelnen Klassen werden nun durch Zustandsdiagramme oder Aktivitätendiagramme untersucht. Das Zusammenspiel der einzelnen Klassen wird durch Sequenzdiagramme abgebildet. Neue Operationen, Attribute und Beziehungen werden in das Klassendiagramm übernommen.

Es bleibt festzuhalten, dass die einzelnen Diagramme zur objektorientierten Modellierung in unterschiedlicher Art und Weise eingesetzt werden, wobei die primäre Sichtweise auf das System maßgebend für die Reihenfolge ihrer Verwendung ist. Da Geschäftsprozesse ihrer Natur nach Abläufe darstellen, wäre zur GPM ein funktionszentriertes Vorgehen angebracht. Das bedeutet, dass die im System stattfindenden Abläufe untersucht werden und die dabei gewonnenen Erkenntnisse in das Klassenmodell übertragen werden. Dies gilt es bei der Entwicklung des Modellierungsansatzes zu berücksichtigen.

2.7 Objektorientierte Modellierung von Geschäftsprozessen

Welche Phänomene mit einer objektorientierten GPM im Detail abgebildet werden können sollen, wird noch in Kapitel **7** herausgearbeitet. Hier werden zunächst generelle Eigenschaften untersucht, die ein objektorientierter Ansatz zur GPM erfüllen muss. Ausgangspunkt der Überlegung sind bestehende objektorientierte Ansätze zur GPM.

In der Literatur finden sich mehrere Ansätze, die einen objektorientierten Ansatz verwenden.[1] Im Ansatz des *Semantischen Objektmodells* (SOM) von FERSTL und SINZ werden zur Modellierung Interaktionsdiagramme und Vorgangs-Ereignis-Diagramme verwendet.[2] Beide Diagramme stellen die Interaktionen zwischen den am Geschäftsprozess beteiligten Objekten aus unterschiedlichen Sichtweisen dar. Der Aufbau und der Inhalt der Interaktionsdiagramme ähnelt den Kollaborationsdiagrammen der UML. Die Vorgangs-Ereignis-Diagramme stellen die Abfolge von Interaktionen im Zeitverlauf, die ausgehenden und eingehenden Ereignisse, und die zugehörigen Vor- und Nachbedingungen als Vor- und Nachereignisse dar.[3] Es ähnelt damit den UML-Sequenzdiagrammen. Zwischen Objekten stattfindende Interaktionen werden in SOM als Geschäfts*transaktion* angesehen, an denen zwei Objekte mit ihren jeweiligen Aufgaben beteiligt sind. Eine

---

[1] Weitere objektorientierte Verfahren zur Modellierung von Geschäftsprozessen sind die objektorientierte Erweiterung der EPK (vgl. [Bungert 1995], [Loos 1998], [Scheer 1997], [Nüttgens 1998]) und die Methode OMEGA (vgl. [Fahrwinkel 1995]).

[2] Vgl. [Ferstl 1994], [Ferstl 1994c], [Ferstl 1997]. Diese werden auch als VES bzw. Vorgangs-Ereignis-Schema bezeichnet.

[3] Vgl. das Beispiel in Abschnitt 5.3.2.

Geschäftstransaktion kann in mehrere elementare Transaktionen zerlegt werden, wobei zwischen verschiedenen Transaktionsarten unterschieden wird.[1]

Im Ansatz *Semantic Object Modelling Approach* (SOMA) von GRAHAM[2] werden die im Rahmen eines Geschäftsprozesses auftretenden Aufgaben hierarchisch in Aufgabenbäume zerlegt. Dies stellt jedoch eher eine Technik zur Strukturierung und Komplexitätshandhabung dar. Die objektorientierte Modellierung der Geschäftsprozesse geschieht durch Sequenzdiagramme.

Die Methode *Multi Perspective Enterprise Modeling* (MEMO) von FRANK wird seit 1991 entwickelt.[3] Sie basiert auf seiner Habilitationsschrift über Unternehmensmodellierung auf der Basis des Objektansatzes.[4] MEMO geht weit über die reine GPM hinaus und ist als Architektur zur vollständigen Modellierung von Unternehmen zu verstehen. MEMO beinhaltet drei Perspektiven zur Modellierung des Unternehmens: Strategie, Organisation und Informationssystem. Innerhalb der drei Perspektiven werden mehrere Fokusse verwendet, welche den hier verwendeten Sichten entsprechen. Der Fokus Prozess stellt Abläufe dar, der Fokus Struktur die statischen Elemente, der Fokus Ressource bildet die für Prozesse notwendigen Ressourcen ab. Schließlich stellt der Fokus Ziel mögliche Abhängigkeiten zwischen Zielen dar.[5] Damit entstehen 12 Modelle. Innerhalb der einzelnen Modelle werden unterschiedliche Diagrammtypen verwendet:[6]

[1] Vgl. [Ferstl 1994c: S. 7]. Diese Transaktionsarten werden im hier entwickelten Ansatz als Koordinationsform bezeichnet und in Abschnitt 6.2.4 behandelt.

[2] Vgl. [Graham 1998].

[3] [Frank 1997b: S. 6].

[4] Vgl. [Frank 1994], [Frank 1995], [Frank 1997b], [Frank 1999].

[5] Vgl. [Frank 1997b: S. 6].

[6] Vgl. [Frank 1997b: S. 7- 12].

	Perspektive		
Fokus	**Strategie**	**Organisation**	**Informationssystem**
Prozess	Wertketten bestehend aus Primär- und Sekundäraktivitäten (Support)	Gerichteter Graph mit Aktivitäten und fließenden Informationen	Workflow in einem Softwaresystem
Struktur	SGE, Märkte, Wettbewerber	OrgML: Objektmodell der Organisation (Organisationseinheit, Organisation, Position)	OML: Objektmodell der Informationen
Ressource	Ressourcen und Beziehungen zwischen Ressourcen	Menschen, Büroausstattung, Dokumente, Objekte	Ressourcen des Ist-Systems: Hardwarekomponenten und Netzwerkverbindungen
Ziel	Ziele und Beziehungen zwischen Zielen. Vordefinierte Ziele: shareholder und stakeholder value, generische Marktstrategien nach PORTER	Vordefinierte Ziele: Qualität, Reaktionszeit, Produktivität	Vordefinierte Ziele: Wartbarkeit, Integrität, Modularität, ...

Tabelle 3: Aufbau von MEMO

Durch die verschiedenen Modelle sollen Unternehmen aus unterschiedlichen Gesichtspunkten untersucht werden. Die Modelle (MEMO-OrgML[1] und MEMO-OML[2]) werden durch Metamodelle beschrieben.[3] Die MEMO-OML wird zur Modellierung der Strukturperspektive in Form eines Klassendiagramms verwendet und beinhaltet beispielsweise Primitive[4] wie Klasse, Attribut oder Operation.[5] Diagramme zur Untersuchung der Abläufe und Kontrollaspekte sind noch nicht

[1] Multi Perspective Enterprise Modeling -Organisation Modeling Language

[2] Multi Perspective Enterprise Modeling - Object Modeling Language

[3] Vgl. [Frank 1998].

[4] Ein Primitiv bzw. Modellprimitiv ist das kleinste definierte Element eines Modellierungsansatzes oder einer Programmiersprache.

[5] Vgl. [Frank 1998b], [Frank 1998c].

Bestandteil von MEMO-OML.[1] Die MEMO-OrgML wird zur GPM verwendet und beinhaltet Konzepte wie Prozess, Aktivität, Ressource und Organisationseinheit, die in Form eines Flussdiagramms dargestellt werden. Die Prozesse können hierarchisch zerlegt werden.[2] Aber MEMO-OrgML ist nicht objektorientiert. Es basiert nicht auf dem Prinzip interagierender Objekte, sondern auf dem Prinzip funktionaler Partitionen.[3]

Im Ansatz von KUENG, KAWALEK, BICHLER und SCHREFL[4] wird von einer Ziel-Mittel-Hierarchie und zu beachtenden Randbedingungen ausgegangen. Es werden Aktivitäten definiert, mit deren Hilfe die Ziele erfüllt werden können. Im nächsten Schritt wird durch ein Aktivitätenablaufmodell die logische Reihenfolge der Aktivitäten bestimmt. Durch den Automatisierungsgrad wird festgelegt, welche Aktivitäten durch ein CIS ausgeführt oder unterstützt werden. Anschließend wird festgelegt, welche manuellen und automatisierten Aktivitäten durch welche Rollen ausgeführt werden.[5] Für die Aktivitäten werden die notwendigen Inputs und Outputs definiert. Der Input einer Aktivität umfasst alle Daten, die zu ihrer Ausführung notwendig sind. Der Output einer Aktivität umfasst die Daten, die sie erzeugt. Nun kommen objektorientierte Modellierungsmethoden zum Tragen. Für die Steuerung des Geschäftsprozesses werden Geschäftsfallklassen (*business case classes*) definiert. Für die bearbeiteten betrieblichen Objekte werden Bestandsklassen definiert. Für die Rollen werden Rollenklassen definiert. Es entsteht ein Klassenmodell, für welches die Notation der OSA[6] verwendet wird. Die Lebenszyklen der Klassen werden mit Zustandsdiagrammen modelliert. Durch Interaktionsdiagramme wird der Informationsaustausch zwischen den Objekten modelliert. An diesem Ansatz ist positiv zu bewerten, dass auch organisatorische Einheiten (Benutzer, Rollen, Abteilungen) als Klassen modelliert werden. Damit wird ein wesentliches Merkmal einer objektorientierten Modellierung des gesamten Geschäftsprozesses erfüllt. Außerdem fällt positiv auf, dass die Untersuchung des Automatisierungsgrades ein integrativer Bestandteil des Modellierungsansatzes ist und auch die Aktivitäten betrachtet werden, die manuell ausgeführt werden. Negativ ist, dass die eigentlichen Geschäftsprozesse nicht durch Abfolgen von Objektinteraktionen, sondern durch Ablaufdiagramme modelliert werden. Die Verwendung der Interaktionsdiagramme erfolgt als letzter Schritt der Modellierung und wirkt wie ein Appendix.

---

[1] Vgl. [Frank 1998b: S. 6, 11].

[2] Vgl. [Wenzel 1997]. Das Metamodell wird in [Frank 1999b] besprochen.

[3] Das bedeutet, dass Eingabe-Verarbeitungs-Ausgabe-Strukturen sukzessiv zerlegt werden. Üblicherweise verwendet man hierfür den Begriff *funktionale Dekomposition*. Hier wurde aber in Abschnitt 2.3.2 der Begriff *Partition* eingeführt.

[4] Vgl. [Kueng 1995b], [Kueng 1996], [Kueng 1996b].

[5] Rollen entsprechen in diesem Ansatz den Akteuren der UML. Vgl. [Rumbaugh 1999: S. 144 f.].

[6] Object-Oriented Systems Analysis von EMBLEY, KURTZ und WOODFIELD [Embley 1992].

Dass im Rahmen der UML vorgeschlagen wird, zur GPM Anwendungsfalldiagramme und Aktivitätendiagramme zu verwenden,[1] stellt eine schwache Lösung dar, da weder Anwendungsfalldiagramme noch Aktivitätendiagramme objektorientiert sind und Prozesse nicht als Abfolgen von Objektinteraktionen dargestellt werden. Im Rahmen einer objektorientierten Modellierung wären Geschäftsprozesse aber auf eben diese Weise darzustellen. JACOBSON propagiert ein Verfahren, in dem aus den Anwendungsfalldiagrammen irgendwie[2] die beteiligten Objekte hergeleitet werden und anschließend deren Interaktion als Sequenzdiagramm dargestellt wird. Die Geschäftsprozesse werden einfach durch strukturierten Text beschrieben.[3] Die Herleitung der Objekte bzw. der Modelle wird nicht methodisch unterstützt. Das Konzept wird in dieser Form auch zur GPM empfohlen.[4] Innerhalb der objektorientierten Literatur finden sich eine Vielzahl von kritischen Veröffentlichungen, die sich mit den Problemen der Anwendungsfallmodellierung auseinandersetzen.[5] Anwendungsfalldiagramme sind vielleicht ausreichend, um die von einem System nach außen hin zur Verfügung gestellte Funktionalität abzugrenzen. Zur weiteren Analyse sind sie dagegen nicht geeignet.[6] Auch zur GPM sind sie ungeeignet, da sie von einer eingeschränkten Sichtweise ausgehen. Das Modell eines Geschäftsprozesses muss die Aufgabenträger beinhalten, da Geschäftsprozesse Abläufe in einem soziotechnischen System sind und die Aufgabenträger zu diesem System dazugehören. In Anwendungsfalldiagrammen sind die Aufgabenträger aber die Akteure und damit explizit kein Gegenstand der Modellierung, weil sie als außerhalb des Systems stehend betrachtet werden. Geschäftsprozesse sind dann aber nur die automatisierten Abläufe eines Informationssystems, die als Reaktion auf die von den Akteuren ausgelösten Ereignisse stattfinden. Manuell ausgeführte Geschäftsprozesse wären aus dieser Sicht überhaupt keine Geschäftsprozesse. Die Unterstützung eines Geschäftsprozesses durch ein Anwendungssystem stellt aber nur eine Option dar. Sie ist keinesfalls der bestimmende Faktor für die Existenz des Geschäftsprozesses.

Ebenfalls problematisch ist die GPM durch Sequenzdiagramme. Durch die verwendete Darstellungsform wächst das Diagramm mit jedem zusätzlich beteiligten Objekt in die Breite.[7] Im Sequenzdiagramm besteht keine Möglichkeit, die zugrunde liegende Konfiguration in Form der

---

[1] Vgl. [Mielke 2002], [Oerstereich 1998: S. 78], [Korthaus 1998: S. 226 f.], [Hitz 1999], [Jacobson 1995], [Leffingwell 2000: S. 49 ff.], [OMG 2001: S. 1-10, 2-183], [Staud 2001: S. 342 ff.].

[2] Es findet sich kein methodisches Verfahren.

[3] Vgl. [BalzertH 1999: S. 64 f.]

[4] Vgl. [Jacobson 1994], [Jacobson 1999]. Für die Anwendung dieses Konzept im Rahmen der Geschäftsprozessmodellierung siehe [Jacobson 1995].

[5] Vgl. [Berard 1995], [Firesmith 1995b], [Frank 1997d: S. 79], [Meyer 1997].

[6] Vgl. [Frank 1999b: S. 12].

[7] Vgl. [Linssen 1999].

Objektbeziehungen darzustellen. Aus diesem Grund ist ein Sequenzdiagramm ohne Klassendiagramm, welches die Struktur der beteiligten Objekte dokumentiert, nicht verständlich. Weiterhin ist im Sequenzdiagramm die gemeinsame Darstellung von Zeitverlauf und Kontrollfluss widersprüchlich und missverständlich. Die Verschiebung einer Operation auf der Y-Achse bedeutet nicht unbedingt eine spätere Ausführung der Operation, sondern kann auch durch einen alternativen Ablauf entstanden sein. Auch der Versuch, algorithmische Konstrukte wie Schleifen zu verwenden, kann nicht überzeugen.[1] So gilt das Sequenzdiagramm mit seinen vielfältigen Erweiterungen und Zusätzen als unübersichtlich.[2] Man kann das Resümee ziehen, dass die UML zwar der etablierte Modellierungsansatz zur objektorientierten Modellierung von Softwaresystemen ist, die Mittel zur GPM aber eher schwach sind.[3]

Auch die auf der UML basierenden Ansätze wie USDP[4], CATALYSIS[5], OPEN[6] oder SELECT[7] beschränken sich bei der Betrachtung der Geschäftsprozesse auf jene Abschnitte, die durch CIS unterstützt werden. Geschäftsprozesse bestehen dabei nur aus den Anwendungsfällen, die Akteure mit einem CIS ausführen wollen.[8] Das bedeutet, dass die Entscheidung, welche Aktivitäten automatisiert werden, schon vor der eigentlichen Analyse stattgefunden hat. Eine ganzheitliche Betrachtung erfordert aber, dass der gesamte Ablauf untersucht wird, und zwar unabhängig davon, ob er automatisiert oder nicht automatisiert durchgeführt wird. Erst anschließend wird entschieden, welche Aktivitäten durch betriebliche Anwendungssysteme unterstützt werden. Dies findet so nicht statt. Wesentliche Gestaltungsmöglichkeiten bleiben auf diese Weise unerkannt. Als Ergebnis kann festgestellt werden, dass diese Ansätze den ersten Schritt einer GPM – die Betrachtung der vom

---

[1] Vgl. auch die Kritik von KAPPEL [Hitz 1999: S. 116].

[2] Vgl. [Frank 1997d: S. 79].

[3] Erfahrungen aus der industriellen Anwendung kommen zu ähnlichen Ergebnissen. BORBERG, MÜLLER und KICK berichten in [Borberg 2000] von den Problemen, die sich bei der Modellierung von Geschäftsprozessen mit Anwendungsfalldiagrammen und Aktivitätendiagrammen gezeigt haben. Auch STAUD unterstreicht die Mächtigkeit der UML zur Modellierung von Softwaresystemen, hält sie aber für die Modellierung von Geschäftsprozessen für wenig geeignet. Vgl. [Staud 2001: S. 351 ff.].

[4] Vgl. [Jacobson 1999: S. 122 ff.], [Jacobson 1995].

[5] Vgl. [D'Souza 1999: S. 545 ff.].

[6] Vgl. [Graham 1997: S. 89 f.], [Graham 1998].

[7] Vgl. [Allen 1998].

[8] Das Problem sind nicht die Anwendungsdiagramme, sondern der verengte Blickwinkel auf die Geschäftsprozesse. BALZERT zeigt in [Balzert 1998: S. 721 – 754], dass man auch mit Anwendungsfalldiagrammen zu durchaus vernünftigen Ergebnissen kommen kann. Aber die Technik, Geschäftsprozesse durch strukturierten Text zu beschreiben, fördert nicht die Qualität der Arbeitsergebnisse und kann nicht überzeugen.

CIS unterstützten Abläufe – auslassen und direkt mit dem zweiten Schritt – den von betrieblichen Anwendungssystemen unterstützten Abläufen – beginnen.

Betrachtet man Geschäftsprozesse als Abläufe, in deren Verlauf Aktivitäten stattfinden, und verwendet man einen objektorientierten Ansatz, wird deutlich, dass man Geschäftsprozesse als Abfolge von Objektinteraktionen zu betrachten hat.[1] Hier stellt ein objektorientierter Ansatz zur GPM, durch den die parallele Erstellung von Objekt- und Klassendiagrammen entfallen kann, einen Fortschritt dar. Dabei wird nicht nur das Informationssystem, sondern alle am Geschäftsprozess beteiligten Elemente sollen als Klassen modelliert werden. Auf der Basis eines funktionszentrierten Ansatzes müsste das Klassenmodell aus der Beschreibung der Abläufe bzw. der Geschäftsprozesse zu erzeugen sein. Darüber hinaus müsste die Möglichkeit bestehen, die Operationen der beteiligten Objekte, eintretende Ereignisse, resultierende Zustandsveränderungen und die invarianten Bedingungen der Operationen zu erfassen. Im weiteren Verlauf könnten dann die Ergebnisse der GPM durch die Sprachmittel der UML weiter untersucht werden.

2.8 Einordnung des zu entwickelnden Ansatzes

Im Rahmen der Softwareentwicklung wird auch eine GPM durchgeführt. Der hier entwickelte Ansatz verwendet eine objektorientierte Sichtweise, um konzeptuelle Modelle[2] für Geschäftsprozesse zu erstellen. Dabei kann es sich um Beschreibungen oder Erklärungen existierender oder geplanter Geschäftsprozesse handeln. Der Ansatz wird die Modellierung aller Projektionen in einer einzigen Repräsentation ermöglichen.[3] Die Schwerpunkte der Modellierung sind die Abläufe und das funktionale Verhalten (die sog. Verhaltenssicht). Die Abläufe werden auf allen Betrachtungsebenen (System, Element, Intra-Element) auf der Grundlage interagierender Objekte modelliert, wobei die Elementebene den Schwerpunkt der Betrachtung darstellt. Der Ansatz wird sowohl in der Phase der Anforderungsanalyse wie auch als Spezifikationsmodell verwendet. Das bedeutet,

---

[1] Da die Betrachtung der Kontrollaspekte (Ereignisse, Zustände und Zustandsveränderungen) auf der Ebene einzelner Objekte stattfindet, ist auch für diese Projektion die Modellierung der Objektinteraktionen notwendig. Die Alternative würde darin bestehen, den Geschäftsprozess als ein Objekt anzusehen und dessen Zustände und Zustandsveränderungen zu modellieren. Im Bereich der Workflowmodellierung existieren Arbeiten, die sich mit der Modellierung von Geschäftsprozessen durch Zustandsdiagramme beschäftigen. Vgl. [Weikum 1997], [Wodke 1995].

[2] Konzeptuell, weil keine Anforderungen einer bestimmten Hard- oder Softwareumgebung berücksichtigt werden.

[3] Das hat zur Folge, dass eine getrennte struktur- und verhaltensorientierte Modellsicht (wie z. B. in SOM [Ferstl 1994: S. 213]) nicht erzwungen wird, sondern simultan in einer geschlossenen Darstellung erfolgen kann.

dass sowohl Vorgänge der realen Welt wie auch Vorgänge in einem Softwaresystem auf der Basis des Objektansatzes abgebildet werden können. Dabei wird ein funktions- und ereigniszentriertes Verfahren verwendet. Das bedeutet, dass bei der Modellierung von den Abläufen in einem System, zwischen dem System und seiner Umwelt und von den Ereignissen, auf die das System reagiert, ausgegangen wird. Das Klassenmodell wird aus der Beschreibung dieser Abläufe zu erzeugen sein.

2.9 Zusammenfassung

Modelle sind eine vereinfachende Abbildung eines Ausschnitts der Wirklichkeit unter Anwendung einer bestimmten Sichtweise. Der hier entwickelte Modellierungsansatz besteht aus einer bestimmten Sichtweise (Metapher), einem Metamodell der Sichtweise, einer bestimmten Repräsentation, einer Architektur des Modellierungsansatzes; darüber hinaus dienen Anwendungsbeispiele der Erläuterung.

Dieses Kapitel stellt eine Einführung in den OA dar. Er ist die hier verwendete Metapher des Modellierungsansatzes. Der Ansatz wird dargestellt, ohne dass dabei auf Konzepte von Programmiersprachen zurückgegriffen wird. Die UML ist eine verbreitete Notation zur Darstellung des Objektansatzes. Sie besteht aus unterschiedlichen Diagrammen, die zur Abbildung bestimmter Projektionen und Ebenen in unterschiedlichen Phasen verwendet werden. Zur Modellierung werden hier jedoch nicht die Diagramme der UML verwendet, sondern eine tabellarische Repräsentation. Die Grundlagen dieses Ansatzes sind unter dem Namen OBJECT BEHAVIOR ANALYSIS (OBA) bekannt.[1] Sie sind mit der Architektur des Modellierungsansatzes Inhalt des nächsten Kapitels.

[1] Erste Ideen zur Verwendung im Rahmen der Geschäftsprozessmodellierung wurden 1996 in einem Arbeitspapier skizziert. Vgl. [Linssen 1996].

3 Die Architektur des Modellierungsansatzes

„Alles strömt und nichts dauert." Heraklit von Ephesus[1]

3.1 Die Object Behavior Analysis (OBA) nach RUBIN und GOLDBERG

3.1.1 Terminologie der OBA

Die OBJECT BEHAVIOR ANALYSIS[2] (OBA) ist ein Ansatz zur Erstellung von objektorientierten Modellen. Ausgangspunkt der Analyse sind die Vorgänge in einem System. Diese Vorgänge werden in einzelne Schritte zerlegt und in einer speziellen Repräsentationsform, den sog. Skripten, dokumentiert. Aus den Skripten werden eine Reihe unterschiedlicher Sichten abgeleitet, darunter die Schnittstellen von Klassen.[3]

Die OBA geht also von der objektorientierten Analyse des Verhaltens eines Systems aus.[4] Dieses Verhalten wird auf die Elemente, die das System konstituieren, verteilt. Die Leitlinie der Untersuchung ist, wer Verhalten initiiert (*initiates*) und wer am Verhalten partizipiert (*participates*). Das gesamte Verhalten eines Systems wird von so genannten Rollen (*Roles*) ausgelöst, und das gesamte Systemverhalten wird von Rollen ausgeführt. Die auslösende Rolle wird in der OBA als *Initiator* bezeichnet. Die Rolle, welche das Verhalten ausführt, wird als *Participant* bezeichnet. Das Verfahren verwendet also die Systemdefinition des Objektansatzes, nach der ein System aus einer *Gesellschaft* von Objekten besteht, die voneinander – über Interaktionsbeziehungen – Dienste anfordern.[5]

---

[1] [Grünwald 1991: S. 98].

[2] An dieser Stelle sei der Firma GEORG HEEG OBJEKTORIENTIERTE SYSTEME gedankt, die mir freundlicherweise ihre Unterlagen zur OBA und das Werkzeug METHODWORKS zur Verfügung gestellt hat.

[3] Die OBA wurde vom Verfasser untersucht und in der Praxis eingesetzt. Sie ist im Rahmen der Vorarbeiten ausführlich in [Linssen 1998b] beschrieben. Die Möglichkeiten der Verwendung der OBA zur Modellierung von Geschäftsprozessen wurden erstmalig in [Linssen 1996] skizziert.

[4] Vgl. [Rubin 1992].

[5] Siehe das Kapitel über den objektorientierten Ansatz.

In der Terminologie der OBA gehen *Initiator* und *Participant* einen Vertrag (*Contract*) ein, in dem der Participant zusichert, eine bestimmte Leistung zu Verfügung zu stellen. An jedem Vertrag sind mit *Initiator* (Spalte 1) und *Participant* (Spalte 3) zwei Parteien beteiligt. Die *Action* (Spalte 2) stellt das Verhalten des Initiators dar, in dessen Rahmen das Verhalten des *Participants* benötigt wird. Der *Service* (Spalte 4) stellt das Verhalten des *Participants* dar, das im Rahmen des Vertrags zur Verfügung gestellt wird. Initiator und Participant stehen in einer Interaktionsbeziehung. Nach RUBIN und GOLDBERG basieren die Verträge auf vier möglichen Mustern.[1] Jeder Vertrag wird als eine Zeile im Skript dargestellt:

Initiator	Action	Participant	Service
thing1	notifies	thing2	thing2 can be notified

Abbildung 10: Muster eines Vertrags[2]

Der Vertrag in Abbildung 10 lässt sich so interpretieren, dass ein *thing1* mit einem *thing2* interagiert, und zwar in der Weise, dass *thing2* darüber informiert wird, dass ein Ereignis eingetreten ist. Damit *thing2* dies bemerken kann, muss es – gemäß den Prinzipien des objektorientierten Ansatzes – eine entsprechende Operation zur Verfügung stellen.

RUBIN und GOLDBERG betonen, dass es sich bei dem *Service* des *Participants* nicht um die Reaktion auf die *Action* des *Initiators* handelt. Es handelt sich vielmehr um die Spezifikation der Schnittstelle, die der *Participant* im Rahmen des *Contracts* zur Verfügung stellen muss, um den Vertrag einzuhalten. Die eigentliche Reaktion des *Participants* wird erst nachfolgend abgebildet. Dabei wird der *Participant* zum *Initiator*, da er nun die aktive Rolle innehat. Die OBA ist eine konsequente Anwendung des *Information-Hiding*-Prinzips im OA: Von Objekten sieht man nur, auf welche Ereignisse sie reagieren und welche Ereignisse sie erzeugen. Die folgende Abbildung zeigt die übrigen Muster der *Contracts*:

[1] Vgl. [Rubin 1992: S. 52].

[2] Vgl. [Rubin 1992: S. 52].

thing1	provides info to	thing2	thing2 can accept info
thing1	requests info from	thing2	thing2 can provide info
thing1	requests service from	thing2	thing2 can provide service

Abbildung 11: Weitere Muster für Verträge[1]

Bei einem Skript handelt es sich um eine Abfolge von *Contracts*. Die folgende Abbildung zeigt einen Ausschnitt aus dem Aufsatz von RUBIN und GOLDBERG:

Initiator	**Action**	**Participant**	**Service**
User	select D1	Spreadsheet	select a cell
User	type text NEW	D1	set content to text
User	set text style to bold	D1	set text style to bold
User	select A2	Spreadsheet	select a cell
	(repeated select and type text for example)	B2, C2, D2, A3 through A10	
User	select Row 2	Spreadsheet	select a row
...	...	...	...

Abbildung 12: Beispiel für ein Skript[2]

Skripte erhalten außerdem Vor- und Nachbedingungen, um auszudrücken, unter welcher Bedingung ein Skript ausgeführt wird und welchen Zustand das System nach der Ausführung des Skripts hat.

[1] Vgl. [Rubin 1992: S. 52].

[2] [Rubin 1992: S. 53].

3.1.2 Vorgehensweise

Im Rahmen der Analyse wird mit Skripten das Verhalten des Systems untersucht. Im nächsten Schritt werden aus diesen Skripten so genannte Glossare erzeugt. Glossare (*Glossary*) sind Sichten, die durch Umformungen aus den Skripten abgeleitet werden sollen. Folgende Glossare werden von RUBIN und GOLDBERG verwendet:[1]

- Teilnehmerglossar (*Party Glossary*)
- Dienstglossar (*Service Glossary*)
- Glossar der logischen Eigenschaften (*Attribute Glossary*)

Das Teilnehmerglossar (*Party Glossary*) beinhaltet alle im Rahmen der Ablaufanalyse verwendeten Rollen. Man erstellt eine Tabelle mit allen in den Skripten verwendeten Begriffen, der Referenz auf das entsprechende Skript, und einem Indikator, ob der Begriff in der Rolle des *Initiators* oder des *Participants* auftritt. Im *Service Glossary* finden sich alle Aktivitäten, die insgesamt vom System ausgeführt werden, und von welchem *Participant* sie ausgeführt werden. Das Glossar der logischen Eigenschaften (*Attribute Glossary*) lässt sich nicht aus den Skripten ableiten. Es soll aus der Überlegung entstehen, welche Attribute ein Teilnehmer (*Party*) besitzen muss, um seine *Contracts* erfüllen zu können. Für jeden *Participant* und für jeden *Initiator* wird ein separates Glossar der Attribute erstellt.

3.1.3 Weiterer Fortgang der Analyse

Im weiteren Fortgang der Analyse wird entschieden, aus welchen Rollen Objekte und Klassen entstehen. Das Ergebnis dieses Schrittes sind so genannte *Objektmodellkarten*, die folgende Informationen enthalten:[2]

- den Namen des Objekts
- die Namen der Objekte, von denen Attribute und Verhalten geerbt wird
- Informationen und Verhalten, welches durch das Objekt hinzugefügt wurde
- die Attribute des Objekts
- zur Verfügung gestellte Dienste bzw. Operationen
- von diesem Objekt angeforderte Operationen anderer Objekte

---

[1] Vgl. [Rubin 1992: S. 55].

[2] Vgl. [Rubin 1992: S. 57 f.].

- Verweise auf die Skripte, in denen das Objekt verwendet wurde

Auf der Basis dieser Informationen lassen sich Diagramme der Vertragsbeziehungen erstellen. Diese beinhalten für jedes Objekt die Informationen, im Rahmen welcher Aktionen es mit welchen Objekten interagiert und welche Operationen dabei in Anspruch genommen werden. Dabei können verbreitete graphische Notationen verwendet werden. Hier wird von den Verfassern explizit auf die Notationen von RUMBAUGH und BOOCH verwiesen.[1] Auf der Basis der Techniken von HAREL[2] soll abschließend eine Zustandsmodellierung durchgeführt werden können.[3]

3.1.4 Offene Fragen zur Object Behavior Analysis

Die OBA wurde als Grundlage des zu entwickelnden Modellierungsansatzes gewählt, weil sie primär auf der Untersuchung der in einem System stattfindenden Interaktionsbeziehungen beruht. Damit entspricht sie dem in Abschnitt 1.4 postulierten *Primat der Interaktionsbeziehung*. Ihre Beschreibung ist aber leider nur ein Fragment geblieben. Die von RUBIN und GOLDBERG geplante Monographie, mit der die Methode vollständig und umfassend dokumentiert werden sollte, wurde nie veröffentlicht. Deshalb ist man bei der Bewertung der OBA insgesamt auf spärliches Schriftgut[4] angewiesen. Dabei bleiben eine Reihe von Fragen unbeantwortet:

- Die Modellierung in der OBA findet auf der Basis der Konzepte *Rolle* und *Objekt* statt.[5] Tatsächlich werden in den Skripten Klassen und Objekte gemischt verwendet. Bei der Ableitung der Teilnehmerglossare, spätestens aber bei der Erstellung der Objektmodellkarten werden die Konzepte Klasse und Objekt vermischt. Die im Rahmen der Skripterstellung verwendeten

[1] Vgl. [Rubin 1992: S. 60].

[2] Vgl. [Harel 1987].

[3] Vgl. [Rubin 1992: S. 61].

[4] Vgl. [Rubin 1992], [Rubin 1993], [Rubin 1994], [Gibson 1990], [ParcPlace 1992], [ParcPlace 1994], [ParcPlace 1995]. Diese Aufzählung lässt den Eindruck eines umfassenden Quellen-Materials entstehen. Tatsächlich sind diese Quellen äußerst skizzenhaft und von geringem Wert, um die OBA als Methode darzustellen.

[5] Da die Skripte das Verhalten eines Systems dokumentieren, beschreiben sie die Interaktion von Objekten. Aus dem oben gezeigten Beispielskript ist ersichtlich, dass das Kalkulationsprogramm nicht auf die Auswahl irgendeiner Zelle durch den Anwender reagiert (was einer klassenorientierten Sichtweise entsprechen würde), sondern auf die Auswahl eines bestimmten Exemplars (also eines Objekts) einer Zelle (in diesem Fall der Zelle „D1“).

Objekte müssten Rollen oder Klassen zugeordnet werden. Ein solcher Abbildungsprozess von der Objekt- auf die Klassenebene fehlt.[1]

- Es ist unklar, wie die Objektmodellkarten entstehen. An Stelle der Objektmodellkarten wäre die Erzeugung von UML-Klassendiagrammen sinnvoller. Hierfür ist ein Verfahren zu entwickeln.
- Die Semantik der von Objekten zur Verfügung gestellten Dienste (Spalte *Service*) ist im Rahmen des Objektansatzes intuitiv zu erschließen: Aus softwaretechnischer Sicht handelt es sich um Methoden, die in einem objektorientierten Programm von Objekten ausgeführt werden. Die Semantik der Aktionen dagegen bleibt unklar.
- Die Verwendung des Begriffes *Contract* stellt eher eine Wortspielerei dar. Die Tatsache, dass ein Objekt eine Leistung eines anderen Objekts verwendet, beinhaltet – außer der Tatsache, dass zwei Vertragspartner vorhanden sind – wenig Vertragsähnliches. Es ist unklar, wie die Bedingungen, unter denen der *Participant* die Operation ausführt, dokumentiert werden sollen.
- Im ersten Entwurf der OBA[2] sind innerhalb der Skripte nur sequentielle Vorgänger-Nachfolger-Beziehungen möglich. Dies ist bei der Interpretation eines Skripts als Szenario (im Sinne einer Abfolge von Ereignissen) sinnvoll und richtig. Darüber hinaus würde die Verwendung von Kontrollanweisungen so etwas wie einen objektübergreifenden Kontrollfluss beinhalten, was im Rahmen des objektorientierten Ansatzes als eine ungewollte prozedurale Sichtweise angesehen wird. Nichtsdestotrotz zeigt die praktische Anwendung des Verfahrens, dass das Fehlen aller Kontrollanweisungen zu äußerst komplizierten Modellen führt. Später wurden tatsächlich Kontrollanweisungen in der OBA eingeführt, ohne dass deren Semantik genauer erläutert wurde.[3]
- Es ist unklar, wie die Glossare der Attribute der Teilnehmer entstehen sollen. Attribute werden in den Skripten nicht verwendet. RUBIN und GOLDBERG geben hier nur vage und unpräzise Hinweise.[4] Es wäre sinnvoller, wenn auch Attribute in Skripten erfasst werden könnten. Dabei müsste jedoch die Syntax des Modellierungsansatzes verhindern, dass das Prinzip des Information Hiding gebrochen wird.

---

[1] RUBIN und GOLDBERG erwähnen die Verwendung eines *Alias*-Glossars, um uneinheitlich verwendete Begriffe durch einheitliche Termini zu ersetzen. Mit Hilfe einer solchen Ersetzungstabelle ließe sich in rudimentärer Form dieser Abbildungsprozess durchführen. Diese Möglichkeit scheint den Autoren nicht bewusst geworden zu sein, sie erwähnen diese Möglichkeit an keiner Stelle. Insgesamt hat man den Eindruck, als ob der Unterschied zwischen Objekt und Klasse sehr vage und unpräzise verwendet wird.

[2] Vgl. [Rubin 1992], [Rubin 1993], [Gibson 1990].

[3] Vgl. [Rubin 1994], [ParcPlace 1995].

[4] Vgl. [Rubin 1992: S. 54].

- Die Skripte ermöglichen nur die Abbildung von Interaktionsbeziehungen. Innerhalb der Skripte ist keine Modellierung von statischen Beziehungen möglich. Struktur und Dynamik bedingen sich in der Regel aber gegenseitig. Dies führt dazu, dass man – neben den Skripte – parallel Klassen- und Objektdiagramme erstellen muss. Bei Änderungen der Skripten müssen diese ständig abgeglichen werden. Aus diesem Grund wäre es sinnvoll, auch statische Beziehungen innerhalb der Skripte dokumentieren zu können.
- Der Inhalt der Interaktion kann nicht abgebildet werden. Es ist nicht klar, welche Daten vom *Initiator* zum *Participant* übertragen werden.
- In den Skripten lassen sich Verweise auf Unterskripte einfügen. Die Semantik dieser Unterskripte ist allerdings unklar. Sie könnten eine funktionale Dekomposition des Dienstes des *Participants* darstellen. Sie könnten aber auch bedeuten, dass im Rahmen der Analyse eine bisher nur unscharf formulierte Verantwortlichkeit eines *Participants* durch präziser formulierte Dienste verfeinert werden. Eine dritte Interpretationsmöglichkeit besteht darin, dass der *Participant* zur Erfüllung seines Dienstes selbst Interaktionen mit anderen Teilnehmern eingeht. Hier ist eine präzise Semantik zu entwickeln.
- Geht man davon aus, dass zur Analyse eines Systems nichttrivialer Komplexität eine Vielzahl von Skripten notwendig ist, stellt sich die Frage, wie diese Skripte zu strukturieren sind. RUBIN und GOLDBERG stellen kein Konzept für die Strukturierung von Skripten vor.
- Bei den Skripten fehlt die Information, auf welcher Ebene die Interaktion beschrieben wird. Stellt das Skript die Interaktion zwischen Systemen dar? Wird die Interaktion eines Akteurs (also zum Beispiel eines Benutzers) mit einem System dargestellt? Wird die Interaktion auf der Ebene miteinander interagierender Subsysteme beschrieben, oder wird die systeminterne Interaktion zwischen den Klassen und Objekten eines Systems dokumentiert? Hier sind Konzepte zur Ebenen- bzw. Abstraktionsbildung notwendig.
- Es wird keine Aussage gemacht, ob alle Spalten eines Skripts besetzt sein müssen. Hier sind Situationen zu untersuchen, bei denen Spalten möglicherweise unbesetzt bleiben können.

Während Methoden wie OOAD oder OMT[1] strukturorientiert mit der Untersuchung der Objekte bzw. Klassen und den zwischen ihnen bestehenden Beziehungen beginnen, setzt die OBA dynamikorientiert bei den im System stattfindenden Abläufen an, zerlegt das System in Objekte, verteilt die Gesamtfunktionalität und geht von dort zur Untersuchung und Definition der strukturellen Beziehungen über. Dabei ist insbesondere die Idee hervorzuheben, dass durch die gewählte Notation die Schnittstelle der Klassen aus den Abläufen generiert werden können soll.

[1] Vgl. [Booch 1995], [Rumbaugh 1993].

Ein weiterer Vorteil besteht darin, dass die Skripte eine kompaktere Darstellung als die Sequenzdiagramme der UML sind, weil sie nicht mit jedem zusätzlichen Objekt in die Breite wachsen.[1]

Die OBA ist außerdem eine der wenigen objektorientierten Methoden, die weniger auf der Erstellung von Diagrammen, sondern mehr auf der Verwendung natürlicher Sprache in einer formalisierten Notation beruht.[2] Sie ist bis heute die einzige Methode im Software Engineering, die ihre Wurzeln in der kognitiven Psychologie und Wissensrepräsentation hat. Sie greift von dort die Idee der Skripte auf, gibt diesen aber durch die verwendete tabellarische Notation eine semiformale Struktur. Diese Form ermöglicht eine maschinelle Verarbeitung der Skripte. Damit ist sie entsprechenden Konzepten der UML – hier sind insbesondere die Use Cases zu nennen – überlegen.[3]

3.2 Die erweiterte Object Behavior Analysis (OBA++)

Der hier entwickelte Modellierungsansatz OBA++[4] stellt eine Erweiterung und Formalisierung der OBA von RUBIN und GOLDBERG dar. Die Idee der OBA, Prozesse als Abfolgen von Interaktionsbeziehungen zu betrachten, wird im Folgenden als Ausgangspunkt für die Weiterentwicklung übernommen. Die tabellarische Notation mit vier Spalten wird um zwei weitere Spalten erweitert, und in den einzelnen Spalten werden zusätzliche Informationen hinzugefügt. Hierfür wird eine spezielle Syntax entwickelt.

Abweichend von der OBA beinhaltet die OBA++ nicht nur eine Repräsentation, sondern außerdem ein Metamodell und eine Architektur. In folgenden Abschnitt wird diese Architektur des Modellierungsansatzes entwickelt.

---

[1] Dies ist ein wichtiges Kriterium, weil sich in der Praxis häufig zeigt, dass an Prozessen eine größere Anzahl von Objekten beteiligt sind, als sich mit Sequenzdiagrammen vernünftig darstellen lassen.

[2] Die andere Methode ist das RESPONSIBILITY DRIVEN DESIGN (RDD) mit den so genannten CRC-Karten (Classes, Responsibilities and Collaborations) von WIRFS-BROCK, WILKERSON und WIENER [Wirfs-Brock 1993]. Nach dem vorliegenden Quellenmaterial (insbesondere [Gibson 1990], [Rubin 1992]) war dieser Ansatz bei der Entwicklung der OBA bekannt. Ob und wie weit er ihn beeinflusst hat, ist nicht nachzuweisen, liegt aber nach der persönlichen Ansicht des Verfassers nahe. Es ist ebenfalls unklar, ob das von JACOBSON stammende Konzept der *Use Cases* (Anwendungsfälle. Vgl. [Jacobson 1994], [Jacobson 2000]) ein weiterer beeinflussender Faktor ist, oder eine Parallelentwicklung darstellt.

[3] Vgl. [Linssen 1999b].

[4] Ausgesprochen: „*O-B-A-plus-plus*“. Das Symbol ‚++’ geht auf den Inkrement-Operator der Programmiersprache C++ (vgl. [Stroustrup 2000]) zurück.

3.2.1 Schemata der Architektur

Eine Architektur beschreibt die Struktur eines Systems durch seine Komponenten und deren Beziehungen untereinander.[1] Vorbild für die verwendete Architektur des Modellierungsansatzes ist die Drei-Ebenen-Schema-Architektur der „*ANSI/X3/SPARC/Study Group on Database Management Systems*“.[2] In Anlehnung an diese Architektur werden drei Schemata verwendet:

- Das *Externe Schema*.[3] Das Externe Schema stellt die Repräsentation des Modellierungsansatzes dar. Da hier zur Repräsentation Skripte verwendet werden, wird es als *Skriptmodell* bezeichnet.
- Das *Konzeptuelle Schema*. Das Konzeptuelle Schema stellt das Gesamtmodell aller Externen Schemata dar. Hier wird ein *Semantisches Netz* verwendet.
- Das *Interne Schema*. Das Interne Schema stellt die softwaretechnische Realisierung des Konzeptuellen Schemas dar. Es wird auch als *Objektmodell* bezeichnet.

Das Konzeptuelle Schema beinhaltet – unabhängig von einer softwaretechnischen Realisierung durch ein Basissystem – alle Externen Schemata. Ein Externes Schema beinhaltet eine problemspezifische Teilsicht bzw. Projektion auf das Konzeptuelle Schema. Die hier verwendeten Skripte stellen – neben möglichen anderen Externen Schemata – eine Repräsentation des Inhalts des Konzeptuellen Schemas dar und geben dem hier verwendeten Externen Schema seinen Namen. Das Interne Schema beschreibt die Realisierung des Konzeptuellen Schemas, die abhängig vom verwendeten Basissystem (Programmiersprache, Datenbankmanagementsystem usw.) ist.

Der Aufbau des Konzeptuellen Schemas wird durch dessen Metamodell festgelegt. Dieses Metamodell ist gleichzeitig das Begriffssystem der verwendeten Metapher. Der Aufbau des Externen Schemas (der Skripte) wird durch ein weiteres Metamodell festgelegt. Dieses Metamodell definiert, wie sich der Modellierungsansatz dem Anwender „nach außen“ repräsentiert, also sein „Aussehen“. Der Aufbau des Internen Schemas ist abhängig vom verwendeten Basissystem.[4] Aus

---

[1] Man beachte, dass dies eine verengende Verwendung des Begriffs Architektur darstellt, wie er in der Softwaretechnik verwendet wird. Der in der Baukunst verwendete Begriff der Architektur ist weit umfassender. Vgl. [Müller 1994b]. Zum Begriff der Architektur im Software Engineering und in der Informatik siehe [Shaw 1996: S. 1], [Rechenberg 1999: S. 523, 784], [Wettstein 1993: S. 16], [Foegen 2001].

[2] Vgl. [Heuer 1995: S. 27 ff.], [Date 1990: S. 31], [Kemper 1999: S. 17 ff.], [Rechenberg 1999: S. 879], [Peckham 1988: S. 153].

[3] Als Synonym für ‚Schema‘ wird auch ‚Sicht‘ verwendet.

[4] Der Aufbau des Internen Schemas ist abhängig von der Realisierung des Modellierungsansatzes als Softwaresystem und nicht Bestandteil dieser Arbeit.

diesem Grund wird hierfür kein Metamodell entwickelt. Die übrigen Metamodelle basieren auf dem OA und haben die Form von UML-Klassendiagrammen:

Schema	Repräsentationsform	Repräsentation des Metamodells
Externes Schema	Skript	UML-Klassendiagramm
Konzeptuelles Schema	Semantisches Netz	UML-Klassendiagramm
Internes Schema	UML-Objektmodell	– abhängig vom verwendeten Basissystem –

Tabelle 4: Darstellung von Inhalt und Metamodell der Schemata

Bei einer softwaretechnischen Realisierung des Modellierungsansatzes wird der Inhalt der Externen Repräsentation (die Skripte) in das Konzeptuelle Schema (das Semantische Netz) umgesetzt. Der Inhalt des Konzeptuellen Schemas wird schließlich – abhängig von der gewählten Basissoftware – als Internes Schema gespeichert.

Für die objektorientierte Modellierung sollen mehrere unterschiedliche Sichten möglich sein, die jeweils auf Projektionen des vorhandenen Konzeptuellen Schemas beruhen. Die Projektionen beinhalten dann anwendungsspezifische Ausschnitte des Konzeptuellen Schemas.[1] Alle Daten werden in einem Gesamtmodell erfasst, welches unabhängig vom gewählten Basissystem sein soll. Nur auf der Ebene des Internen Schemas basiert die Realisierung des Konzeptuellen Schemas auf einem gewählten Softwaresystem. Auf diese Weise abstrahiert das Konzeptuelle Schema von den Anforderungen, die sich aus der Speicherung und Verarbeitung der Modelle durch ein Basissystem ergeben. Damit abstrahiert das Externe Schema von der Komplexität des Gesamtmodells und ermöglicht eine problemspezifische Sichtweise.

Zwischen diesen drei Schemata, die idealisiert als Schichten betrachtet werden können, sind Transformationsregeln notwendig, um den Inhalt der Schemata aufeinander abzubilden. Dabei ist insbesondere die Transformation des Externen Schemas in das Konzeptuelle Schema von Bedeutung, da dadurch auch der Zusammenhang zwischen der Repräsentation und dem Metamodell des Modellierungsansatzes hergestellt wird.

[1] Vgl. [Heuer 1995: S. 28].

3.2.2 Abbildungsprozess

Das Konzeptuelle Schema basiert nicht ohne Grund auf dem Konzept Semantischer Netze. Es ist eine Darstellungsform notwendig, mit der sich Sachverhalte auf einer frei wählbaren, also auch beliebig elementaren Ebene darstellen lassen. Die Abstraktionsebene des Objektansatzes, einzelne Klassen zu betrachten, ist nicht feingranular genug, da es notwendig ist, beispielsweise einzelne Operationen einer Klasse referenzieren zu können. Bei der Suche nach einer geeigneten Repräsentationsform fiel die Wahl auf einen Ansatz aus dem Bereich der Wissensrepräsentation, die so genannten Semantische Netze,[1] weil mit ihnen die Granularität der Darstellung frei wählbar ist. Semantische Netze bestehen aus unterschiedlichen Arten von Knoten und Kanten. Die Knoten werden durch Kanten verbunden. Die Knoten eines Semantischen Netzes repräsentieren Umstände, die Kanten setzen diese Umstände in Beziehung zueinander. Das hier konzipierte Semantische Netz basiert außerdem auf gerichteten Kanten. Jede Kante besitzt genau einen Ausgangsknoten und genau einen Zielknoten. Die Richtung der Kante stellt die Leserichtung der Beziehung dar. Die von BRACHMAN und WOODS[2] geäußerte Kritik über die mangelnde formale Präzision Semantischer Netze berücksichtigend, haben alle Knoten und Kanten des Semantischen Netzes eine definierte Semantik, die durch einen Typ definiert ist. Der Typ eines Knotens heißt *Knotentyp*, der Typ einer Kante heißt *Kantentyp*. Alle Knoten und Kanten des Semantischen Netzes besitzen neben ihrem Typ optional einen Namen. Der Typ der Knoten und Kanten wird durch einen Doppelpunkt vom Namen des Knotens und der Kante getrennt. Die folgende Grafik beinhaltet zwei Knoten, die durch eine Kante verbunden werden:

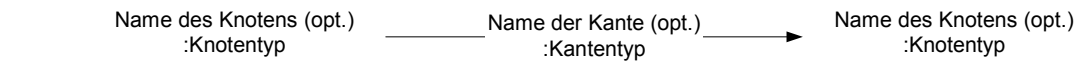


Abbildung 13: Aufbau des Konzeptuellen Schemas als Semantisches Netz

Durch den Typ wird angegeben, um welche Art von Knoten oder Kante es sich handelt. Durch den Namen wird angegeben, um welches bestimmte Exemplar dieses Knoten- oder Kantentyps es sich handelt. In Situationen, in denen der Name des Knotens oder der Kante nicht von Interesse ist, wird ein so genannter anonymer bzw. unbenannter Knoten (bzw. eine Kante) verwendet, der keinen Namen besitzt.

[1] Vgl. [Winston 1992], [Brachman 1979], [Woods 1975], [Reimer 1991], [Sowa 1984: S. 69 ff.], [Findler 1979], [Sowa 1991].

[2] Vgl. [Woods 1975].

Durch das Metamodell des Konzeptuellen Schemas wird die Struktur der Semantischen Netze definiert. Das Metamodell definiert eine Taxonomie für die im Konzeptuellen Schema verwendeten Knoten- und Kantentypen. Durch diese Taxonomie wird eine Systematik festgelegt, in welcher Weise in einem Konzeptuellen Schema die unterschiedlichen Knotentypen durch Kantentypen verbunden werden. In diesem Sinne sind die im Metamodell des Konzeptuellen Schemas definierten Knoten- und Kantentypen die Metatypen der Knoten und Kanten eines spezifischen Schemas. Jedes Element des Semantischen Netzes ist ein Exemplar eines Knoten- oder Kantentyps des Metamodells. Bei der Entwicklung des Metamodells wird das Ziel verfolgt, eine möglichst große Anzahl von Sachverhalten durch vordefinierte Modellprimitive, also durch domänenunabhängige Knoten- und Kantentypen, abzudecken.[1] Verwendet man ein Semantisches Netz, sind allerdings zwei Einschränkungen zu beachten:

- In einem Semantischen Netz werden nur Knoten durch Kanten verbunden. Eine Verbindung von Kanten mit Kanten ist nicht vorgesehen.
- In einem Semantischen Netz sind nur die Knoten informationstragend. Kanten besitzen außer ihrem Typ und ggf. einem Namen keine weiteren Informationen.

Um diese Restriktion zu umgehen, werden auch die Kanten des Konzeptuellen Schemas als Knoten betrachtet. Dies basiert auf dem von CERCONE und SCHUBERT entwickelten Konzept der *Aussagen-* bzw. *Propositionsknoten*.[2] Danach werden auch die Beziehungen als Knoten repräsentiert, die mittels elementarer Kanten die *Konzeptknoten* verbinden. An Stelle der Bezeichnung Propositionsknoten bzw. Propositionsknotentyp wird im Folgenden die Bezeichnung *Beziehungsknoten* bzw. *Beziehungsknotentyp* verwendet. Diese Konzeption hat folgende Vorteile:

- Werden Beziehungen als Beziehungsknoten dargestellt, können diese über weitere Beziehungsknoten miteinander verbunden werden.
- Durch die Repräsentation als Beziehungsknoten sind auch Beziehungen informationstragend.

Für die Beziehung zwischen Externem und Konzeptuellem Schema ergibt sich damit, dass die Objekte und Klassen eines Externen Schemas als Knoten im Konzeptuellen Schema repräsentiert werden. Außerdem werden die Beziehungen eines Externen Schemas im Konzeptuellen Schema als

---

[1] Dies entspricht den Prinzipien eindeutig interpretierbarer Semantischer Netze. Vgl. [Reimer 1991: S. 96 - 99].

[2] Vgl. [Schubert 1976: S. 165], [Schubert 1979: S. 128 ff.], [Cercone 1975], [Cercone 1975b]. Die Propositionsknoten basieren auf der Idee Semantischer Netze, dass jede semantische Beziehung in einen Knoten umgewandelt werden kann, der zwei Knoten mittels zweier elementarer Kanten verbindet. Vgl. [Reimer 1991: S. 101].

Beziehungsknoten repräsentiert. Das folgende Beispiel dient zur Demonstration dieses Prinzips am Beispiel eines UML-Klassendiagramms:

In einem UML-Klassendiagramm als einem Externem Schema ist modelliert worden, dass die Klasse `Abteilungsleiter` mit der Klasse `Sachbearbeiter` durch eine Assoziationsbeziehung verbunden ist (vgl. Abbildung 14 (a)). Im Metamodell des Konzeptuellen Schemas ist definiert, dass Assoziationsbeziehungen eines Externen Schemas als Beziehungsknoten des Typs *ASSOCIATION*[1] im Konzeptuellen Schema abgebildet werden. Dieser Beziehungsknoten erhält zwei Verweise – `StartNode` und `EndNode` – auf exakt zwei Knoten, die er verbindet (vgl. Abbildung 14 (b)). Die Klassen des Externen Schemas werden im Konzeptuellen Schema als Knoten des Typs *CLASS* abgebildet. Der Beziehungsknoten des Typs *ASSOCIATION* ist im Konzeptuellen Schema durch zwei Kanten mit den Knoten des Typs *CLASS* verbunden.

Auf diese Weise ist die Assoziationsbeziehung `Weisungsberechtigung` des Externen Schemas im Konzeptuellen Schema als Beziehungsknoten des Typs *ASSOCIATION* mit dem Namen `Weisungsberechtigung` abgebildet worden. Die Klassen `Abteilungsleiter` und `Sachbearbeiter` des Externen Schemas sind im Konzeptuellen Schema als Knoten des Typs *CLASS* mit den Namen `Abteilungsleiter` und `Sachbearbeiter` abgebildet worden (vgl. Abbildung 14 (c)):

[1] Die Masse der Literatur zum Thema objektorientierte Modellierung ist in Englisch. Aus diesem Grund wurde die Entscheidung getroffen, für die Knoten- und Beziehungsknotentypen sowie die im weiteren Verlauf der Arbeit eingeführten so genannten *Prädikate* englische Namen zu verwenden. Dies geschah nicht mit der Absicht, dem Text eine besondere Exotik zu verleihen, sondern resultierte aus dem Problem, dass für eine Vielzahl von Fachbegriffen keine allgemein akzeptierten Übersetzungen existieren und im Rahmen dieser Arbeit auch keine Neuschöpfungen geprägt werden sollen. Ein weiteres Argument für die gewählte Lösung ist, dass Englisch die allgemein akzeptierte Fachsprache der Informatik darstellt. Der Leser möge die daraus entstehenden, etwas hölzern und künstlich anmutenden Satzkonstruktionen entschuldigen.

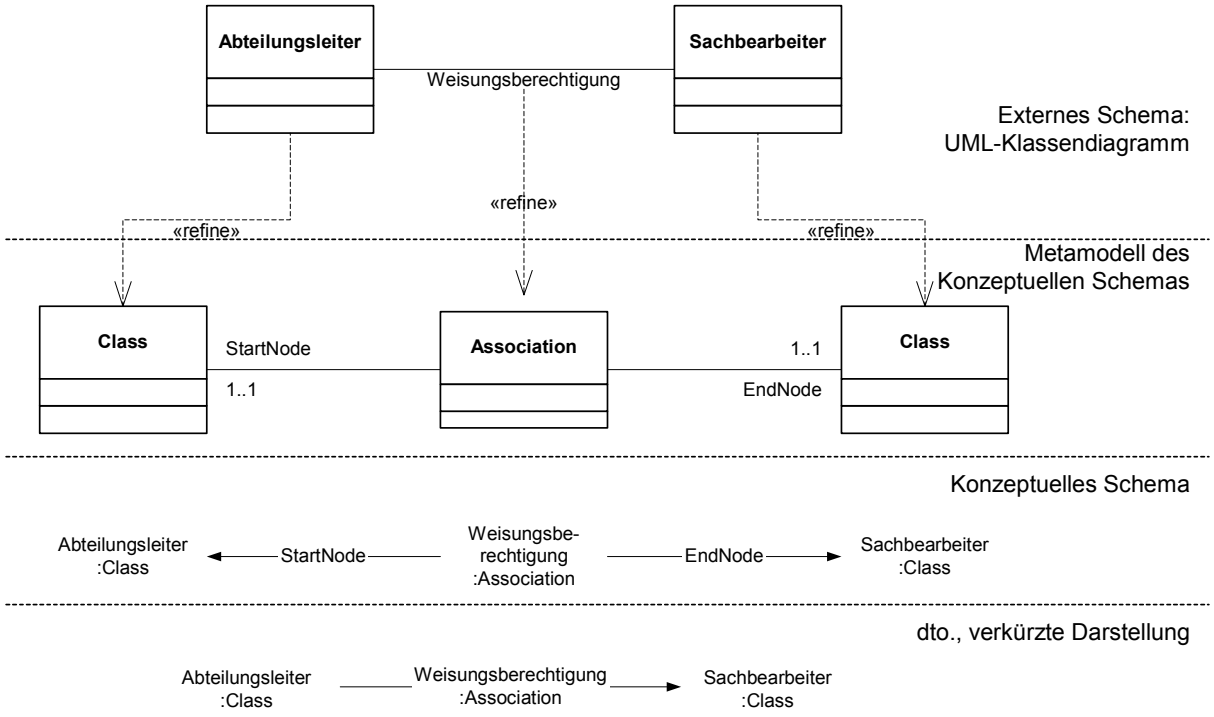


Abbildung 14: Umwandlung von Beziehungen in Knoten[1]

Aus Gründen der Übersichtlichkeit wird im weiteren Verlauf eine kompaktere Notation verwendet. Der Beziehungsknoten und die beiden Kanten werden als eine gerichtete Beziehung dargestellt. Der Knoten, auf den über den Verweis `StartNode` verwiesen wird, wird als Ausgangspunkt der gerichteten Beziehung angesehen. Der Knoten, auf den über den Verweis `EndNode` verwiesen wird, wird als Endpunkt dieser gerichteten Beziehung angesehen (Vgl. Abbildung 14 (d)). Die gerichtete Beziehung beginnt bei `Abteilungsleiter :Class` und endet bei `Sachbearbeiter :Class`.

Wird für das Interne Schema ein objektorientiertes Datenbanksystem verwendet,[2] entsteht eine Objektstruktur, die hier in Form eines UML-Objektdiagramms dargestellt wird. Für jeden Knoten im Konzeptuellen Schema wird ein Objekt erzeugt. Das den Beziehungsknoten abbildende Objekt erhält zwei Referenzen auf die Objekte, die Knoten abbilden:

[1] Nur aus Gründen der Anschaulichkeit wird in der Abbildung die Klasse *CLASS* zweimal dargestellt. Verwendet wurde außerdem der Beziehungstyp Verfeinerung. Vgl. [Rumbaugh 1999: S. 409].

[2] Für diesen Abschnitt wurde exemplarisch das ODBMS JASMINE der Fa. COMPUTER ASSOCIATES untersucht. Vgl. [Linssen 1999c], [Linssen 1999d].

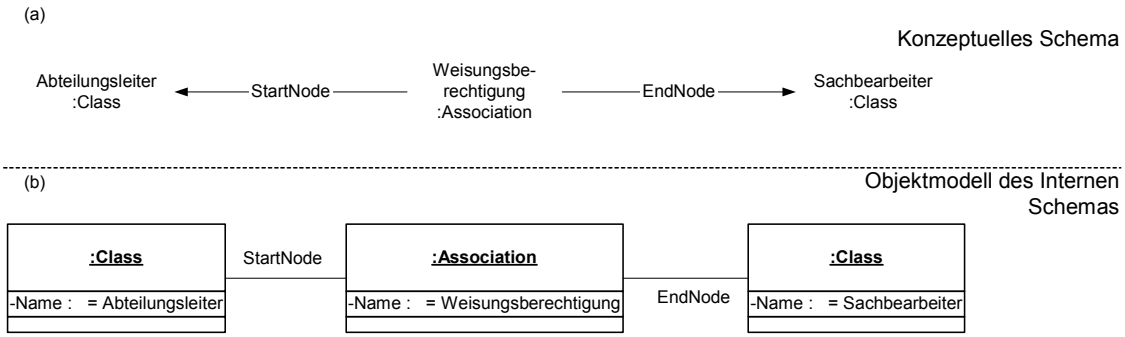


Abbildung 15: Aus dem Abbildungsprozess entstehende Objektstruktur als UML-Objektdiagramm

3.2.3 Ein Beispiel des Abbildungsprozesses

In einem weiteren Beispiel soll demonstriert werden, wie die im Externen Schema verwendeten Skripte im Konzeptuellen Schema und im Internen Schema abgebildet werden. Dieses Beispiel beinhaltet in Form einer didaktischen Reduktion einen Vorgriff auf die noch zu definierenden Metamodelle. Dadurch soll das Verständnis der weiteren Ausführungen erleichtert und insbesondere der in Kapitel 5 dargestellte Abbildungsprozess vom Skriptmodell auf das Semantische Netz des Konzeptuellen Schemas nachvollziehbar werden. Zur Veranschaulichung wird deshalb in diesem Beispiel auch nur eine vereinfachte Form der Skripte verwendet. Zunächst wird zwischen den Spalten Action (*Spalte 3*) und Participant (*Spalte 5*) eine weitere Spalte eingefügt, in der festgelegt wird, welche Art von Beziehung zwischen Initiator und Participant besteht (*Spalte 4*). Die erste Spalte (*Spalte 1*) wird hier noch nicht verwendet und bleibt zunächst leer.

In einem Skript soll dokumentiert werden, dass ein `Abteilungsleiter` einem `Sachbearbeiter` eine Anweisung erteilt:

Spalte 1	*Spalte 2*	*Spalte 3*	*Spalte 4*	*Spalte 5*	*Spalte 6*
<leer>	Abteilungsleiter	erteilt Anweisung	Interaktion	Sachbearbeiter	nimmt Anweisung entgegen

`Abteilungsleiter` (*Spalte 2*) und `Sachbearbeiter` (*Spalte 5*) stehen in einer Interaktionsbeziehung (*Spalte 4*). Sie werden im Konzeptuellen Schema als Knoten des Typs *CLASS* abgebildet. `Erteilt Anweisung` (*Spalte 3*) und `nimmt Anweisung entgegen` (*Spalte 6*) sind

die Aktivitäten, die diese Klassen im Rahmen der Interaktion ausüben. Die Aktivität des Abteilungsleiters wird im Konzeptuellen Schema als Knoten des Typs *ACTION* abgebildet, die Aktivität des Sachbearbeiters wird im Konzeptuellen Schema als Knoten des Typs *OPERATION* abgebildet. Die Knoten für Klassen und Aktivitäten werden im Konzeptuellen Schema durch Kanten verbunden.

Im Rahmen der Abbildung wird überprüft, ob im Konzeptuellen Schema schon zwei Knoten `Abteilungsleiter` und `Sachbearbeiter` existieren. Dies geschieht, indem nach zwei Knoten mit dem Namen `Abteilungsleiter` und `Sachbearbeiter` gesucht wird, die den Knotentyp *CLASS* besitzen. Sind keine Knoten mit diesem Namen und Typ vorhanden, werden sie erzeugt.

Im nächsten Schritt wird geprüft, ob die im Skript verwendeten Aktivitäten schon existieren. Es wird überprüft, ob Knoten mit dem Namen `erteilt Anweisung` (`eA`) und `nimmt Anweisung entgegen` (`nA`) existieren, die außerdem den Knotentyp *ACTION* bzw. *OPERATION* besitzen. Ist dies nicht der Fall, werden sie angelegt und durch eine Kante mit dem jeweiligen Knoten für `Abteilungsleiter` (Al) und `Sachbearbeiter` (`Sb`) verbunden.

Existiert ein Knoten mit dem Namen `eA` und dem Typ *ACTION*, wird überprüft, ob dieser Knoten mit einem Knoten `Al` des Typs *CLASS* eine verbindende Kante besitzt. Ist dies der Fall, benötigt das Semantische Netz keine Erweiterung. Ist der Knoten `eA` mit einem anderen Knoten als `Al` verbunden, wird ein weiterer Knoten mit dem Namen `eA` und dem Typ *ACTION* erzeugt und durch eine Kante mit `Al` verbunden. Dies ist notwendig, weil im objektorientierten Ansatz zwei verschiedene Klassen durchaus Aktivitäten mit identischen Namen besitzen können. Für den Knoten `nA` gilt Entsprechendes.

Die Tatsache, dass die beiden Klassen im Skript interagieren (Spalte4), wird im Semantischen Netz durch eine Kante repräsentiert, welche die Knoten `eA` und `nA` verbindet. Dabei spielt es keine Rolle, ob schon eine verbindende Kante existiert. Ausgangspunkt der Kante ist der Knoten des Typs *ACTION*. Endpunkt der Kante ist der Knoten des Typs *OPERATION*. Die folgende Abbildung zeigt das resultierende Semantische Netz:

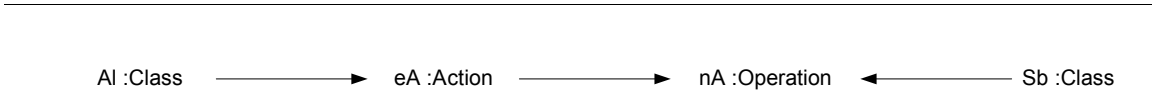


Abbildung 16: Ergebnis der Umformung im Konzeptuellen Modell

An dieser Stelle wird ein Unterschied zwischen dem hier verwendeten Semantischen Netz und üblichen Klassendiagrammen[1] der objektorientierten Modellierung deutlich. In einem Klassenmodell würde eine Beziehung direkt zwischen dem Knoten `Abteilungsleiter` (`Al`) und `Sachbearbeiter` (`Sb`) geknüpft. Die Tatsache, dass diese Beziehung auf der hier im Externen Schema dokumentierten Interaktion und den dabei stattfindenden Aktivitäten beruht, wird nicht explizit abgebildet und ist im Klassenmodell auch nicht ersichtlich.

Die Abbildungsvorschrift vom Konzeptuellen Schema zum Internen Schema ist naturgemäß vom gewählten Basissystem abhängig. Würde beispielsweise ein relationales Datenbankverwaltungssystem verwendet, müsste die Abbildung entsprechend gewählt werden. Die Transformation des Semantischen Netzes in ein Objektmodell eines objektorientierten Datenbankverwaltungssystems ist dagegen relativ nahtlos möglich, weil beide Ansätze auf einer Netzstruktur beruhen. Dagegen führt die Abbildung auf ein Relationenmodell eines RDBMS zu einem Paradigmenbruch mit aufwändigen Umformungsprozessen, weil das Relationenmodell nicht auf einer Netzstruktur, sondern auf Tabellen beruht. In dem folgenden Beispiel wird davon ausgegangen, dass ein objektorientiertes DBMS zur Realisierung des Internen Schemas verwendet wird.

Das Grundprinzip der Abbildung des Konzeptuellen auf das Interne Schema ist, dass für jeden Knoten und für jeden Beziehungsknoten ein Objekt erzeugt wird. Für die Klassenknoten im Semantischen Netz wird ein Objekt des Typs *CLASS* erzeugt. Für die Knoten der Typen *ACTION* und *OPERATION* wird ein Objekt entsprechender Klassen erzeugt. Für die Kante zwischen *ACTION* und *OPERATION* wird ein Objekt des Typs *INTERACTION* erzeugt. Außerdem wird im Konzeptuellen Schema festgelegt werden, dass für die Kanten zwischen *CLASS* und *ACTION* bzw. *OPERATION* ein Objekt des Typs *INTENSION* erzeugt wird.[2] Den für Kanten erzeugten Objekten wird mitgeteilt, welches Objekt im Konzeptuellen Schema Ausgangspunkt der Kante ist, und welches der Endpunkt der Kante ist.

Die Abbildung auf das Interne Schema geschieht in folgender Weise: Im Objektmodell wird geprüft, ob Objekte des Typs *CLASS* für das Attribut Namen den Wert `Al` bzw. `Sb` besitzen. Ist dies nicht der Fall, werden sie erzeugt. Danach wird geprüft, ob Objekte des Typs *ACTION* und *OPERATION* existieren, deren Attribut ‚Name' den Wert `eA` bzw. `nA` besitzen. Ist dies nicht der Fall, werden sie ebenfalls erzeugt. Anschließend wird überprüft, ob diese Objekte durch Objekte des Typs *INTENSION* mit den Objekten `Al` bzw. `Sb` verbunden sind. Diese Objekte repräsentieren die Kante zwischen `Al` und `eA` des Semantischen Netzes. Existiert kein solches Objekt, wird es erzeugt. Dieses Objekt des Typs *INTENSION* wird über Links mit dem `Al`-Objekt und dem `eA`-

[1] Vgl. exemplarisch [Booch 1995], [Coad 1991], [Rumbaugh 1993].

[2] Dieser Vorgriff ist hier notwendig, um den Aufbau des Konzeptuellen Schemas zu verstehen, bevor es im nächsten Kapitel eingeführt wird.

Objekt verbunden.[1] Für die Kante des Semantischen Netzes, welche die Knoten `eA` und `nA` verbindet, wird ein Objekt des Typs *INTERACTION* erzeugt und mit den Objekten `eA` und `nA` über Links verbunden. Es entsteht folgende Objektstruktur:[2]

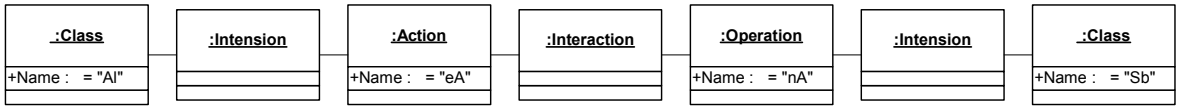


Abbildung 17: Ergebnis der Umwandlung in das Interne Schema

An dieser Stelle sei noch einmal auf den Umstand hingewiesen, dass im Semantischen Netz nicht nach einem beliebigen Knoten gesucht wird, dessen Name `erteilt Anweisung` ist, sondern nach einem, der zusätzlich in Verbindung mit einem Knoten `Abteilungsleiter` steht. Ein solcher Knoten könnte nämlich auch eine Kante von einem Knoten `Firmeninhaber` (`Fi`) besitzen. Würde nun zusätzlich eine weitere Kante vom Knoten `Abteilungsleiter` geknüpft, würde folgendes Semantische Netz entstehen:

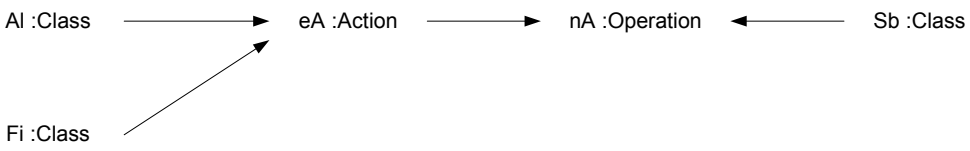


Abbildung 18: Abteilungsleiter und Firmeninhaber üben die gleiche Aktivität aus

Auf diese Weise wird zwar ersichtlich, dass sowohl `Abteilungsleiter` wie auch `Firmeninhaber` die genannte Aktivität ausüben, es ist aber nicht mehr festzustellen, ob hier nun die Anweisung vom `Firmeninhaber` oder vom `Abteilungsleiter` stammt. Diese Information soll aber im Semantischen Netz festgehalten werden.

Im Rahmen des Objektansatzes wird nicht nur abgebildet, ob eine bestimmte Aktivität überhaupt ausgeübt wird, sondern es ist außerdem wichtig, von wem (sprich: von welchem Objekt) die Aktivität ausgeübt wird. Eine eindeutige Zuordnung des zugehörigen Klassenknotens zum Knoten

[1] Für `Angestellter` und `nimmt Anweisung entgegen` analog.

[2] Man beachte, dass es sich um ein UML-Objektmodell handelt, in dem Instanzen und ihre Links untereinander dokumentiert werden. Vgl. [Rumbaugh 1999: S. 60].

eA wäre aber nicht möglich, da der Knoten mit zwei Knoten (Al und Fi) eine verbindende Kante besitzt. Aus diesem Grund werden für beide Personen Aktivitätenknoten angelegt. Selbst wenn tatsächlich beide Personen dem Angestellten eine solche Anweisung geben sollten, ist dies durch zwei Knoten im Semantischen Netz repräsentiert, weil die Aktivität von unterschiedlichen Klassen ausgeführt wird und die Aktivität trotz des gleichlautenden Namens unter Umständen eine unterschiedliche Semantik besitzen kann. Zur Fortführung des Beispiels wird eine weitere Zeile zu dem Skript hinzugefügt:

	Abteilungsleiter	erteilt Anweisung	Interaktion	Sachbearbeiter	nimmt Anweisung entgegen
	Firmeninhaber	erteilt Anweisung	Interaktion	Sachbearbeiter	nimmt Anweisung entgegen

Auf der Ebene der jeweiligen Person (also der Klassenknoten) wird für eine Aktivität nur ein Knoten angelegt, unabhängig davon, wie oft sie verwendet wird. Reagiert der Sachbearbeiter auf die genannte Anweisung des Firmeninhabers und des Abteilungsleiters in gleicher Weise[1], so werden zum Knoten nA zwei Kanten hinführen: einmal für die Interaktion mit dem Abteilungsleiter und eine weitere für die mit dem Firmeninhaber:

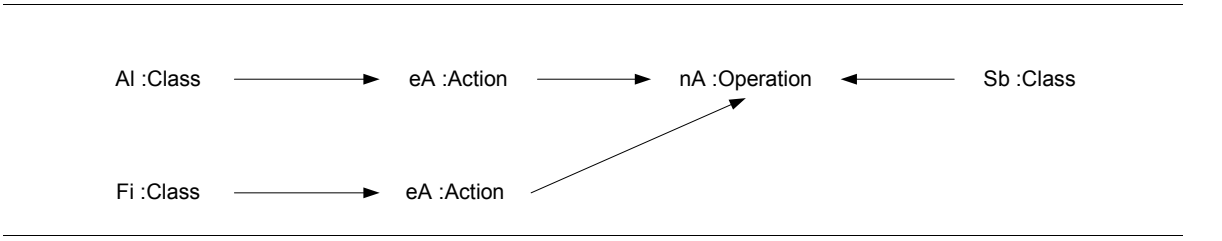


Abbildung 19: Die Aktivität nA des Sachbearbeiters wird mehrfach verwendet

Sollte es dagegen der Fall sein, dass der Sachbearbeiter auf die Anweisung mit unterschiedlichen Operationen reagiert, werden auch auf seiner Seite zwei Knoten anzulegen sein.

[1] Was in diesem Fall angenommen wird.

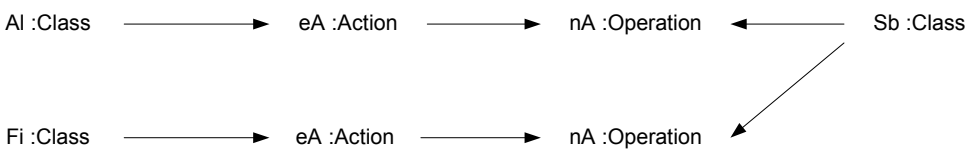


Abbildung 20: Der Knoten Sachbearbeiter besitzt zwei Aktivitäten mit Namen `nA`

Auch das in Abbildung 20 dargestellte Semantische Netz ist sinnvoll. Es kann den Sachverhalt repräsentieren, dass der `Sachbearbeiter` – je nachdem, von wem er die Anweisung erhält – eine andere Aktivität ausführt. In der objektorientierten Programmierung würde man dies als überladene Operation bezeichnen. Diese Möglichkeit wird hier jedoch nicht verfolgt, weil für jede Klasse die Namen der Operationen eindeutig sein sollen. Operationen von Klassen müssen im Semantischen Netz eindeutig sein, wodurch folgendes Netz entsteht:

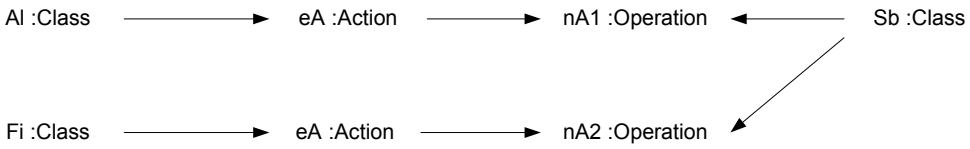


Abbildung 21:Eindeutige Namen für Aktivitäten der Objekte im Semantischen Netz

Geht man schließlich von der Situation aus, dass der `Firmeninhaber` dem `Abteilungsleiter` und dem `Sachbearbeiter` in unterschiedlicher Weise Anweisungen erteilt, dann erhält der `Firmeninhaber` zwei Knoten `erteilt Anweisung` (`eA`1 und `eA`2) und – entsprechend dem festgelegten Verfahren – der `Abteilungsleiter` und der `Sachbearbeiter` einen Knoten `nimmt Anweisung entgegen` (`nA`), die durch entsprechende Kanten miteinander verbunden werden. Es entsteht folgende Anordnung:

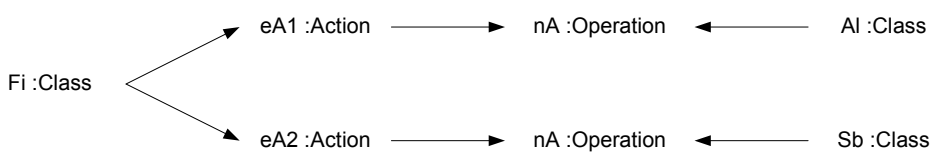


Abbildung 22: Die Semantik der Anweisungserteilung `eA` *ist je nach Adressat unterschiedlich*

Diese Konzeption wurde gewählt, weil bei der objektorientierten Modellierung der Fokus der Betrachtung auf dem Umstand ruht, welche Aktivitäten von welchen Klassen ausgehen, und nicht, welche Aktivitäten überhaupt ausgeführt werden. Darüber hinaus ist es mit der gewählten Konzeption nun trivial festzustellen, welche Objekte in welcher Situation mit welchen Objekten interagieren.

Abschließend sei der Fall dargestellt, dass in einem Skript eine Interaktion zwischen Objekten gelöscht wird. Es wird von der Situation ausgegangen, dass im Rahmen der Modellierung folgender Ablauf modelliert wurde:

1.	Abteilungsleiter Al	erteilt Anweisung eA	Interaktion	Sachbearbeiter Sb	nimmt Anweisung entgegen nA
2.	Sachbearbeiter Sb	erfragt Bestätigung eB	Interaktion	Firmeninhaber Fi	bestätigt Anweisung bA
3.	Firmeninhaber Fi	erteilt Anweisung eA	Interaktion	Sachbearbeiter Sb	nimmt Anweisung entgegen nA

Diese drei Zeilen sollen einen sequentiellen Ablauf abbilden und stellen in rudimentärer Form ein Skript dar. Durch die Nummerierung in der ersten Spalte (Spalte 1) wird eine Reihenfolgebeziehung zwischen den Zeilen des Skripts ausgedrückt. Daraus ergibt sich nach dem bisher entwickelten Verfahren folgendes Semantisches Netz (vgl. Abbildung 23). Aus der ersten Zeile wird (a), aus der zweiten Zeile (b), aus der dritten Zeile wird (c). Aus Gründen der Übersichtlichkeit werden die Kanten, welche die Reihenfolgebeziehung darstellen, hier weggelassen:

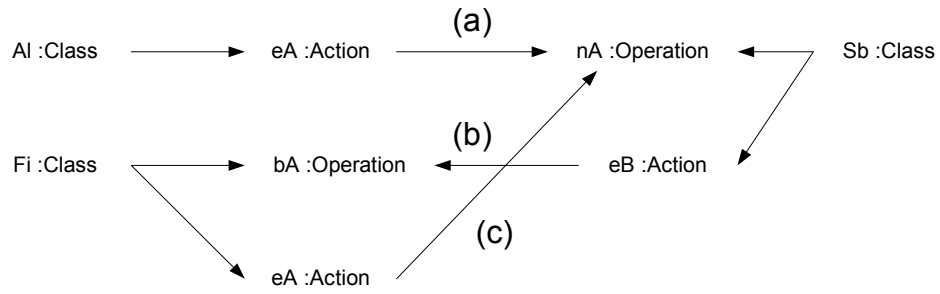


Abbildung 23: Inhalt des Skripts als Semantisches Netz

Die Kanten zwischen *ACTION* und *OPERATION* werden im Internen Schema durch ein Objekt des Typs *EVENTFLOW* verbunden. Dadurch wird im Internen Schema die modellierte Reihenfolgebeziehung festgehalten. Das zugehörige Objektmodell hätte folgende Konfiguration:

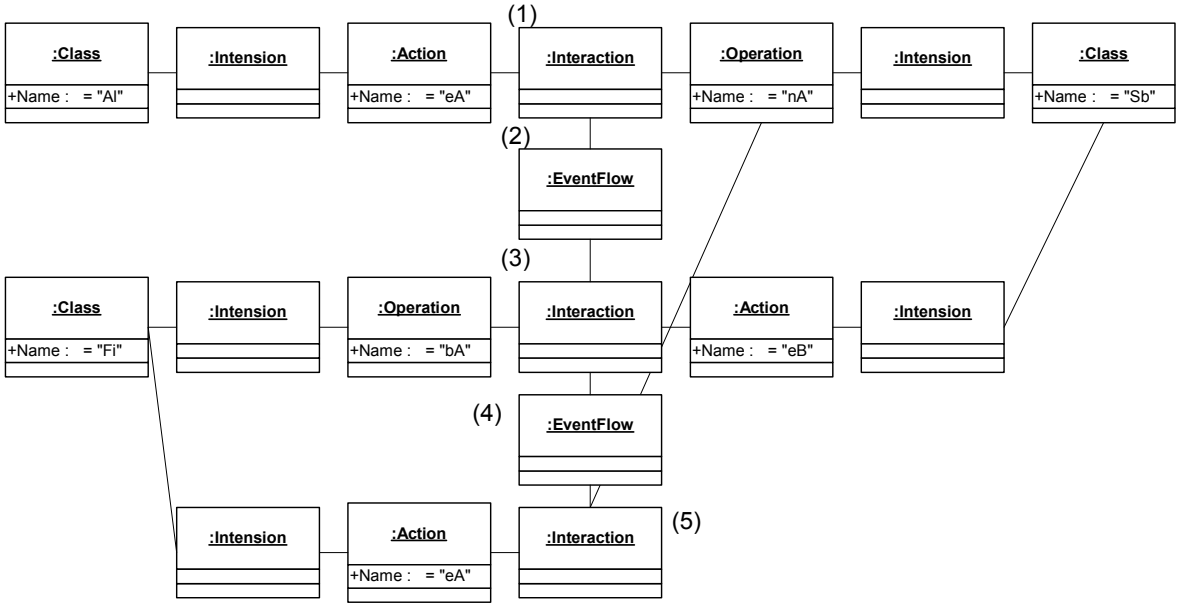


Abbildung 24: Inhalt des Internen Schemas als UML-Objektdiagramm

Für `Al`, `Sb` und `Fi` ist im Internen Schema jeweils ein Objekt des Typs *CLASS* angelegt worden. Für jede Aktivität, die von diesen ausgeübt wird, ist ein Objekt des Typs *ACTION* bzw. *OPERATION* erzeugt worden. Diese Objekte erhalten durch das Attribut `Name` den Bezeichner des Knotens der Aktivität aus dem Semantischen Netz. Die im Skript durch die Nummern explizit modellierte Reihenfolgebeziehung ist im Objektmodell durch zwei Objekte der Klasse *EVENTFLOW* repräsentiert. Diese verbinden im Konzeptuellen Schema die vorangegangene Beziehung mit der folgenden Beziehung. Die Leserichtung für die Reihenfolge der Interaktionen ist von oben nach unten. Deshalb werden die Bezeichnungen `StartNode` und `EndNode` weggelassen.

In dem Skript soll nun die zweite Zeile gelöscht werden. Dabei wird die Semantik der Löschoperation anhand der Internen Ebene demonstriert. Zunächst wird das in Abbildung 24 mit (3) markierte Objekt gelöscht. Der bisher Objekt (2) und Objekt (3) verbindende Link wird mit Objekt (5) verbunden. Objekt (4) wird gelöscht. Die beiden an der Interaktion beteiligten Objekte der Typen *ACTION* und *OPERATION* werden gelöscht, weil sie an keiner Interaktion mehr teilnehmen. Da die beiden *INTENSION*-Objekte (die für Kanten im Konzeptuellen Schema stehen) keine Knoten mehr verbinden, werden auch sie gelöscht. Das Objektmodell nach der Löschoperation gibt die folgende Grafik wieder:

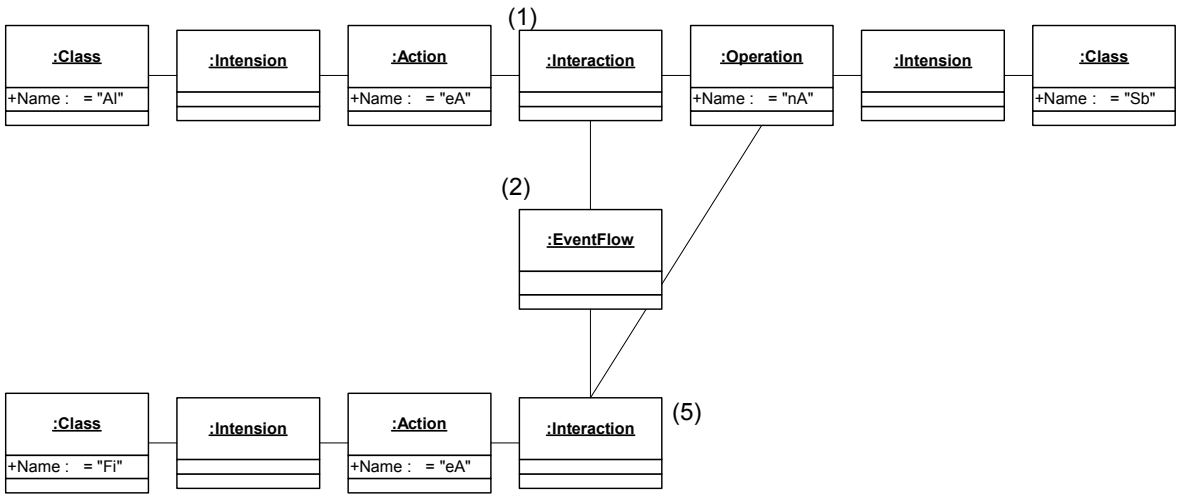


Abbildung 25: Zustand des Objektmodells nach dem Löschen der zweiten Zeile im Skript

Das Beispiel soll die Semantik der Löschoperation verdeutlichen: Solange ein zur Intension eines Konzepts gehörender Knoten im Skript verwendet wird, bleibt er im Internen Schema als Objekt erhalten. Ist dies nicht mehr der Fall, wird er gelöscht. Wird dagegen nur eine von mindestens zwei Referenzen auf ihn gelöscht, bleibt er erhalten. Die folgende Grafik demonstriert dieses Prinzip. Statt der zweiten Zeile ist nun die erste Zeile im Skript gelöscht worden:

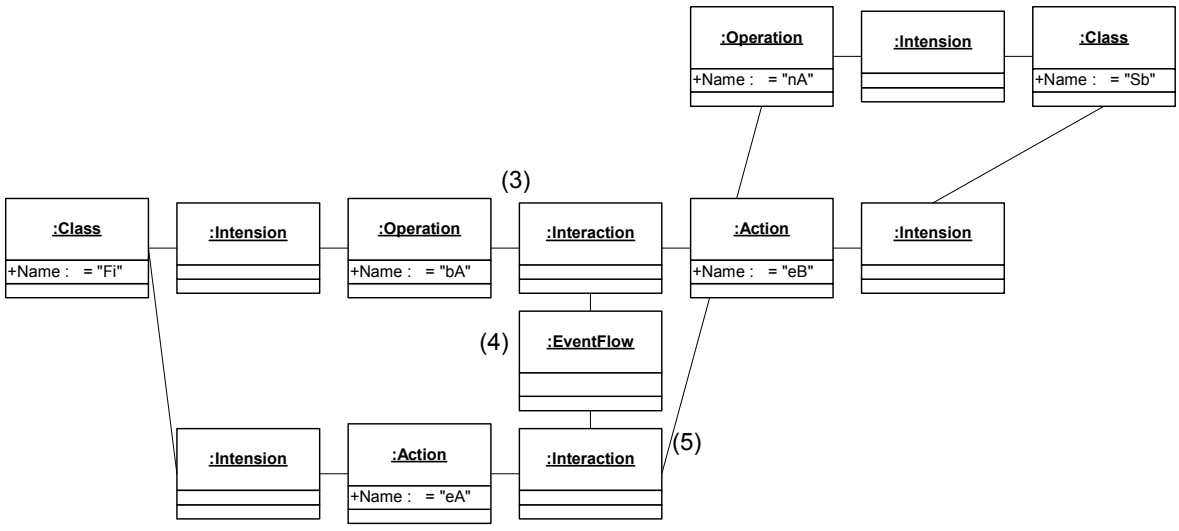


Abbildung 26: Zustand des Objektmodells nach dem Löschen der ersten Zeile im Skript

Das Objekt (1) aus Abbildung 24 wird gelöscht. Das Objekt mit Namen `ea` nimmt an keiner Interaktion mehr teil und wird ebenfalls gelöscht. Damit hat das `Al`-Objekt keine Intension mehr und wird ebenfalls gelöscht. Da die *OPERATION* `nA` des `Sachbearbeiter`-Objekts noch in einer anderen Interaktion verwendet wird, bleibt sie bestehen. Objekt (2) wird ebenfalls gelöscht, weil es eine der zu verbindenden Interaktionen nicht mehr gibt.

3.3 Zusammenfassung

Das letzte Kapitel stellte eine Einführung in den objektorientierten Ansatz dar, der die Metapher des hier entwickelten Modellierungsansatzes ist. Dieses Kapitel stellt eine Übersicht über die Drei-Ebenen-Schema-Architektur des Modellierungsansatzes dar. Das Externe Schema ist die externe Repräsentation des Ansatzes. Hier finden die aus der OBA von RUBIN und GOLDBERG übernommenen Skripte in einer erweiterten und formalisierten Form Verwendung. Das im nächsten Kapitel entwickelte Konzeptuelle Schema stellt das Begriffssystem der Metapher in Form eines Metamodells dar. Das Interne Schema stellt die softwaretechnische Realisierung des Konzeptuellen Schemas dar. Der Transformationsprozess zwischen den Schemata wurde exemplarisch demonstriert. Im übernächsten Kapitel wird auf der Grundlage des Metamodells des Konzeptuellen Schemas das Metamodell der Repräsentation entwickelt.

4 Das Konzeptuelle Schema

„Worüber man nicht sprechen kann, darüber muss man schweigen."
Wittgenstein, Ludwig: Tractatus logico-philosophicus. [1]

Die Struktur des Konzeptuellen Schemas wird durch dessen Metamodell festgelegt.[2] Dieses Metamodell wird durch UML-Klassendiagramme definiert und ist das Begriffssystem der verwendeten Metapher. Das Metamodell besteht aus Metaklassen und Metabeziehungen. Die Knoten eines Konzeptuellen Schemas sind die Exemplare der (Meta-) Klassen aus dem Metamodell des Konzeptuellen Schemas. Auch die Kanten eines Konzeptuellen Schemas sind Exemplare der (Meta-) Klassen aus dem Metamodell des Konzeptuellen Schemas.[3]

Alle Elemente des Konzeptuellen Schemas heißen Modellelemente und sind Exemplare der Metaklasse *MODELELEMENT* des Metamodells des Konzeptuellen Schemas.[4] Diese Metaklasse wird verwendet, um Eigenschaften, die allen Elementen eines Modells gemeinsam sind, zentral zu definieren. Hier werden für alle Modellelemente die Eigenschaften `Name`, `Description`, `Stpe` und `Specification`[5] definiert. Durch das Attribut `Name` kann jedes Modellelement einen Namen erhalten. Durch `Description` kann jedes Modellelement optional eine natürlichsprachige Beschreibung erhalten. Durch `Specification` kann jedes Modellelement optional eine Spezifikation in einer formalen Spezifikationssprache erhalten. `Specification` wird verwendet, um Modellelementen beispielsweise einen Ausdruck in der OBJECT CONSTRAINT LANGUAGE

[1] [Wittgenstein 1994: S. 46].

[2] Zur Verwendung von Metamodellen in der objektorientierten Modellierung vgl. [Blaha 1992], [Züllighoven 1998: S. 19 ff.].

[3] Im weiteren Verlauf werden an Stelle der Begriffe Metaklasse oder Metatyp auch die Begriffe Klasse oder Typ verwendet, wenn aus dem Kontext deutlich wird, dass damit die Metatypen des Konzeptuellen Schemas gemeint sind.

[4] Zum Vergleich mit anderen Metamodellen des Objektansatzes sei der Leser auf [OMG 1999], [Firesmith 1997] und [Frank 1998] verwiesen.

[5] GRAHAM verwendet für solche Eigenschaften von Modellelementen den Begriff *Facette*. Vgl. [Graham 1998: S. 92]. Eigenschaften und Prädikate der Typen des Metamodells werden im weiteren Verlauf des Textes durch die Verwendung dieses Fonts hervorgehoben.

(OCL) oder den ACTION SEMANTICS[1] zuordnen zu können.[2] Durch `Stpe` kann – entsprechend der UML – jedem Modellelement einen Stereotyp zugeordnet werden.[3] Diese Eigenschaften sind im Metamodell Attribute des Metatyps *MODELELEMENT*.[4] Zu jedem Attribut existiert eine Operation, die den Wert dieses Attributs setzt.[5] Diese Operationen der Metaklassen werden im weiteren Verlauf als *Prädikate*[6] bezeichnet. Die aktuellen Argumente der Prädikate werden als *Terme* bezeichnet. Ein Prädikat mit seinen Termen stellt ein *Faktum* dar, d. h. eine Tatsache, die für das entsprechende Modellelement gilt.[7]

Es wird davon ausgegangen, dass ein Knoten vom Metatyp *MODELELEMENT* mit den Werten `Name = „IrgendeinElement“`, `Description = „Einfach nur ein Knoten“`, und `Stpe = „Abstraktes Konzept“` existiert. In der Notation der UML würde dies folgender Darstellung entsprechen:

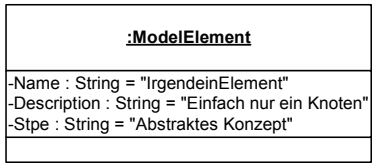


Abbildung 27: Ein Modellelement als UML-Objektdiagramm

[1] Vgl. [OMG 2001b], [OMG 2002].

[2] Vgl. [Warmer 1999].

[3] Vgl. [OMG 1999: S. 2-63 ff.]. Dies wird auch als *leichtgewichtige Metamodellierung* bezeichnet. Vgl. [Atkinson 2000: S. 34]. In der neusten Version der UML können auch mehrere Stereotype zugeordnet werden. Vgl. [OMG 2001: S. 2-81, Abb. 2-19]. Auf die Verwendung von Stereotypen wird im weiteren Verlauf der Arbeit noch ausführlich eingegangen.

[4] Eine Alternative zur hier gewählten Konstruktion wäre die Abbildung der Attribute `Name`, `Description`, `Specification` und `Expression` als eigenständige Knotentypen im Konzeptuellen Schema. Diese Alternative führt aber zu einer unnötigen Komplexitätssteigerung des Konzeptuellen Schemas. Aus diesem Grund wurde davon abgesehen.

[5] Solche Operationen werden üblicherweise als *Set-Methoden* oder *Setter* bezeichnet.

[6] Als Prädikat wird eine Bestimmung von Gegenständen durch einen sprachlichen Ausdruck bezeichnet. [Duden 1993]. Orientiert man sich an der Terminologie der Programmiersprache PROLOG, müsste es eigentlich *Prädikatensymbol* heißen. Vgl. [Rechenberg 1999: S. 568]. Der Begriff Prädikat wird verwendet, um die umständliche Formulierung *Operation der Metaklasse* vermeiden zu können.

[7] Orientiert man sich an der Terminologie der Programmiersprache PROLOG, sind es *Klauseln*. Vgl. [Rechenberg 1999: S. 568]. Im Ansatz der Semantischen Netze, die den Ausgangspunkt der Entwicklung dieses Metamodells darstellten, stellen die Prädikate *Eigenschaftsklassen* und die Terme die aktuelle Merkmalsausprägung der Eigenschaft dar. Vgl. [Reimer 1991: S. 83].

Der Knoten ist ein Exemplar bzw. ein Objekt des Metatyps *MODELELEMENT*. Dieses Objekt besitzt die Werte „`IrgendeinElement`“ für das Attribut `Name`, „`Einfach nur ein Knoten`“ für das Attribut `Description` und „`Abstraktes Konzept`“ für das Attribut `Stpe`. Der Wert des Attributs `Name` wird vor den Knotentyp geschrieben und als Objektidentifizierer verwendet. Dies entspricht der Notation, wie sie in der UML für Objekte verwendet wird:

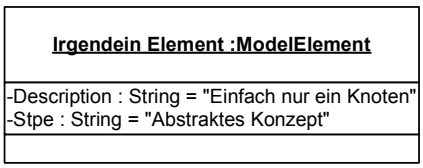


Abbildung 28: Ein Knoten mit einem Namen

Die Werte der Attribute eines Knotens werden durch seine Prädikate gesetzt. Für `Description` wird das Prädikat `Desc()`, und für `Stpe` wird das Prädikat `Stpe()` verwendet. Zur Vereinfachung werden im weiteren Verlauf für die Terme die Anführungszeichen weggelassen. Auf diese Weise erhält man folgende Schreibweise:

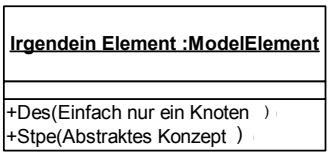


Abbildung 29: Die Attribute eines Knotens werden durch Prädikate und Terme gesetzt

Das Objekt in Abbildung 29 wird wie folgt gelesen: Das Objekt hat den Metatyp (d. h. die Klasse) `ModelElement` und den Namen `IrgendeinElement`. Darüber hinaus gelten folgende Fakten: Das Objekt hat die Beschreibung `Einfach nur ein Knoten` und das Stereotyp `Abstraktes Konzept`. Alle Klauseln (d. h. alle Prädikate und ihre Terme) stehen in einer logischen Konjunktionsbeziehung. Im Metamodell des Konzeptuellen Schemas erhalten die Terme der

Prädikate einen formalen Namen und einen Typ.[1] Wie in Abbildung 30 zu sehen ist, haben die Terme der Prädikate `Desc()` und `Stpe()` den Typ Zeichenkette (`String`).

Im weiteren Verlauf wird nicht mehr auf die Attribute der Metatypen eingegangen, sondern nur auf die Prädikate, durch die ein Attribut einen Wert erhält. Die Terme der Prädikate sind die Argumente, durch die die verborgenen Attribute des Metatyps verändert werden. Wie dies geschieht, ist abhängig von der Realisierung des Metamodells im Internen Schema durch ein Softwaresystem. Dies ist nicht Bestandteil des Modellierungsansatzes.

4.1 Muster des Metamodells

Das Metamodell des Konzeptuellen Schemas basiert auf einem Muster, welches die Verbindung von Knotentypen durch Beziehungsknotentypen definiert. Von der Metaklasse *MODELELEMENT* werden die Metaklassen *NODE* und *ASSOCIATIVELINK* spezialisiert. Alle Knoten des Konzeptuellen Schemas basieren auf der Metaklasse *NODE*. Alle Beziehungsknotentypen (d. h. Kanten) des Semantischen Netzes basieren auf der Metaklasse *ASSOCIATIVELINK*. Das bedeutet, dass die unterschiedlichen Typen von Knoten von *NODE,* und die unterschiedlichen Typen von Beziehungsknoten von *ASSOCIATIVELINK* spezialisiert werden. Auf der Basis dieser Klassen wird die Struktur des Metamodells erläutert. Wie in Abbildung 30 zu sehen ist, abstrahiert jeder Beziehungsknoten von einer gerichteten Beziehung zwischen einem Ausgangs- und einem Endknoten. Alle Beziehungsknoten sind mit exakt einem Ausgangsknoten `StartNode` und mit exakt einem Endknoten `EndNode` verbunden. Alle Knoten stehen über Beziehungsknoten als Ausgangsknoten und Endknoten mit anderen Knoten in Verbindung: Sie können mehrere direkte Nachfolgerknoten besitzen, mit denen sie über einen Beziehungsknoten verbunden sind. Sie können mehrere direkte Vorgängerknoten besitzen, mit denen sie über einen Beziehungsknoten verbunden sind. Dies wird als *Fan-In* und *Fan-Out* bezeichnet.[2] Jeder Knoten muss mit mindestens einem anderen Knoten verbunden sein.[3] Als von *MODELELEMENT* spezialisierte Metaklassen besitzen sie die schon dort definierten Prädikate. Sie fügen den Klassenbeschreibungen jeweils die für sie definierten Beziehungen hinzu. Dadurch gilt: Jeder Beziehungsknotentyp erreicht über `StartNode` seinen Ausgangsknoten und über `EndNode` seinen Zielknoten. Alle Knotentypen, die direkt oder indirekt von *NODE* spezialisiert werden, erreichen über `Fan-Out` und `Fan-In` die Beziehungsknoten, mit denen sie verbunden sind. Durch die in Abbildung 30 verwendeten

[1] Siehe Anhang der Arbeit.

[2] Vgl. [Page-Jones 1995].

[3] Dies wird durch den OCL Ausdruck `(self.Fan-Out->size + self.Fan-In->size)>0` festgelegt. Vgl. [Warmer 1999].

Kardinalitäten[1] wird deutlich, dass keine isolierten Knoten und keine Kanten existieren, die nicht mit zwei Modellelementen verbunden sind. Die Knoten *NODE*, *ASSOCIATIVELINK* und *MODELELEMENT* sind abstrakte Metaklassen. Es gibt in einem Konzeptuellen Schema keine Knoten dieser Typen, sondern nur von den jeweiligen Spezialisierungen. Damit ergibt sich folgendes Muster für das Metamodell des Konzeptuellen Schemas:

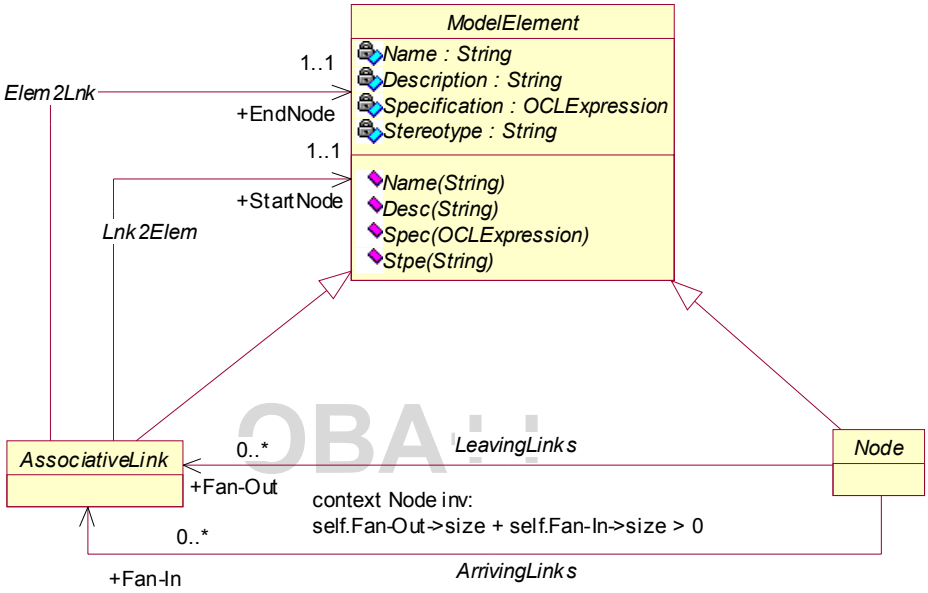


Abbildung 30: Das Muster für das Semantische Netz des Konzeptuellen Schemas

Alle verwendeten Knoten- und Beziehungsknotentypen basieren auf diesem Muster. Die jeweils zwischen dem Knotentyp und dem Beziehungsknotentyp definierten Beziehungen realisieren die Beziehungen `Elem2Lnk`, `Lnk2Elem`, `LeavingLinks` und `ArrivingLinks`.[2]

Die Assoziationen zwischen *ASSOCIATIVELINK* und *MODELELEMENT* sind notwendig. Der `Start-` und `EndNode` eines Beziehungsknotens müssen kein Knoten sein, sondern können selbst Beziehungsknoten des Metatyps *ASSOCIATIVELINK* sein.[3] Auf diese Weise wird es möglich, dass

[1] Der Begriff Kardinalität wird hier an Stelle des in der UML üblichen Begriffs Multiplizität verwendet.

[2] Das hat zur Folge, dass bei der Realisierung des Metamodells durch eine objektorientierte Programmiersprache oder ein objektorientiertes Datenbankverwaltungssystem die Beziehungen `Elem2Lnk`, `Lnk2Elem`, `LeavingLinks` und `ArrivingLinks` nicht implementiert werden. Sie stellen praktisch Meta-Meta-Beziehungen dar, um die Struktur des Metamodells zu erläutern.

[3] Diese Aussage trifft für die Beziehungsknoten des Typs *DEPENDENCY* zu. Vgl. Abschnitt 4.3.6.

Beziehungsknoten durch Beziehungsknoten verbunden werden können, wie die folgende Grafik veranschaulicht:

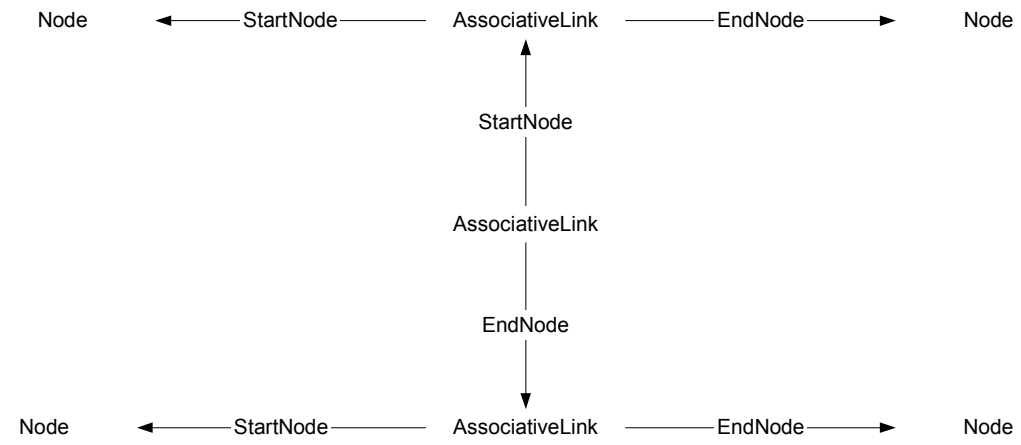


Abbildung 31: Prinzip der Verbindung von Beziehungsknoten über Beziehungsknoten[1]

Das für die Definition von *MODELELEMENT*, *NODE* und *ASSOCIATIVELINK* verwendete Prinzip, gemeinsame Eigenschaften von Knoten und Kanten in Klassen zusammenzufassen, ist bestimmend für die im weiteren Verlauf definierte Taxonomie der Knoten- und Kantentypen. Die Taxonomie basiert auf der Semantik der Vererbungsbeziehung der UML.[1] Weitere Kriterien für die Bildung der Taxonomie sind folgende Gesichtspunkte:

- Welche Primitive des Objektansatzes (Objekt, Klasse, Operation, Attribut, Vererbung, Assoziation, ...) im Konzeptuellen Schema entweder als Knoten und Kanten abgebildet werden.
- Welche Eigenschaften diese Primitive besitzen und welche Eigenschaften mehrere Primitive gemeinsam haben.
- Wie sich die Primitive in Spezialisierungsbeziehungen anordnen lassen.

Bei der Bildung des Metamodells des Konzeptuellen Schemas werden folgende Restriktionen beachtet:

- Entlang der Spezialisierungsbeziehungen der Beziehungsknotentypen werden auch die über `StartNode` und `Endnode` verbundenen Knotentypen spezialisiert oder beibehalten, aber nicht verallgemeinert.

[1] Das Beispiel dient der Erläuterung. Es beinhaltet kein wohlgeformtes Konzeptuelles Schema.

- Die Kardinalität von `StartNode` und `EndNode` bleibt immer gleich.
- Die Kardinalität von `Fan-In` und `Fan-Out` wird entlang der Spezialisierungsbeziehung der Knotentypen nie verändert.

Die Definition des Metamodells des Konzeptuellen Schemas geschieht in zwei Stufen: Die Knoten und Kanten werden in diesem Kapitel eingeführt. Dafür erfolgt – bis auf wenige Ausnahmen – keine Definition von Prädikaten, da sie auf dieser elementaren Ebene nicht nachvollziehbar sind. Das Skriptmodell verwendet die in diesem Kapitel definierten Knoten und Kanten und fügt die Prädikate und deren Terme für die unterschiedlichen Metatypen hinzu.[2]

4.2 Knoten

Bei den Knoten werden die Metaklassen der Konzepte und die Metaklassen der Intension unterschieden. Konzepte sind die Dinge, die in der Modellierung abgebildet werden. Die Intension umfasst die Eigenschaften, mit denen die Dinge beschrieben werden. Die hierfür verwendeten Metaklassen sind Spezialisierungen der Metaklasse *NODE*.

4.2.1 Konzeptknoten

Für Klassen, Objekte und Rollen wird als gemeinsame Abstraktion der Begriff *Konzept* verwendet. *Konzept* stellt hier den allgemeinsten Begriff für alle Dinge der Welt dar, unabhängig davon, ob sie konkrete oder abstrakte Dinge beschreiben.[3] Klasse, Rolle und Objekt sind *epistemische Primitive*[4], die nicht an einen bestimmten Diskursbereich gebunden sind. Konkrete Dinge existieren in der Welt, abstrakte Dinge haben keine reale Existenz, sondern repräsentieren konkrete Dinge, Ideen

---

[1] Vgl. [Rumbaugh 1999: S. 299].

[2] Aus diesem Grund wird in diesem Kapitel noch nicht auf die Prädikate der Metaklassen eingegangen. Sie werden in Abschnitt 5.2.6 eingeführt und in Kapitel 6 verwendet. Im Kapitel 11 (Anhang 3: Die Schnittstellen der Klassen des Metamodells) finden sich die Schnittstellen aller verwendeten Klassen des Konzeptuellen Schemas.

[3] Vgl. [Alhir 1998: S. 41].

[4] Ein epistemisches Primitiv ist wie folgt definiert: *„Ein epistemisches Primitiv steht für eine Klasse gleichartiger Sachverhalte, die so allgemein sind, dass ihr Auftreten nicht an einen bestimmten Diskursbereich (oder eine Klasse bestimmter Diskursbereiche) gebunden ist – epistemische Primitive sind also domänenunabhängig.“* [Reimer 1991: S. 15]. Epistemische Primitive sind domänenunabhängig, weil sie unabhängig von einem bestimmten Weltausschnitt sind. [a. a. O.: S. 81]. Die Verwendung dieses Begriffs im Rahmen der Informatik geht auf BRACHMAN [Brachman 1979: S. 35] zurück.

oder Vorstellungen. Für alle Konzepte wird die gemeinsame Metaklasse *CONCEPTNODE* definiert. Diese ist eine Spezialisierung der Metaklasse *NODE*. Im Konzeptuellen Schema wird zwischen drei verschiedenen Arten von Konzeptknoten unterschieden:

Eine *Klasse*[1] ist ein Konzept, dessen Extension[2] kein, ein oder mehrere Objekte enthält. Sie ist umgangssprachlich eine Gruppe von Dingen oder Begriffen mit gemeinsamen, sich von anderen unterscheidbaren Merkmalen.[3] Dinge, Ideen, Begriffe oder auch Prozesse[4] sind Klassen. Klassen werden im Semantischen Netz als Exemplare des Metatyps *CLASS* repräsentiert, der über die Metaklasse *CONCEPTNODE* von der Metaklasse *NODE* abgeleitet ist.

Objekte sind Individualkonzepte[5] und implizit immer die Extension einer Klasse. Man bezeichnet Klassen als *Muster, Stanzform, Blaupause* bzw. allgemeine Beschreibung für Objekte. Als *Objekte* bezeichnet man dagegen die Exemplare, Ausprägungen oder Instanzen[6] einer Klasse. Die Extension eines Objekts ist das Objekt selbst.[7] Objekte sind ebenfalls Konzeptknoten. Sie werden als Exemplare des Metatyps *OBJECT* repräsentiert, der über die Metaklasse *CONCEPTNODE* von der abstrakten Metaklasse *NODE* abgeleitet ist.

Die *Objektmigration* von Klassen bedeutet, dass ein Objekt die Zugehörigkeit zu einer Klasse wechseln kann.[8] Diese Veränderung der Zugehörigkeit zu einer Klasse wird im OA allgemein nicht unterstützt.[9] Stattdessen wird das Konzept der *Rolle* verwendet. Mit Hilfe von Rollen werden Aussagen über die Art und Weise der Verwendung der Objekte einer Klasse repräsentiert. Beispielsweise kann eine Klasse Person sowohl die Rolle Antragsteller, als auch Steuerpflichtiger, Ehepartner usw. übernehmen. Man spricht davon, dass die Person als Antragsteller oder Steuerpflichtiger auftritt bzw. diese Rollen übernimmt.[10] Diese Rollen sind je nach Situation

---

[1] Eine ausführliche Erläuterung des Klassenbegriffs findet sich in [Martin 1995] und [Booch 1995].

[2] Zu den Begriffen *Extension* und *Intension* siehe [Martin 1995: S. 15, 17], [Quibeldey-Cirkel 1994: S. 82] oder [Reimer 1991: S. 17].

[3] Vgl. [Duden 1993].

[4] In der UML werden Prozesse durch sog. *aktive Klassen* repräsentiert, d. h., sie besitzen einen eigenen Kontrollstrang (engl.: *thread-of-control*). Vgl. [OMG 1999: S. 2-27].

[5] Vgl. [Reimer 1991: S. 19].

[6] Die Verwendung des Begriffs *Instanz* geht auf eine Fehlübersetzung des englischen Begriffes "*instance*" zurück, der sich als Fachjargon im Deutschen eingebürgert hat.

[7] Vgl. [Helbig 1996: S. 110].

[8] Vgl. [Züllighoven 1998: S. 377].

[9] Vgl. [Züllighoven 1998: S. 35], [Frank 1998c: S. 13]. Vgl. auch die Fußnote zum Thema *Reklassifikation* auf Seite 109.

[10] Vgl. [Riehle 2000: S. 33].

unterschiedlich und können wechseln.[1] Rollen sind wie ein dynamischer Typ zu verstehen, den Objekte zeitweilig übernehmen können.[2] In dieser Rolle zeigen die Objekte unterschiedliches (rollenspezifisches) Verhalten, bleiben aber Exemplar derselben Klasse. So verhält sich ein Mitarbeiterobjekt in der Rolle des Vorgesetzen anders als in der Rolle des Untergebenen. Auf diese Weise lässt sich durch das Rollenkonstrukt die Notwendigkeit zur *Reklassifikation* reduzieren.[3]

Rollen werden – im Gegensatz zur UML[4] – im Konzeptuellen Schema als eigenständiger Metatyp repräsentiert. Die Rolle eines Objekts hat nicht nur Zugriff auf die Schnittstelle des zugehörigen Objekts, sondern unter Ausschaltung des Geheimnisprinzips auch auf dessen Zustand. Dies wird durch die Beobachtung begründet, dass eine Person in einer Rolle immer noch die gleiche Person – also ein und dasselbe Objekt – ist. Aus diesem Grund werden hier Rollen im Konzeptuellen Schema auch nicht als eine besondere Form von Objekten behandelt, die auf speziellen Rollenklassen basieren.[5] Der Metatyp *ROLE* ist *per definitionem* keine Klasse. Er ist über die Metaklasse *CONCEPTNODE* von der abstrakten Metaklasse *NODE* abgeleitet.

Damit ergibt sich im Metamodell des Konzeptuellen Schemas für Konzeptknoten folgendes Teilmodell:[6]

[1] Diese Interpretation des Rollenbegriffs folgt der in der OML (vgl. [Henderson-Sellers 1998: Appendix E; Stichwort „Rôles“]). Diese Verwendung des Rollenbegriffs geht auf die Konzeption von KRISTENSEN [Kristensen 1996] zurück. Ähnliche Ideen finden sich auch in der Programmiersprache KL-ONE aus dem Bereich der künstlichen Intelligenz [Brachman 1985] und in der Methode OOram [Reenskaug 1996: S. 43 ff.]. Ähnlich ist auch die Verwendung des Rollenbegriffs im Bereich der Wissensrepräsentation [Reimer 1991: S. 106], des Geschäftsprozessmanagements [Scheer 1998: S. 57] und des Workflowmanagements [WfMC 1996: S. 58].

[2] Vgl. [Züllighoven 1998].

[3] *Reklassifikation* bedeutet, dass ein Objekt der Klasse Person zu einem Objekt der Klasse Angestellter transformiert werden müsste, wenn es (das Objekt) Angestellter einer Firma wird. Dies wird nach [Odell 1992] auch als dynamische Klassifikation bezeichnet. Reklassifikation wird in nahezu allen objektorientierten Ansätzen – mit Ausnahme von [Martin 1999] – untersagt. Durch die Verwendung von Rollen ist auch eine feinere Zuordnung von Rechten und Fähigkeiten möglich, die ansonsten auf der Ebene von Klassen erfolgen müsste. Vgl. zu diesem Thema BOCK und ODELL [Bock 1998], GOTTLOB, SCHREFL und RÖCK [Gottlob 1996], FRANK [Frank 1997c], sowie SCIORE [Sciore 1989].

[4] [Rumbaugh 1999: S. 85, 164, 194, 203, 250, 414]. Zu den unterschiedlichen Bedeutungen des Begriffs Rolle in der UML vgl. [Hitz 1999: S. 56].

[5] Im Gegensatz zur Konzeption von FRANK. Vgl. [Frank 1997c]. Siehe auch den Abschnitt über die Beziehung zwischen Rollen und Klassen (Abschnitt 4.3.2.4).

[6] Vergleiche den ähnlichen Entwurf der OML Version 1.1 in [Henderson-Sellers 1998: S. 22]. Abweichend davon ist das Metamodell der UML Version 1.3a, welches Klassen, Datentypen, Komponenten, Schnittstellen und Hardware-Knoten unterscheidet. Vgl. [OMG 1999: S. 2-17].

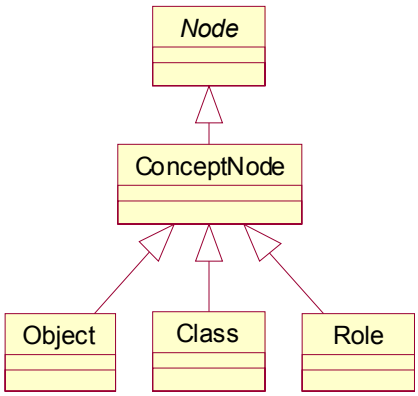


Abbildung 32: Teilmodell des Konzeptuellen Schemas für Konzeptknoten

4.2.2 Intensionsknoten

Die *Intension* beschreibt die *Charakteristika* bzw. *Merkmale* einer Klasse.[1] Zunächst ist zu untersuchen, welche Charakteristika zur Beschreibung von Klassen verwendet werden.[2]

Attribute beschreiben inhärente Eigenschaften, Qualitäten oder Besonderheiten von Konzepten.[3] Mit Attributen werden die statischen, zeitinvarianten Merkmale eines Konzepts beschrieben, die das so genannte *Gedächtnis* des Objekts darstellen.[4] In der objektorientierten Modellierung werden sie als Abstraktionen der Informationen verwendet, die ein Objekt besitzt.[5] Die Darstellung der Informationen hat die Form von Daten.[6] Die Lebensdauer von Attributen stimmt mit dem des Konzepts, dessen Attribut sie sind, überein: Etwas wird als Attribut eines Konzepts angesehen, wenn zwischen ihm und dem Konzept, welches es beschreibt, eine inhärente referentielle Integrität

---

[1] Mit Intension wird in der Literatur eine *„vollständige und intensive Definition“* einer Abstraktion bezeichnet. Vgl. [Firesmith 1995: S. 219]. Durch die Verwendung des Begriffe Intension wird eine weitere Definition angestrebt, als dies über den ähnlich positionierten Begriff *„Intent“* in der UML der Fall ist. Vgl. [Rumbaugh 1999].

[2] Vgl. [Firesmith 1995], [Firesmith 1997], [Henderson-Sellers 1998], [Rumbaugh 1999], [Booch 1999].

[3] [Kappel 1996: S. 25 f.], [Webster 1993].

[4] [Schienmann 1997: S. 38].

[5] Die Norm DIN 44300 definiert den Begriff der *Information* umgangssprachlich als Kenntnis über Sachverhalte und Vorgänge. Vgl. [Schmitz 1992: Sp. 958]. Vgl. auch Seite 11.

[6] Vgl. [Ehrich 1989: S. 1]. Von Daten spricht man, wenn Informationen eine einheitliche Struktur besitzen, aus einer vordefinierten Menge von Zeichen bestehen und in einer maschinell zu verarbeitenden und speichernden Form vorliegen. Vgl. [Krüger 1994: S. 143], [Schmidt 1985: S. 76].

besteht.[1] Ist die Existenz eines Attributs unabhängig von der Existenz des Konzepts, dessen Attribut es sein soll, so stellt es kein Attribut dar, sondern ein eigenständiges Konzept. Ein Attribut gehört zur Intension genau einer Klasse und kann nicht gleichzeitig zur Intension einer anderen Klasse gehören. Daraus folgt, dass mehrere Klassen Attribute mit dem gleichen Namen besitzen können, die dann aber nicht identisch sind. Wenn zwei Klassen X und Y ein Attribut a besitzen, handelt es sich um zwei Attribute mit dem gleichen Namen a.[2] Üblicherweise werden sie durch X.a und Y.a referenziert. Attribute sind demnach als lokal zu betrachten. Auf sie kann nicht von außen

---

[1] Diese Festlegung vermeidet zwei Probleme des Objektansatzes: 1. Sind Attribute Klassen? 2. Was unterscheidet Werte (= Attribute) von Dingen (= Klassen)? DE CHAMPEAUX differenziert zwischen Attributen und Beziehungen zu anderen Klassen durch die Untersuchung, ob eine bestimmte Eigenschaft *wesentlich* für die Definition eines Konzepts ist. Ist dies der Fall, handelt es sich um ein Attribut. Im anderen Fall handelt es sich um eine Beziehung zu einer anderen Klasse. Vgl. [de Champeaux 1993: S. 52]. Nach HENDERSON-SELLERS sind Attribute Referenzen auf interne Objekte, d. h. ihre Lebensdauer ist mit der des Objekts, dessen Attribut sie sind, identisch. [Henderson-Sellers 1998: S. 25 (Fig. 2.4)]. Streng genommen kann damit die Beziehung eines Attributs zu einem Konzept auch als interne referentielle Beziehung zu einer Klasse angesehen werden, wie dies im Metamodell der OML geschehen ist (vgl. [Firesmith 1998b: S. 3]). Bei dieser Sichtweise sind auch Attribute Klassen. Dies ist beispielsweise in MEMO-OML der Fall. Vgl. [Frank 1998b]. RUMBAUGH versteht ein Attribut als einen Bezeichner, der durch eine Klasse gekapselt ist, keine eigene Identität besitzt und nicht von anderen Klassen referenziert wird. Vgl. [Rumbaugh 1996], [Frank 1998c: 9]. In diesem Fall sind Attribute auf Werte beschränkt. In der UML werden seit der Version 1.4 Attribute als *Klassifizierer*, also als klassenähnliche Metatypen betrachtet. Vgl. [OMG 2001: S. 4-44]. Vorher wurden Attribute als benannte Eigenschaften bezeichnet und nicht festgelegt, ob es sich um Klassen handelt. Vgl. [Rumbaugh 1999: S. 166]. BOOCH unterschied überhaupt nicht zwischen Attributen und Klassen. Attribute wurden von ihm durch Aggregationsbeziehungen zwischen Klassen abgebildet. Vgl. [Booch 1995: S. 135]. PAPURT grenzt Attribute in der Form ab, dass Bezeichner, die von mehreren Klassen referenziert werden, als Klasse anzusehen sind. Vgl. [Papurt 1994], [Papurt 1994b], [Eckert 1995]. VETTER spricht dann von Attributen, wenn sie dazu dienen, die Objekte einer Klasse zu identifizieren, zu beschreiben oder Auskunft über ihren Status zu geben. Vgl. [Vetter 1995: S. 38]. Nach COOK und DANIELS ist keine exakte Unterscheidung zwischen Werten (Attributen) und Objekten (Assoziationen) möglich. Folgende Heuristiken können die Entscheidung unterstützen: Attribute sind unveränderliche Werte: Der Wert 3 kann niemals zum Wert 4 werden. Werte haben – im Gegensatz zu Objekten – neben ihrem Wert keine Identität: Zwei Werte 4 sind nicht voneinander unterscheidbar. Vgl. [Cook 1994: S. 30]. Vgl. zu dieser Problematik auch die Erörterung von SCHIENMANN [Schienmann 1997: S. 186 ff.], der die unterschiedlichen Positionen gegenüberstellt. SHLAER [Shlaer 1995] ist die einzige bekannte Quelle, die sich mit funktionalen Abhängigkeiten zwischen Attributen auseinandersetzt. ENGLMEIER befasst sich mit der Problematik, wie Attribute auf Klassen zu verteilen sind. Vgl. [Englmeier 1997]. ZÜLLIGHOVENS Darstellung zu diesem Thema ist wegen ihrer Anschaulichkeit besonders eingängig. Vgl. [Züllighoven 1998: S. 57].

[2] Siehe hierzu den Abschnitt 4.4.2.

zugegriffen werden.[1] Gehört ein Attribut zur Intension mehrerer Klassen, handelt es sich ebenfalls nicht um ein Attribut, sondern um eine eigenständige Klasse, welche durch referentielle Beziehungen (siehe Abschnitt 4.3.3) mit anderen Klassen verbunden ist. Eine Eigenschaft eines Konzepts soll dann als Attribut abgebildet werden, wenn diese Eigenschaft nicht selbst ein Gegenstand des Anwendungsbereichs ist und dessen Verhalten ggf. von Interesse ist.[2]

Operationen (Synonym: *Dienste*[3] oder *Methoden*) sind Abstraktionen des Verhaltens von Objekten,[4] welches durch den Erhalt einer Nachricht – als Handlungsaufforderung, Reiz oder Stimulus bezeichnet – ausgelöst wird. *Aktionen* stellen Abstraktionen des Verhaltens dar, welches die Objekte einer Klasse von sich aus oder im Rahmen von Operationen ausführen.[5] *Invarianten* sind auf Attributausprägungen und Beziehungen bezogene Abstraktionen von Integritätsregeln. Invarianten müssen immer erfüllt werden.[6] Dies wäre beispielsweise, dass die Summe der Innenwinkel eines Vierecks 360° betragen muss. *Vor-* und *Nachbedingungen* sind auf Operationen bezogene Abstraktionen von Integritätsregeln, die eingehalten werden müssen.[7] Der *Zustand* eines Objekts wird durch die aktuellen Werte seiner Attribute und der Beziehungen beschrieben, die das Objekt mit anderen Objekten besitzt.[8] Er stellt einen Vektor aller Attribute mit ihren aktuellen Werten und den Beziehungen dar, die ein Objekt zu einem bestimmten Zeitpunkt besitzt.[9] Hier

---

[1] Das folgt aus dem Geheimnisprinzip.

[2] Vgl. [Schienmann 1997: S. 187].

[3] [Henderson-Sellers 1998: Appendix E; Stichwort „Services"]

[4] Man beachte, dass dies von der üblichen Definition abweicht, Operationen seien prozedurale Abstraktionen, wie dies zum Beispiel bei LISKOV [Liskov 1986: S. 40] zu finden ist. Es wird nicht bestritten, dass Operationen durch Prozeduren realisiert werden *können*. Aber nach Ansicht des Verfassers stellt dies eine Sichtweise dar, die zu nah an der Programmierung orientiert ist. Im Rahmen der Analyse von Phänomenen der realen Welt ist sie auch wenig zweckdienlich. Wenn ein Objekt durch eine Verrichtung eines anderen Objekts seinen Zustand verändert, führt es nicht unbedingt eine Prozedur aus: Wenn ein Formular ausgefüllt wird, wird die Prozedur eher vom Ausfüllenden ausgeführt. Insofern war von dieser Sichtweise Abstand zu nehmen.

[5] Dies geht auf die von COLEMAN verwendete Unterscheidung zurück, dass sich ein Objekt durch die Dienste (gemeint sind prozedurale Abstraktionen) charakterisieren lässt, die es a) anderen Objekten zur Verfügung stellt und b) die es von anderen Objekten anfordert. Vgl. [Bear 1990], [Coleman 1992: S. 9]. In der UML wird eine Aktion leicht abweichend als elementarer Schritt bezeichnet, der im Rahmen einer Berechnung ausgeführt wird. Vgl. [Rumbaugh 1999: S. 122]. Auch diese Formulierung ist kompatibel.

[6] Vgl. [Burkhardt 1997: S. 125].

[7] In ähnlicher Weise bei FIRESMITH. Vgl. [Firesmith 1995: S. 27 f., 335 f.]. Siehe auch [Martin 1999: S. 203], [Kappel 1996], [Waldén 1995], [Cook 1994].

[8] Vgl. [de Champeaux 1993: S. 64], [Rumbaugh 1993: S. 107].

[9] Vgl. [McGregor 1993: S. 61]. Dies wird auch als *konkreter Zustand* bezeichnet.

wird im weiteren Verlauf mit Zustand eine Abstraktion bezeichnet, die einer bestimmten Kombination von Attributwerten und aktuellen Beziehungen einen für das Konzept eindeutigen Namen gibt.[1] *Ausnahmen* sind Fehlerabstraktionen.[2] Sie stellen hier eine Aktivität eines Objekts dar, durch die angezeigt wird, dass eine als außergewöhnlich betrachtete Situation eingetreten ist, auf die reagiert werden muss. Auf das Entstehen einer Ausnahme sollte ein System durch besondere Bearbeitungsschritte oder den Wechsel seines Verarbeitungsmodus reagieren.[3]

Für Attribute, Aktionen und Operationen ist die Abbildung als Intensionsknoten sinnvoll. Ihre Existenz ist immer an die Existenz der Klasse gebunden, da sowohl Attribute wie auch Operationen ohne zugehörige Klasse nicht existent sind. Die Abbildung der Beziehungen zu anderen Klassen geschieht dagegen nicht durch Intensionsknoten; sie werden als separate Metaklassen abgebildet. Auch Invarianten stellen Intensionsknoten dar, da sie Aussagen über die Klasse beinhalten, die diese zu erfüllen hat.

Auf den ersten Blick scheint dies auch für Vor- und Nachbedingungen zu gelten. Doch diese beschreiben nicht die Semantik der Klasse, sondern die einer Operation. Deshalb werden sie nicht als Intensionsknoten abgebildet, sondern als Prädikat der Operation. Die Abbildung des abstrakten Zustands als Intensionsknoten ist sinnvoll, um die Zustände eines Objekts nicht auf der Ebene der aktuellen Attributausprägungen und Beziehungen zu anderen Klassen betrachten zu müssen.

Die Typhierarchie für Intensionsknoten wird wie folgt aufgebaut: Die Intension wird im Konzeptuellen Schema durch Intensionsknoten dargestellt. Dies sind Knoten des Metatyps *INODE* und der davon spezialisierten Knotentypen. Attribute werden durch den Metatyp *ATTRIBUTE* abgebildet.

Der Begriff der *Aktivität* (*ACTIVITY*) wird als Abstraktion zum üblichen Begriff der Operation[4] eingeführt, da nun zwei Formen von Aktivitäten unterschieden werden. Unter *Aktivität* wird das gesamte Verhalten subsumiert, das ein Konzept aktiv oder als Reaktion auf einen Umwelteinfluss

---

[1] Dies wird in der Literatur als *expliziter* oder *abstrakter Zustand* bezeichnet und entspricht der üblichen Verwendung in der objektorientierten Modellierung. Vgl. [Booch 1999: S. 188], [Kappel 1996: S. 45], [Hitz 1999: S. 128]. FIRESMITH bezeichnet diesen abstrakten Zustand als *Zustandsattribut*. Vgl. [Firesmith 1995: S. 418]. BEHRINGER bezeichnet es als Makro-Zustand. Vgl. [Behringer 1997: S. 156].

[2] Vgl. [Firesmith 1995: S. 163]. In der UML werden Ausnahmen als besondere Form von Objekten modelliert, die erzeugt werden, wenn die Ausnahme eingetreten ist. Vgl. [Booch 1999: S. 285 ff.]. Man spricht davon, dass die Ausnahme vom Objekt „geworfen“ wurde. In der OML stellen Ausnahmen einen Metatyp der Eigenschaften von Klassen dar. Vgl. [Henderson-Sellers 1998: S. 22, 24]. Für eine Betrachtung des Ausnahme-Konstrukts im Bereich der Programmiersprachen siehe [Goodenough 1975] und [Oberweis 1991].

[3] Vgl. [Berard 1993: S. 147].

[4] Vgl. [Henderson-Sellers 1998: S. 22], [Booch 1999: S. 51].

ausführt.[1] Dies umfasst – im Gegensatz zu anderen Konzepten der objektorientierten Modellierung – nicht nur die Operationen, die als Reaktion auf den Erhalt einer Nachrichten ausgelöst werden, die das Konzept von einem anderen Konzept erhalten hat[2], sondern auch die Aktivitäten, die das Objekt selbständig ausführt und durch die ggf. eine Handlungsaufforderung für andere Konzepte entsteht.

Aktivitäten, durch die ein Stimulus für andere Konzepte entsteht, heißen *Aktionen* und werden durch den Metatyp *ACTION* abgebildet.[3] Aktivitäten, die als Ergebnis eines Stimulus stattfinden, heißen *Operationen* und werden durch den Metatyp *OPERATION* abgebildet. Die Metaklassen *ACTION* und *OPERATION* stellen Spezialisierungen der Metaklasse *ACTIVITY* dar. Invarianten werden durch den Metatyp *INVARIANT* repräsentiert. Vor- und Nachbedingungen von Operationen werden im Konzeptuellen Schema nicht als Metatyp, sondern als Prädikate `Pre()` und `Post()` des Knotentyps *OPERATION* abgebildet. Zustände werden zur expliziten Formulierung eines abstrakten Objektzustandes verwendet und durch den Metatyp *STATE* repräsentiert. Ausnahmen stellen eine Aktivität eines Objekts dar, durch die angezeigt wird, dass eine als außergewöhnlich betrachtete Situation eingetreten ist, auf die reagiert werden muss. Die Behandlung von Ausnahmen erfolgt durch Operationen.[4] Da ein Objekt durch eine Aktivität anzeigt, dass eine Ausnahme eingetreten ist, werden Ausnahmen als besondere Form der Aktion betrachtet und als Spezialisierung des Metatyps *ACTION* durch den Metatyp *EXCEPTION* abgebildet.

Dadurch ergibt sich zur Beschreibung der Intension von Konzepten im Metamodell des Konzeptuellen Schemas folgendes Teilmodell:

---

[1] Die Interpretation folgt der von FIRESMITH. Vgl. [Firesmith 1995: S. 302].

[2] [Reenskaug 1996: S. 38].

[3] Vgl. [OMG 1999: S. 2-103].

[4] Diese werden als Ausnahmebehandlungsroutinen oder *exception handler* bezeichnet. Vgl. [Louden 1994: S. 287], [Pratt 1997: S. 484 ff.].

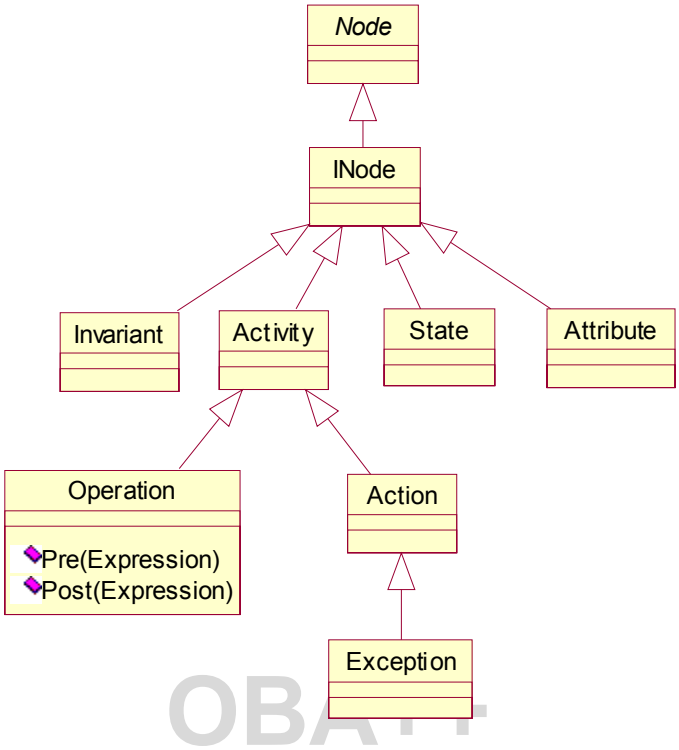


Abbildung 33: Teilmodell des Konzeptuellen Schemas für Intensionsknoten

4.3 Beziehungsknoten

4.3.1 Übersicht

In gleicher Weise wie für die Knoten wird für die Kanten des Semantischen Netzes eine Taxonomie entwickelt. Die unterschiedlichen Metaklassen der Beziehungsknoten werden von der Metaklasse *ASSOCIATIVELINK* spezialisiert. Die Beziehungen zwischen den Beziehungsknoten und den Knoten stellen Realisierungen der Beziehungen `Elem2Lnk`, `Lnk2Elem`, `LeavingLinks` und `ArrivingLinks` dar, die zwischen *MODELELEMENT*, *ASSOCIATIVELINK* und *NODE* definiert sind.[1]

[1] Nach WARMER und KLEPPE (VGL. [Warmer 1999: S. 73 ff.]) kann dieser Zusammenhang auch durch spezialisierte Assoziationen abgebildet werden. In diesem Fall werden die genannten Beziehungen in den spezialisierten Klassen redefiniert. Hiervon wurde abgesehen, da die Semantik dieses Konstrukts in der UML nicht präzise [Rumbaugh 1999: S. 163] bzw. überhaupt nicht [OMG 1999] definiert ist.

Auch die definierten Kantentypen sind epistemische Primitive, da sie unabhängig von einem bestimmten Problembereich sind. Auf oberster Ebene lassen sich drei unterschiedliche Kategorien unterscheiden, die als Beziehungsknotentypen abgebildet und von der Metaklasse *ASSOCIATIVELINK* spezialisiert werden:

- *Kooperationen*: Kooperationen (*COOPERATION*) subsumieren die dynamischen und statischen Beziehungen zwischen Konzepten. Die dynamischen Beziehungen bilden die Oberklasse *RESPONSIBILITYCOOPERATION*, die statischen Beziehungen bilden die Oberklasse *RELATIONSHIP*. Beide Metatypen stellen Spezialisierungen des Metatyps *COOPERATION* dar.
- *Intension*: Durch Intensionsbeziehungen (*INTENSION*) werden Intensionsknoten mit Klassenknoten verbunden.
- *Abhängigkeiten*: Abhängigkeiten zwischen Modellelementen werden mit Hilfe des Beziehungsknotentyps *DEPENDENCY* und der davon spezialisierten Klassen abgebildet.

Die folgende Abbildung beinhaltet die nächste Ebene der Vererbungshierarchie für Beziehungsknoten:

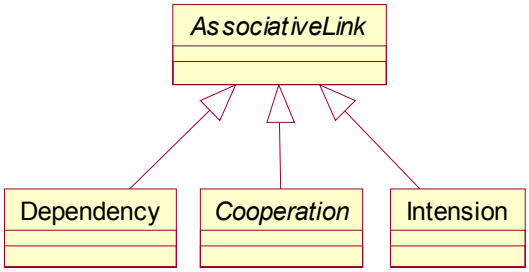


Abbildung 34: Differenzierung der Beziehungsknotentypen

Zunächst werden die Unterklassen der abstrakten Metaklasse *COOPERATION* dargestellt. *COOPERATION* stellt die Abstraktion für die statischen und dynamischen Beziehungstypen dar. Die statischen Beziehungstypen sind jene, die man üblicherweise in Klassendiagrammen verwendet.[1] Die dynamischen Beziehungstypen sind jene, die in Interaktions- und Zustandsdiagrammen verwendet werden. Dafür werden die Metaklassen *RELATIONSHIP* und *RESPONSIBILITYCOOPERATION* verwendet. Die Metaklasse *RELATIONSHIP* stellt die

[1] Man beachte, dass in dieser Menge die Abhängigkeitsbeziehungen nicht enthalten sind, da diese einen eigenständigen Metatyp darstellen.

Oberklasse für alle statischen Beziehungsknotentypen dar, die Konzeptknoten verbinden. Die Metaklasse *RESPONSIBILITYCOOPERATION* stellt die Oberklasse für alle dynamischen Beziehungsknotentypen dar, die Intensionsknoten miteinander verbinden:

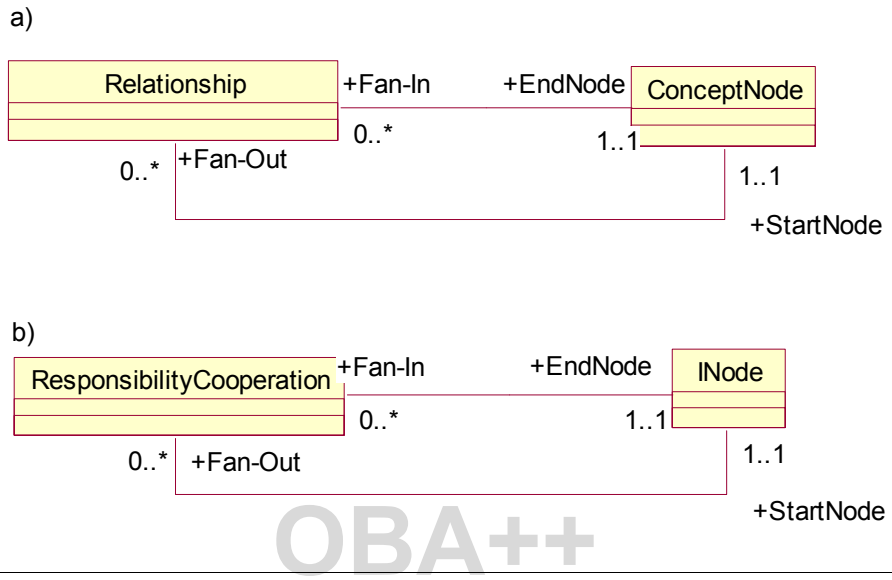


Abbildung 35: RESPONSIBILITYCOOPERATION *und* RELATIONSHIP

Aus der oberen Hälfte der Abbildung 36 (a) wird deutlich, dass Knoten der Metaklasse *RELATIONSHIP* und der davon spezialisierten Klassen immer zwei Knoten der Metaklasse *CONCEPTNODE* verbinden. Aus der unteren Hälfte der Abbildung 36 (b) wird deutlich, dass Knoten der Metaklasse *RESPONSIBILITYCOOPERATION* und der davon spezialisierten Klassen immer zwei Knoten der Metaklasse *INODE* verbinden. Eine Verbindung von Beziehungen durch Beziehungen des Typs *COOPERATION* wird nicht vorgenommen und wird deshalb auch nicht unterstützt.[1] Sie kann aber auf andere Art und Weise ermöglicht werden. In diesem Fall wären die Klassen *RESPONSIBILITYCOOPERATION* und *RELATIONSHIP* nicht mit *CONCEPTNODE* bzw. mit *INODE*, sondern mit *MODELELEMENT* als `EndNode` bzw. `StartNode` zu verbinden.[2] Für die Knoten der Metaklassen *INODE* und *CONCEPTNODE* gilt, dass sie durch mehrere

[1] Einige wenige Beispiele hierfür finden sich in der Literatur. FRANK, KAPPEL, HITZ und RUMBAUGH verwenden Vererbungsbeziehungen zwischen Beziehungen. Vgl. [Frank 1998c: S. 17 ff.], [Hitz 1999: S. 44 f.], [Rumbaugh 1999: S. 164]. Eine ähnliche Idee für Aggregationsbeziehungen findet sich in [Bock 1998b].

[2] Alternativ könnte auch schon die Klasse *COOPERATION* mit der Klasse *MODELELEMENT* verbunden werden.

Beziehungen des Typs *RESPONSIBILITYCOOPERATION* resp. *RELATIONSHIP* verbunden sein können.

Bei den Beziehungsknotentypen der Metaklasse *RELATIONSHIP* lassen sich die referentiellen und die taxonomischen Beziehungen unterscheiden. Die folgende Abbildung stellt den entsprechenden Ausschnitt aus dem Metamodell des Konzeptuellen Schemas dar. Die subsumierten Beziehungstypen werden in den folgenden Abschnitten ausführlicher dargestellt.

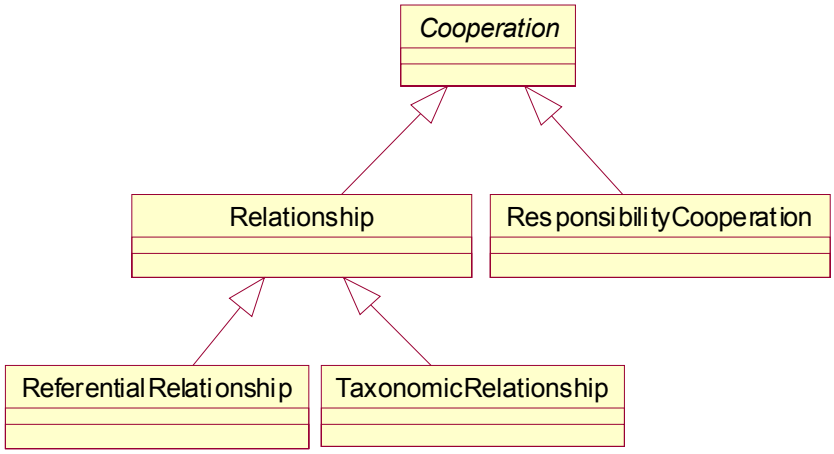


Abbildung 36: Beziehungsknoten des Typs COOPERATION im Konzeptuellen Schema

4.3.2 Taxonomische Beziehungen

Allgemein umfassen taxonomische[1] Beziehungen jene Beziehungen, bei denen ein Konzept in irgendeiner Form durch ein anderes Konzept klassifiziert wird oder als Basis für dessen Definition verwendet wird.[2] Umgangssprachlich lässt sich dies durch die Aussage „*A ist ein B*“ ausdrücken. Doch hinter dieser Aussage steht mitunter eine stark abweichende Semantik, wie BRACHMAN ausführt.[3] Hier werden fünf Formen unterschieden:

- **Spezialisierung**: Die Spezialisierungsbeziehung entspricht der klassifizierenden Beziehung aus dem Bereich der Wissensrepräsentation.

[1] Als *Taxonomie* bezeichnet der Duden den Zweig der Systematik, der sich mit dem praktischen Vorgehen bei der Klassifizierung der Lebewesen in systematischen Kategorien befasst. Vgl. [Duden 1993]. Vgl. auch die Darstellung der Historie der Taxonomie in [Meyer 1997: S. 864].

[2] Aus diesem Grund spricht man in der OML von *definierenden Beziehungen*. Vgl. [Firesmith 1997: S. 81].

[3] Vgl. [Brachman 1983].

- **Vererbung**: Die Vererbungsbeziehung ist eine besondere Form der Spezialisierungsbeziehung. Sie entspricht in ihrer Semantik einem Subtyp im Bereich der objektorientierten Programmierung, d. h., die spezialisierte Klasse erhält die Intension der vererbenden Klasse, kann diese aber nach festgelegten Regeln verändern.[1]
- **Instanzierung**: Die Instanzierungsbeziehung besteht zwischen einer Klasse und ihrer Extension.
- **Rollenbeziehung**: Die Rollenbeziehung besteht von einer Rolle zu einer Klasse.
- **Rollenübernahme**: Ein Objekt übernimmt in einer bestimmten Situation eine Rolle.[2]

Alle taxonomischen Beziehungen werden im Konzeptuellen Schema durch Knoten des Metatyps *TAXONOMICRELATIONSHIP* und seiner Unterklassen repräsentiert.

4.3.2.1 Spezialisierung

Die *Spezialisierung*[3] ist das Ergebnis einer Abstraktionsbildung durch Klassifikation, Kategorisierung bzw. durch Subsumption.[4] Sie entspricht dem Beziehungstyp der *Generalisierung* in der UML.[5] Begriffe werden in einer Struktur allgemeinerer und speziellerer Begriffe angeordnet, wobei ein Begriff die Spezialisierung eines anderen Begriffs – der Generalisierung – ist.[6] Auf diese Weise entsteht eine Hierarchie von Begriffen, an deren Wurzel allgemeine Begriffe zu finden sind und deren Blätter die spezielleren Begriffe darstellen. Als *Diskriminator* bezeichnet man die Eigenschaft, auf deren Basis die Spezialisierungshierarchie aufgebaut wird.[7] Die Spezialisierung kann wie folgt interpretiert werden:[8]

---

[1] Vgl. [Meyer 1997: S. 459 ff.].

[2] Vgl. [Firesmith 1997: S. 84 ff.], [Firesmith 1998d: 15], [Henderson-Sellers 1998: S. 29].

[3] Für die Semantik der Vererbung bzw. Spezialisierung siehe [Reimer 1991: S. 87 ff.], [Alhir 1998: S. 153].

[4] Vgl. [Henderson-Sellers 1998: Appendix E; Stichworte „Generalization, inheritance and polymorphism“, „Abstraction“, „Classification and partitions“]. Die Spezialisierungsbeziehung wird von SCHIENMANN als *Inklusion* bezeichnet. Vgl. [Schienmann 1997: S. 197].

[5] Vgl. [Rumbaugh 1999: S. 287].

[6] Vgl. [Firesmith 1995: S. 413].

[7] Vgl. [Firesmith 1995: S. 150], [Henderson-Sellers 1998: Appendix E; Stichwort „Discriminant“].

[8] Die Vererbung unter Unsicherheit wird hier nicht dargestellt, weil sie – abgesehen von GRAHAM – in der gesamten Literatur keine Verwendung findet. Vgl. [Graham 1991: S. 333]. Aus dem gleichen Grund bleibt auch die dynamische Klassifikation von MARTIN und ODELL (vgl. [Martin 1999]) unbeachtet.

- **Extensionale Sicht**[1] bzw. **Teilmengenbeziehung**: Für zwei Mengen A und B, von denen B eine Teilmenge von A ist, gilt: Alle x, die Element aus der Menge B sind, sind auch Element aus der Menge A.
- **Intensionale Sicht**[2] bzw. **Erweiterungsbeziehung**: Das speziellere Konzept übernimmt die Intension des übergeordneten Konzepts und fügt weitere Merkmale zur Erweiterung der eigenen Intension hinzu.
- **Konditionale Sicht**:[3] Die Vererbung postuliert das **Zutreffen von Eigenschaften** zwischen Konzepten, zwischen denen eine Spezialisierungsbeziehung besteht. Dies stellt eine Beziehung zwischen Prädikaten dar. Wenn ein prädikatenlogischer Ausdruck für Konzept K_2 zutrifft und dieses Konzept eine Spezialisierung des Konzepts K_1 darstellt, so trifft dieser Ausdruck auch auf K_1 zu.
- „**a kind of**" (auch als *AKO* bezeichnet) **Sicht**:[4] Dies ist die im Bereich Semantischer Netze am häufigsten verwendete Interpretation der Spezialisierung. Die AKO-Sicht beinhaltet keine Aussagen über Intension, Extension oder prädikatenlogische Ausdrücke, auch wenn diese zutreffen können. Sie besagt nur: Ein Konzept **ist eine Art** eines anderen Konzepts. Beispiel: Stühle sind eine Art Sitzgelegenheiten.
- **Substitutions-Sicht:** Die Exemplare einer spezielleren Klasse verhalten sich nicht anders als die der allgemeineren Klasse.[5]

Die Spezialisierungsbeziehung wird im Konzeptuellen Schema für alle genannten Formen der Kategorienbildung verwendet. Sie entspricht dem Metatyp *SPECIALISATION*. Die Spezialisierungsbeziehung geht vom spezielleren Knoten[6] aus und endet beim allgemeineren Knoten.[7] Die durch die Spezialisierungsbeziehung verbundenen Knoten müssen den gleichen Metatyp (d. h. Klasse, Rolle oder Objekt) besitzen. Die Spezialisierungsbeziehung ist asymmetrisch, transitiv und nicht reflexiv. Für zwei durch eine Spezialisierungsbeziehung verbundene Konzeptknoten X und Y gilt:

- Wenn X eine Spezialisierung von Y ist, kann Y keine Spezialisierung von X sein.

---

[1] Vgl. [Kappel 1996: S. 18, 36], [Helbig 1996: S. 105], [Odell 1997], [Vetter 1995: S. 69 ff.].

[2] Vgl. [Kappel 1996: S. 17], [Burkhardt 1997: S. 157], [Odell 1997].

[3] Vgl. [Brachman 1983: S. 32].

[4] Vgl. [Brachman 1983: S. 32].

[5] Vgl. [Liskov 1993], [Liskov 1994], [Züllighoven 1998: S. 50]. Das bedeutet, dass sie zusätzliches Verhalten besitzen dürfen, aber mindestens das geerbte erfüllen müssen.

[6] Auch als Spezialisierung, Erweiterung, Unter- oder abgeleitete Klasse bezeichnet.

[7] Auch als Generalisierung, Basis-, Ober- oder Superklasse bezeichnet.

- Wenn Y eine Spezialisierung von X ist und Z eine Spezialisierung von Y, dann ist Z auch eine Spezialisierung von X.
- X ist keine Spezialisierung von sich selbst.

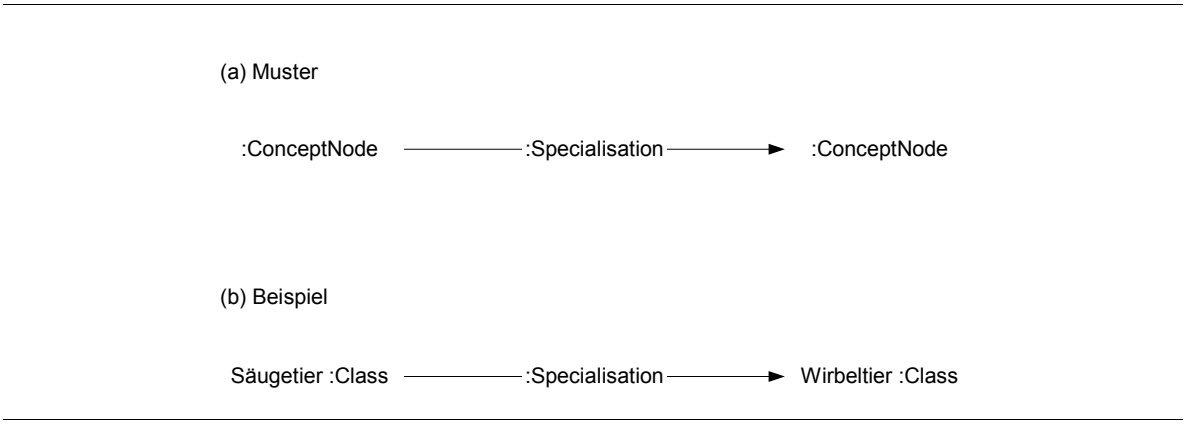


Abbildung 37: Muster und Beispiel einer Spezialisierungsbeziehung

Für den Beziehungsknotentyp *SPECIALISATION* gilt: Ein Knoten des Metatyps *CONCEPTNODE* kann im Konzeptuellen Schema über einen Beziehungsknoten des Metatyps *SPECIALISATION* mit mehreren Knoten des Metatyps *CONCEPTNODE* verbunden sein. Das bedeutet auch, dass Knoten der Unterklassen von *CONCEPTNODE* mit mehreren anderen Knoten der Unterklassen von *CONCEPTNODE* verbunden sein können. Dies entspricht folgendem UML-Klassendiagramm:

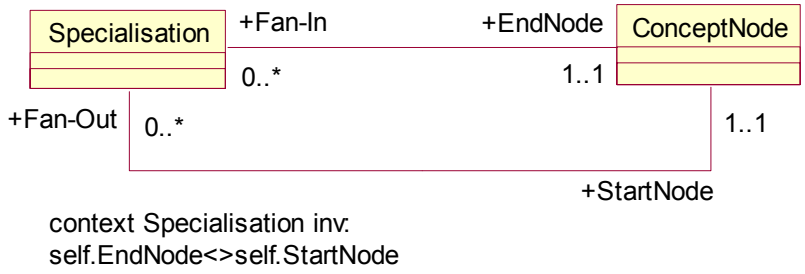


Abbildung 38: Spezialisierung im Konzeptuellen Schema

4.3.2.2 Vererbung

Die Vererbung wird als restriktivere Form der Spezialisierungsbeziehung verwendet, für die die in der objektorientierten Programmierung übliche Semantik gilt.[1] Vererbung zwischen zwei Klassen bedeutet, dass eine Klasse ihre Intension an Erben weitergibt und diese nur nach bestimmten Regeln neue Elemente zur Intension hinzufügen bzw. ererbte verändern dürfen.[2] Auf diese Weise werden Typhierarchien aufgebaut. Im Konzeptuellen Schema wird diese Beziehung durch den

[1] Die Vererbung ist eine besondere Form der Spezialisierung. Sie beinhaltet eine präzisere Semantik. Jede Vererbung stellt auch eine Spezialisierung dar. Umgekehrt ist dies nicht der Fall. Eine sinnvolle Spezialisierung stellt nicht unbedingt eine sinnvolle Vererbung dar; teilweise stehen beide sogar im Widerspruch. Hierzu ein Beispiel: Die Spezialisierungshierarchie Quadrat→ Rechteck→ Viereck, bei der das Viereck dass allgemeinste Konzept darstellt, wird intuitiv bzw. durch Erfahrungswissen als vernünftig angesehen. Eine übliche Sprechweise wäre: „Ein Quadrat ist eine besondere Form des Rechecks." Eine Vererbungshierarchie könnte aber auch umgekehrt aufgebaut werden, nämlich Viereck→ Rechteck→ Quadrat, weil die erbenden Klassen jeweils zusätzlich Attribute und Operationen hinzufügen bzw. existierende redefinieren. Die hierzu verwendete Sprechweise wäre: „Rechteck und Viereck fügen zusätzliche Attribute und Operationen hinzu."

[2] Weiter wird auf die Semantik der Vererbung nicht eingegangen, da zu diesem Thema eine Vielzahl von Veröffentlichungen existieren. Vgl. [Frank 1998b], [Taivalsaari 1996], [Hitz 1999: S. 45 ff.], [Alhir 1998: S. 59 f.], [Meyer 1997: S. 459 ff., S. 808 ff.], [Liskov 1994], [Wegner 1988], [Henderson-Sellers 1999], [Cook 1994: S. 4467 ff.], [Papurt 1996], [Züllighoven 1998: S. 35]. In [Lecoeuche 1995], [Firesmith 1996], [Coleman 1992], [McGregor 1993], [Atkinson 2000: S. 33 ff.] wird der Zusammenhang zwischen Vererbung und den zugehörigen Zustandsmaschinen der Klassen untersucht.

Beziehungsknotentyp *INHERITANCE* repräsentiert. Für eine wohldefinierte Vererbungsbeziehung sind eine Reihe von Kriterien zu erfüllen:[1]

- Die vererbende Klasse besitzt keinen Intensionsknoten, der nicht auch für die erbende Klasse sinnvoll ist.
- Die erbende Klasse darf überall verwendet werden, wo die vererbende Klasse verwendet werden darf.[2]
- Die Exemplare der erbenden Klasse können sich wie die Exemplare der vererbenden Klasse verhalten und zusätzliches Verhalten in Form von *ACTIVITY*-Knoten hinzufügen.
- Die erbende Klasse kann zusätzliche *ATTRIBUTE*-Knoten oder eine *REFERENTIAL-RELATIONSHIP* zu den vererbten hinzufügen oder redefinieren.
- Die erbende Klasse kann zusätzliche Einschränkungen in Form von Invarianten oder restriktiveren Vor- oder Nachbedingungen verwenden.[3]

Dass das Verhalten einer erbenden Klasse zum Verhalten der vererbenden Klasse „passen“ muss, wird auch als *Substitutionsprinzip* von LISKOV bezeichnet.[4] Danach soll in erbenden Klassen die Semantik der vererbenden Klasse nicht in einer Art und Weise verändert werden, dass die erzeugten Exemplare ein anderes Verhalten aufzeigen als die Exemplare der vererbenden Klasse.[5]

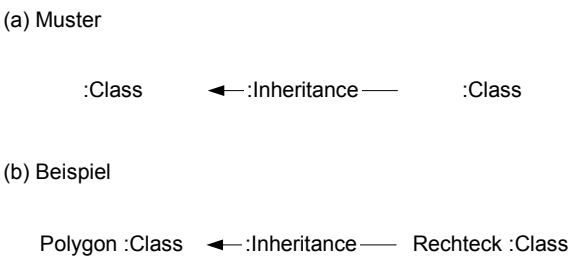


Abbildung 39: Muster und Beispiel einer Vererbungsbeziehung

[1] Vgl. [Hitz 1999: S. 45 ff.], [Louden 1994], [Meyer 1997: S. 820], [Alhir 1998], [Page-Jones 2000: S. 278 – 309], [Züllighoven 1998: S. 36, 49 ff.].

[2] Die wird auch als *Typ-Konformität* bezeichnet. Vgl. [Constantine 1999: S. 281].

[3] Vgl. [Meyer 1997: S. 573]. Dies wird als *Kovarianz* und *Kontravarianz* bezeichnet. Vgl. [Constantine 1999: S. 282], [Pomberger 1993: S: 226].

[4] Auch: *Liskov'sches Substitutionsgesetz* (LSP).

[5] Vgl. [Hitz 1999: S. 46 f.], [Liskov 1993], [Liskov 1994], [America 1991].

Klassen können durch mehrere Vererbungsbeziehungen verbunden sein. Für den Beziehungsknotentyp *INHERITANCE* gilt: Ein Knoten des Metatyps *CLASS* kann im Konzeptuellen Schema über einen Beziehungsknoten des Metatyps *INHERITANCE* mit mehreren Knoten des Metatyps *CLASS* verbunden sein. Es gibt Klassen, die weder von anderen Klasse erben noch an andere Klassen vererben. Die Vererbungsbeziehung verbindet nur Klassenknoten mit Klassenknoten:[1]

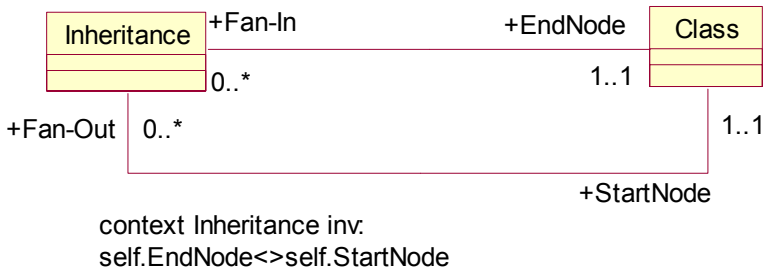


Abbildung 40: Beziehungsknotentypen des Typs INHERITANCE im Konzeptuellen Schema

4.3.2.3 Instanzierung

Ein Exemplar bzw. eine *Instanz* ist eine konkrete Manifestation einer Abstraktion. Die *Instanzierung* ist die Bildung eines Exemplars (sprich: ein Objekt als Bestandteil der Extension) auf der Basis einer Klasse.[2] Im OA wird die Instanzierungsbeziehung zwischen Klassen und Objekten verwendet, um auszudrücken, dass ein Objekt zur Extension einer Klasse gehört.[3] Die Instanzierung ist eine Beziehung zwischen Individualkonzepten (Metatyp *OBJECT*) und Klassen

[1] Es sei darauf hingewiesen, dass der objektorientierte Vererbungsbegriff nicht der umgangssprachlichen Verwendung entspricht. In der realen Welt erbt eine Person von einer anderen Person. Dies würde eine Vererbungsbeziehung auf Objektebene entsprechen. Doch diese Form der Vererbung ist nicht gemeint. Die objektorientierte Vererbungsbeziehung bezieht sich immer auf Klassen.

[2] Vgl. [Booch 1999: S. 185]. Die Beziehungen zwischen Klassen und Metaklassen stellen eine besondere Art der Instanzierungsbeziehung dar, die von SCHIENMANN als *Hypostasierung* bezeichnet wird. Vgl. [Schienmann 1997: S. 208]. Werden zu dem vorliegenden Metamodell Metaklassen als Konzepttyp hinzugefügt, so wären die taxonomischen Beziehungen um diesen Beziehungsknotentyp zu erweitern. Zur Metamodellierung und Metatypen vgl. [Martin 1999: S. 305 – 334].

[3] Teilweise wird bei Klasse auch von allgemeinen Konzepten, bei Objekten von individuellen Konzepten gesprochen. Vgl. [Brachman 1997: S. 34 f.].

(Metatyp *CLASS*, Basiskonzept). Dabei ist ein Objekt immer nur die Extension exakt einer Klasse.[1] Die Begriffe *Instanz*, *Exemplar* und *Objekt* werden als Synonyme verwendet.[2] Die Instanzierungsbeziehung wird für folgende Beobachtungen der Realität verwendet:

- Die **Gemeinsamkeiten einer Gruppe von Objekten** werden kategorisiert und in einer Klasse zusammengefasst.[3]
- Für das Basiskonzept **zutreffende Prädikate** treffen auch auf das Individualkonzept zu.[4]
- Die Instanzierung wird auch verwendet, um die Beziehung zwischen einer **Menge** und einem **Element dieser Menge** zu repräsentieren.[5]
- Das Basiskonzept ist eine statische Beschreibung, die als **Muster zur Erzeugung** des Individualkonzepts verwendet wird.[6]

Im Konzeptuellen Schema wird diese Beziehung durch den Beziehungsknotentyp *INSTANCE* repräsentiert.[7] Die Instanzierungsbeziehung ist nicht transitiv,[8] asymmetrisch und nicht reflexiv. Die Extension einer Klasse besitzt keine Extension.[9] Wenn X ein Exemplar von Y ist, kann Y kein Exemplar von X sein. Schließlich kann X kein Exemplar von sich selbst sein. Ausgangspunkt der Instanzierungsbeziehung ist ein Objekt, Endpunkt ist eine Klasse.

Für den Beziehungsknotentyp *INSTANCE* gilt: Ein Knoten des Metatyps *CLASS* muss im Konzeptuellen Schema über einen Beziehungsknoten des Metatyps *INSTANCE* mit keinem Knoten des Metatyps *OBJECT* verbunden sein. Jeder Knoten des Metatyps *OBJECT* muss über einen Beziehungsknoten des Metatyps *INSTANCE* mit genau einem Knoten des Metatyps *CLASS*

---

[1] Vgl. [Burkhardt 1997: S. 166], [Burkhardt 1994: S. 55]. Anders in der UML, wo Objekte Exemplare mehrerer Klassen sein können.

[2] Vgl. [Booch 1999: S. 185].

[3] Vgl. [Henderson-Sellers 1998: Appendix E; Stichwort „Classification and partitions“]

[4] Vgl. [Brachman 1983: S. 32].

[5] Vgl. [Brachman 1983: S. 32].

[6] Dies ist die im Bereich der objektorientierten Programmierung verwendete Interpretation.

[7] Diese Beziehung entspricht der InstanceOf-Beziehung in der UML. Vgl. [Booch 1999: S. 181], [Burkhardt 1997: S. 81]. Die Instanzierungsbeziehung ist in der UML keine taxonomische Beziehung, sondern wird als Dependency-Beziehung eingeordnet. Vgl. [Rumbaugh 1999: S. 251].

[8] Vgl. [Atkinson 2000: S. 34].

[9] Mit der Ausnahme von Metaklassen, deren Instanzen Klassen sind. Deren Extension sind dann die Objekte. Hier liegt ein Fall transitiver Instanzierungsbeziehungen vor. Auf diesen Fall wird hier nicht weiter eingegangen.

verbunden sein, da er die Extension genau einer Klasse ist; sein Fan-Out ist gleich 1.[1] Dies entspricht folgendem UML-Klassendiagramm:

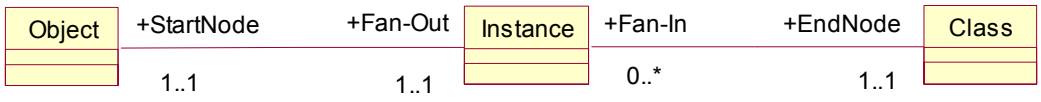


Abbildung 41: Beziehungsknotentypen des Typs INSTANCE im Konzeptuellen Schema

Das folgende Beispiel stellt die Aussage dar, dass Herr oder Frau Müller ein Exemplar der Klasse Angestellter ist. Die Instanzierungsbeziehung beginnt beim Objektknoten und endet beim Klassenknoten.

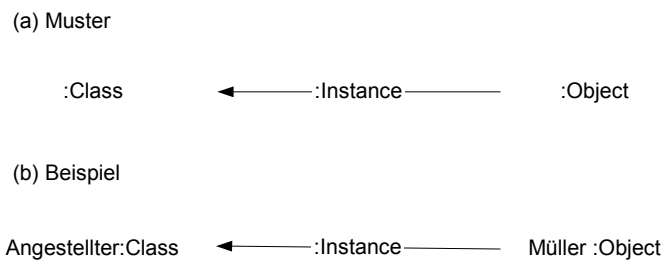


Abbildung 42: Muster und Beispiel einer Instanzierungsbeziehung

4.3.2.4 Rollenbeziehung

Die *Rollenbeziehung* ist die Beziehung einer Rolle zu einer Klasse. Sie ist eine besondere Form der taxonomischen Beziehung.[2] GOTTLOB et. al. schreiben, dass eine Rolle die Intension der Klasse beinhaltet und ihre eigene Intension hinzufügt. Eine Rolle ‚Angestellter' beinhaltet die Intension der Klasse ‚Person', zeigt aber in dieser Rolle ein besonderes Verhalten. Umgekehrt erhält eine Klasse die Intension ihrer Rollen. Fähigkeiten, die ein Objekt in einer Rolle besitzt, sind

[1] Das bedeutet, dass es keine Objekte gibt, die zu keiner Klasse gehören.

[2] KRISTENSEN verwendet den Begriff der *Roleification*. Vgl. [Kristensen 1996: S. 145]. FRANK verwendet den Rollenbegriff zur Realisierung der *Delegation*, d. h. des transparenten Weiterreichens *(dispatching)* von Nachrichten zwischen Objekten. Vgl. [Frank 1996], [Frank 1997c].

Fähigkeiten seiner Klasse. Klassen erweitern durch Rollen ihre Intension und partitionieren ihre Extension.[1]

Die Rollenbeziehung verbindet Rollenknoten als Ausgangspunkt mit Klassenknoten als Endpunkt. Im Konzeptuellen Schema wird diese Beziehung durch einen Beziehungsknoten vom Metatyp *ROLEOF* dargestellt. Der Klassenknoten wird dabei als Rollenfüller (*Rolefiller*) bezeichnet. In der folgenden Abbildung wird der Sachverhalt dargestellt, dass eine Person in der Rolle eines Vorgesetzten die Aktivität *erteilt Anweisung* ausführen kann. Dieses Verhalten zeigen die Objekte einer Klasse, wenn sie die entsprechende Rolle einnehmen.

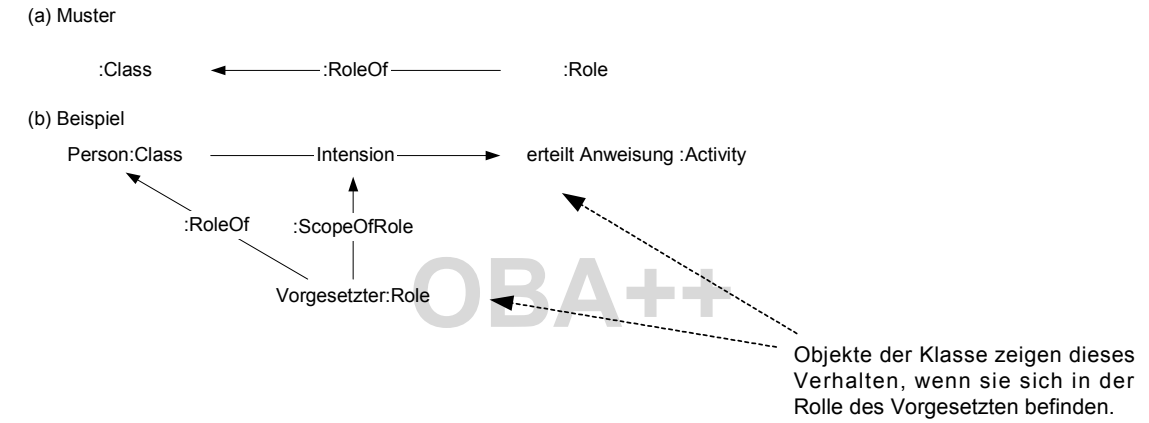


Abbildung 43: Muster und Beispiel einer Rollenbeziehung[2]

Rollen werden hier weder als Objekte noch als Klassen angesehen. In der OML werden Rollen verwendet, wenn die Instanzen einer Klasse in bestimmten Situationen zusätzliche Verantwortlichkeiten übernehmen. Sie werden als unvollständiges Objekt („partial object“) bezeichnet.[3] Auch Objekte verschiedener Klassen können die gleiche Rolle übernehmen.[4] FRANK verwendet Rollen als einen speziellen Klassentyp, deren Objekte eingehende Nachrichten transparent weiterreichen.[5] Das Rollenobjekt beinhaltet die Schnittstelle des Rollenfüllerobjekts und hat den transparenten

[1] Zu den Eigenschaften von Rollen siehe den Aufsatz von GOTTLOB, SCHREFL und RÖCK [Gottlob 1996]. In der Dissertation von RIEHLE wird ausführlich auf die Vorteile der Verwendung von Rollen im objektorientierten Entwurf von Softwaresystemen eingegangen. Vgl. [Riehle 2000].

[2] Die verwendete Beziehung des Typs *SCOPEOFROLE* wird in Abschnitt 4.3.6.2 eingeführt.

[3] Vgl. [Firesmith 1998c: S. 63].

[4] Vgl. [Firesmith 1998d: S. 8].

[5] Vgl. [Frank 1997c].

Zugriff auf den Zustand des Rollenfüllerobjekts.[1] Dies ist auf der einen Seite eine Art von Vererbung, auf der anderen Seite bricht es durch den transparenten Zugriff des Rollenobjekts auf das Rollenfüllerobjekt das Geheimnisprinzip. Da das Rollenobjekt durch seinen Objektcharakter einen eigenen Zustand besitzt, setzt sich der Zustand von Rollenfüller- und Rollenobjekt jeweils aus ihrem eigenen Zustand und aus dem des zugehörigen Rollen- oder Rollenfüllerobjekts zusammen. Ein weiteres Problem besteht bei der Bestimmung der Intension. Wenn die Rollenobjekte selbst eine Intension besitzen würden, müsste zu der Intension der Rollenfüller-objekte die Intension aller zugehörigen Rollenobjekte hinzugefügt werden. Wegen dieser Problematik erhalten hier Rollen (im Konzeptuellen Schema[2]) weder Intensionsknoten noch Beziehungen[3] zu anderen Konzeptknoten. Der Beziehungsknotentyp *SCOPE* wird unter anderem dafür verwendet, um die *Pseudo-Intension* einer Rolle durch Abhängigkeitsbeziehungen abzubilden.[4]

Für den Beziehungsknotentyp *ROLEOF* gilt: Ein Knoten des Metatyps *CLASS* muss im Konzeptuellen Schema mit keinem Knoten des Metatyps *ROLE* über einen Beziehungsknoten des Metatyps *ROLEOF* verbunden sein. Jeder Knoten des Metatyps *ROLE* muss mit genau einem Knoten des Metatyps *CLASS* über einen Beziehungsknoten des Metatyps *ROLEOF* verbunden sein; sein Fan-Out ist gleich 1. Das bedeutet, dass eine Rolle immer im Bezug auf eine Klasse definiert ist und dass eine Klasse in mehreren Rollen auftreten kann. Rollen sind also immer im Bezug auf Klassen definiert.[5] Dies entspricht folgendem UML-Klassendiagramm:

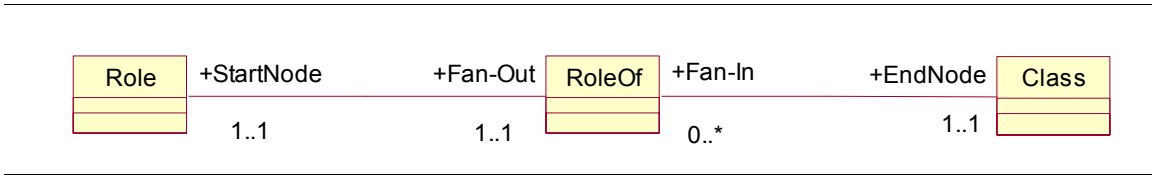


Abbildung 44: Beziehungsknotentyp ROLEOF im Konzeptuellen Schema

[1] Vgl. [Frank 1997c: S. 12].

[2] Im Rahmen der objektorientierten Programmierung kann eine Realisierung von Rollen durch Klassen und Objekte durchaus sinnvoll sein.

[3] Abgesehen von der RoleOf-Beziehung.

[4] Vgl. Abschnitt 4.3.6.2.

[5] Hier weicht das Konzeptuelle Schema von der UML ab, in der auch Rollen ohne Klassenbezug verwendet werden dürfen. Vgl. [OMG 2001: S. 3-131]. Allerdings ist im Metamodell der UML für statische Modelle kein Metatyp Rolle definiert [OMG 2001: S. 2-17, Figure 2-8]. Im Bereich der dynamischen Modellierung ist eine *ClassifierRole* [OMG 2001: S. 2-126] definiert, deren Extension die Elemente einer Klasse enthält, die in einer Kollaboration verwendet werden. Daraus kann gefolgert werden, dass auch die *ClassifierRole* immer nur in Verbindung mit einer Klasse auftritt.

4.3.2.5 Rollenübernahme

Weder aus der Rollen- noch aus der Instanzierungsbeziehung geht hervor, dass ein Objekt in einer bestimmten Situation eine Rolle übernimmt. Durch die Rollenbeziehung *ROLEOF* wird auf der Klassenebene ausgedrückt, dass die Exemplare der Klasse eine Rolle übernehmen. Muss dagegen zusätzlich ausgedrückt werden, dass ein bestimmtes Objekt zu einem bestimmten Zeitpunkt eine Rolle innehat, ist eine Kante zwischen dem Rollen- und dem Objektknoten notwendig. In der OML wird dafür die Beziehung *plays the role of* verwendet, die hier als *PLAYSROLE*-Beziehungstyp definiert wird.[1] Der Beziehungsknotentyp *PLAYSROLE* verbindet Knoten vom Metatyp *OBJECT* mit Knoten vom Metatyp *ROLE*.

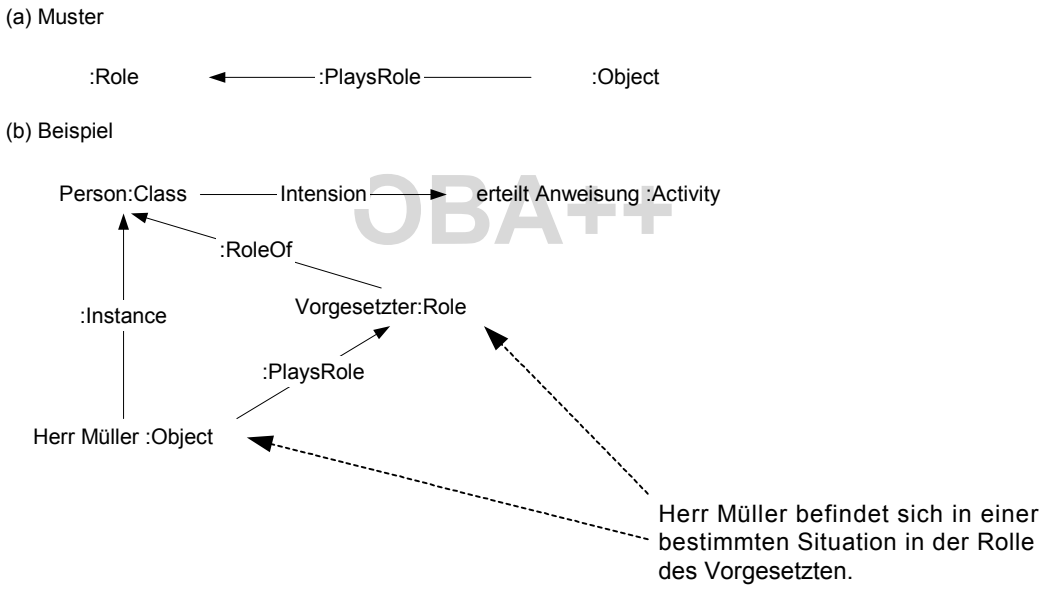


Abbildung 45: Ein Objekt übernimmt eine Rolle

Für den Beziehungsknotentyp *PLAYSROLE* gilt: Ein Knoten des Metatyps *OBJECT* kann im Konzeptuellen Schema mit mehreren Knoten des Metatyps *ROLE* verbunden sein. Jeder Knoten des Metatyps *ROLE* kann mit mehreren Knoten des Metatyps *OBJECT* verbunden sein. Das bedeutet, dass Objekte mehrere Rollen übernehmen können und dass Rollen von mehreren Objekten übernommen werden können. Dies entspricht folgendem UML-Klassendiagramm:

[1] Vgl. [Firesmith 1997: S. 84 ff.], [Firesmith 1998d: 15], [Henderson-Sellers 1998: S. 29].

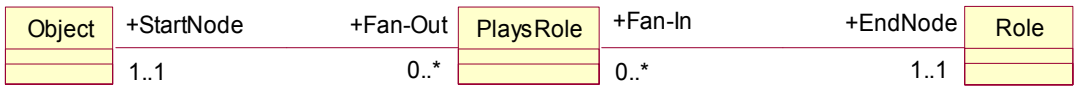


Abbildung 46: Beziehungsknotentyp des Typs PLAYSROLE im Konzeptuellen Schema

4.3.2.6 Taxonomische Beziehungen im Konzeptuellen Schema

Das folgende Teilmodell des Konzeptuellen Schemas gibt die taxonomischen Beziehungen wieder.

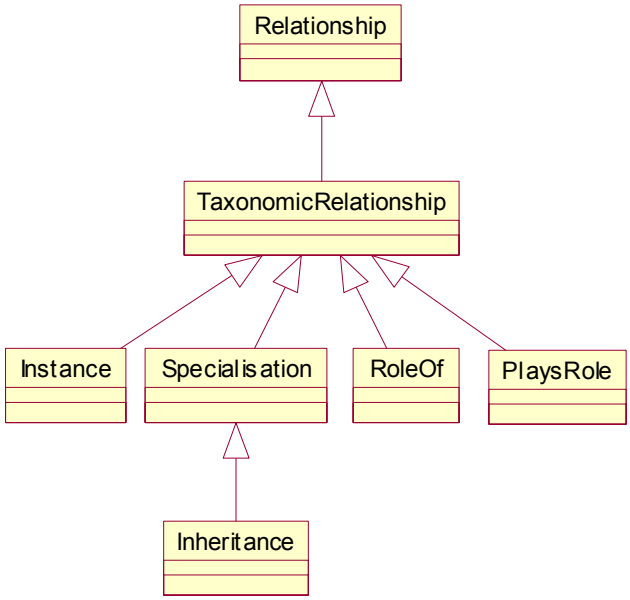


Abbildung 47: Taxonomische Beziehungen im Konzeptuellen Schema

Die definierten Beziehungstypen lassen sich über Vererbung noch weiter spezialisieren. Dies trifft insbesondere auf die Vererbungsbeziehung (*INHERITANCE*) zu, die in der OML weiter in Schnittstellenvererbung und Implementierungsvererbung differenziert wird.[1] Diese Beziehungstypen beziehen sich auf Aspekte der Vererbung, die im Rahmen der objektorientierten Programmierung notwendig sind und hier nicht weiter von Belang sind.

[1] Vgl. [Firesmith 1997: S. 91 ff.].

Eine mögliche Erweiterung dieser Vererbungsstruktur wäre die Einführung eines Beziehungsknotentyps *POWERTYPE*, der in Anlehnung an die UML[1] die Spezialisierungsbeziehung zwischen Metaklassen und Klassen abbildet.[2] Dieser würde vom Beziehungsknotentyp *SPECIALISATION* erben, da er eine besondere Form der Spezialisierung, nicht jedoch eine Vererbungsbeziehung darstellt. Im Unterschied zur UML wird hier durch die Beziehung zu anderen Klassen festgehalten, ob es sich um eine Klasse, um eine Metaklasse oder um eine Meta-Metaklasse handelt. In der UML erhält die Klasse das Stereotyp «powertype», wodurch nur eine weitere Ebene, die einer Metaklasse, abgebildet werden kann. Verwendet man hierfür eine Beziehung, ist es möglich, im Konzeptuellen Schema beliebige Strukturen von Klassen, Metaklassen, Meta-Metaklassen usw. abzubilden.[3] Diese Möglichkeit wird an dieser Stelle aus Gründen der Vollständigkeit erwähnt, aber im Metamodell des Konzeptuellen Schemas nicht eingeführt, weil mit dem entwickelten Ansatz keine Metamodellierung durchgeführt wird.

4.3.3 Referentielle Beziehungen

Referentielle Beziehungen sind alle Beziehungen, bei denen ein Konzeptknoten eine Referenz (Verweis) auf einen anderen Konzeptknoten besitzt.[4] Referentielle Beziehungen werden im Konzeptuellen Schema durch Beziehungsknoten des Metatyps *REFERENTIALRELATIONSHIP* und den davon spezialisierten Metatypen repräsentiert. Bei den referentiellen Beziehungen wird zwischen der Assoziation, der Ganzes-Teil-Beziehung mit ihren Untertypen und der Containerbeziehung unterschieden.[5] Auch hier verbinden alle referentiellen Beziehungen im Metamodell des Konzeptuellen Schemas Knoten des Metatyps *CLASS*.

Für den Beziehungsknotentyp *REFERENTIALRELATIONSHIP* gilt: Ein Knoten des Metatyps *CLASS* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *REFERENTIALRELATIONSHIP* mit mehreren Knoten des Metatyps *CLASS* verbunden sein. Das bedeutet, dass Klassen mit mehreren anderen Klassen verbunden sein können, aber nicht müssen. Dies entspricht folgendem UML-Klassendiagramm, welches auch für alle erbenden Metatypen gilt:

---

[1] Vgl. [Rumbaugh 1999: S. 392], [OMG 1999: S. 2-27, 3-50].

[2] Vgl. Abschnitt 2.2.3. und [Atkinson 2000: S. 32].

[3] MARTIN und ODELL gehen auf diese Möglichkeit explizit ein. Vgl. [Martin 1999: S. 309].

[4] Vgl. [Firesmith 1997: S. 105].

[5] Siehe auch [Henderson-Sellers 1998: S. 26].

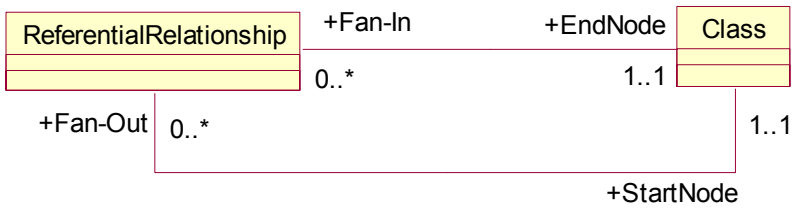


Abbildung 48: Referentielle Beziehungen im Konzeptuellen Schema

4.3.3.1 Assoziation

„Eine Assoziation beschreibt eine Menge von Verknüpfungen mit einer gemeinsamen Struktur und Semantik“, schreibt RUMBAUGH.[1] Assoziationen abstrahieren von den Verknüpfungen, die zwischen einzelnen Exemplaren existieren. Durch sie werden Exemplare einer Klasse mit Exemplaren einer anderen Klasse verknüpft bzw. diesen zugewiesen. Sie stellt eine Abbildung von den Elementen einer Urbildmenge auf die Elemente einer Bildmenge dar, die die Form einer Relation besitzt.[2] Eine einzelne Verknüpfung ist ein geordnetes Tupel von zwei Exemplaren[3] und verknüpft ein Exemplar aus der Urbildmenge mit einem Exemplar der Bildmenge. Die Assoziation ist die Menge der Verknüpfungen und damit ihr Typ.[4] Sie ist eine Teilmenge des kartesischen Produkts der beiden beteiligten Mengen. Die Tatsache, dass durch die Assoziation ein Element der Urbildmenge mit mehreren Elementen der Bildmenge verknüpft sein kann, wird durch Kardinalitäten dargestellt. Ähnlich wie in der Semantischen Datenmodellierung bzw. in der Entity-Relationship-Modellierung[5] kann beispielsweise festgelegt werden, dass ein Abteilungsleiter keinen, einen oder mehrere Mitarbeiter hat. Alle Exemplare eines Assoziationstyps verknüpfen Exemplare der gleichen Klassen, d. h. alle Exemplare der Urbildmenge und alle Exemplare der

[1] [Rumbaugh 1993: S. 34]. Der Duden bezeichnet mit Assoziation eine ursächliche Verknüpfung von Vorstellungen. [Duden 1993]. Für eine umfangreiche Darstellung der Interpretationsmöglichkeiten der Assoziation als Beziehung, Tupelbildung und Abbildung siehe MARTIN [Martin 1999: S. 37 ff.], BOCK und ODELL [Bock 1997], [Bock 1997b] oder VETTER [Vetter 1995: S. 110]. ZÜLLIGHOVEN bezeichnet die Assoziation als Benutzt-Beziehung. Vgl. [Züllighoven 1998: S. 40]. SCHIENMANN bezeichnet die Assoziation als *Relation*. Vgl. [Schienmann 1997: S. 206].

[2] Korrekt muss es heißen, dass die Urbildmenge in die Potenzmenge des Bildes abgebildet wird. Dadurch kann einem Element der Urbildmenge eine Menge von Elementen der Bildmenge zugewiesen werden. Vgl. [Odell 1997].

[3] Vgl. [Rumbaugh 1993: S. 34], [Riehle 2000: S. 27].

[4] Vgl. [Papurt 1994].

[5] Vgl. [Peckham 1988], [Chen 1976], [Linssen 1999e], [IDEF1X 1993b].

Bildmenge gehören jeweils einer Klasse oder deren Spezialisierungen an. Assoziationen zwischen den Exemplaren einer einzigen Klasse sind ebenfalls möglich.[1]

Die *Assoziation* wird in der objektorientierten Modellierung zur Bildung von Beziehungen zwischen Konzepten verwendet,[2] wenn diese Beziehungen nicht durch andere Beziehungsarten ausgedrückt werden.[3] Sie gilt als die semantisch schwächste und allgemeinste der referentiellen Beziehungstypen.[4] Assoziationen erhalten in der Regel einen Namen, der als Bezeichnung der Zuordnungsvorschrift verwendet wird.[5] Häufig wird diese Zuordnungsvorschrift durch ein Verb benannt:[6]

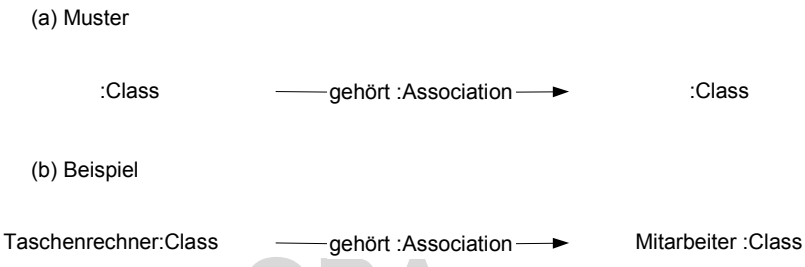


Abbildung 49: Muster und Beispiel einer Assoziationsbeziehung

[1] Beispielsweise kann ein Bauteil einer Maschine selbst aus mehreren Bauteilen bestehen. Dies wird als *rekursive Assoziation* bezeichnet.

[2] Die Assoziationsbeziehung beinhaltet hier sowohl Beziehungen auf Objekt- wie auch auf Klassenebene. In der UML wird dagegen zwischen *Assoziation* und *Link* unterschieden. *Assoziationen* bilden die Beziehungen zwischen Klassen, *Links* bilden die Beziehungen zwischen Objekten ab. Vgl. [Alhir 1998: S. 57 f.], [Hitz 1999]. Sie stellen außerdem die Pfade dar, auf denen Objekte Nachrichten austauschen (vgl. [Booch 1999: S. 206, 207, 209]). HEIDE BALZERT bezeichnet Assoziationen als permanente Beziehungen und Kommunikationsbeziehungen (die hier als Interaktion bezeichnet werden; vgl. 4.3.4.) als temporäre Beziehungen. Vgl. [BalzertH 1999: S. 76].

[3] Vgl. [Henderson-Sellers 2000: S. 79].

[4] Vgl. [Booch 1995: S. 141].

[5] Die Assoziation entspricht in etwa der *Relationship* im relationalen Paradigma, wird aber anders realisiert: Die *Relationship* basiert auf übereinstimmenden Werten von Attributen, die Assoziation basiert auf einer Tupelbildung über Objektidentitäten.

[6] Vgl. [Rumbaugh 1993: S. 34].

Im Konzeptuellen Schema entsprechen diese Beziehungen Knoten der Metaklasse *ASSOCIATION*. Assoziationen werden hier gerichtet verwendet.[1] Die Assoziation geht vom Knoten des Urbilds (*Domain*) aus und führt zum Knoten des Bildbereichs (*Range*). Im Konzeptuellen Schema verbinden Beziehungsknoten des Metatyps *ASSOCIATION* Knoten des Metatyps *CLASS*.

4.3.3.2 Ganzes-Teil

Diese Beziehung stellt die referentielle, mereologische[2] Beziehung zwischen Teilen und einem Ganzen dar. Die Ganzes-Teil-Beziehung fasst alle Beziehungstypen zusammen, in denen ein Konzept Anteil an einem anderen Konzept hat. Sie ist das Ergebnis einer Hierarchiebildung in Form von Ganzes-Teil-Beziehungen. Auf diese Weise werden komplexe Konzepte aus elementareren Konzepten zusammengesetzt bzw. elementarere Konzepte dienen als Bausteine für komplexerer Konzepte. Die Ganzes-Teil-Beziehung geht vom Ganzen aus und endet bei den Teilen. Die Ganzes-Teil-Beziehung ist – mit Einschränkungen – transitiv[3], asymmetrisch[4] und für Objekte nicht reflexiv.[5]

Die Ganzes-Teil-Beziehung wird umgangssprachlich mit unterschiedlicher Bedeutung verwendet[6] und auch in der objektorientierten Modellierung in der Regel nicht präzise definiert.[7] Zur Differenzierung der unterschiedlichen Arten von Ganzes-Teil-Beziehungen in den Abschnitten 4.3.3.2 bis 4.3.3.5 wird folgende Tabelle verwendet:[8]

---

[1] In der UML werden Assoziationen bidirektional interpretiert. Bidirektionale Beziehungen werden hier durch zwei unidirektionale Kanten dargestellt. Dies folgt dem Ansatz von HENDERSON-SELLERS [Henderson-Sellers 1998b: S. 1 ff.] und COOK [Cook 1994: S. 34].

[2] Nach dem griechischen Wort „*meros*“ für ‚Teil‘. Vgl. [Wedekind 1992: S. 35], [Winston 1987: S. 418].

[3] Transitivität liegt nur bei Verwendung gleicher Aggregationstypen vor. Vgl. die Diskussion in [Winston 1987: S. 431 ff.], [Henderson-Sellers 1997: S. 5].

[4] Vgl. [Englmeier 1997: S. 27], [Henderson-Sellers 2000: S. 81].

[5] Vgl. [Winston 1987: S. 418], [Helbig 1996: S. 110], [Henderson-Sellers 2000: S. 81].

[6] Vgl. [Winston 1987], [Borgida 1984], [Bock 1994], [Martin 1995], [Henderson-Sellers 1998], [Firesmith 1998], [Smith 1977], [Smith 1977b], [Henderson-Sellers 1998: Appendix E; Stichwort „Aggregation, membership and containment“], [Henderson-Sellers 2000: S. 80 ff.], [Schienmann 1997: S. 201 ff.], [Benjamin 1994].

[7] Vgl. [Lano 1995: S. 52].

[8] Übersetzung von [Firesmith 1998: S. 48]. Jeder Autor verwendet und verändert diese Tabelle nach eigenen Ansichten. Einen ähnlichen Aufbau findet man beispielsweise auch bei MARTIN. Vgl. [Martin 1999: S. 253 ff.]. Er lässt aber die Form Feature-Aktivität weg, weil sie seines Erachtens eine funktionale Dekomposition impliziert und deshalb in einem objektorientierten Umfeld nicht erwähnt werden sollte. HEIDE BALZERT bezeichnet die Ort-Bereich-Aggregation als gleichartig. Als Beispiel wird genannt: „*München ist*

	Konfigurierend	**Gleichartig**	**Invarianz**
Aggregation (Aggregation)			
Komposition (Component-Integral Object)	Ja	Nein	Nein
Material-Objekt-Komposition (Material-Object)	Ja	Nein	Ja
Feature-Aktivität (Feature-Activity)	Ja	Nein	Ja
Ort-Bereich (Place-Area)	Ja	Nein	Ja
Mitgliedschaft (Membership)			
Portion-Objekt-Komposition (Portion-Object)	Nein	Ja	Nein / Ja
Kollektion (Member-Bunch)	Nein	Nein	Nein
Partnerschaft (Member-Partnership)	Nein	Nein	Ja
Konfigurierend („Configurational"): Die Teile beinhalten eine spezifische funktionelle oder strukturelle Eigenschaft untereinander oder zum Ganzen.			
Gleichartig („Homeomerous"): Wenn die Teile von der gleichen Art sein müssen wie das Ganze.			
Invarianz („Invariance"): Wenn die Teile nicht aus dem Ganzen entnommen werden können, ohne das Ganze zu zerstören.			

Tabelle 5: Ausprägungen der Ganzes-Teil-Beziehung

Konfigurierend ist eine Beziehung dann, wenn die Teile in ihrer Gesamtheit für die Funktion des Ganzen notwendig sind. Die Einzelteile eines Motors sind insgesamt für des Funktion notwendig.[1] *Invarianz* liegt für eine Ganzes-Teil-Beziehung dann vor, wenn die Lebensdauer eines Teils mit der Lebensdauer des Ganzen übereinstimmt. Kein Teil kann entfernt werden, ohne das Ganze zu vernichten. *Gleichartigkeit* liegt vor, wenn Teil und Ganzes von der gleichen Art (Klasse) sind. Diesen unterschiedlichen Eigenschaften der Ganzes-Teil-Beziehung wird durch zwei Beziehungsknotentypen Rechnung getragen:

- **Aggregation**: Die Teile besitzen eine funktionale Beziehung untereinander und zum Ganzen. Es liegt eine konfigurierende Beziehung vor.
- **Mitgliedschaft**: Das Teil ist Element des Ganzen. Es liegt keine konfigurierende Beziehung vor.

---

ein Teil von Bayern." Üblicherweise würde man München aber als Stadt und Bayern als Bundesland bezeichnen, wodurch die Gleichartigkeit zumindest in Frage gestellt werden kann. [BalzertH 1999: S. 49].

[1] Vgl. [Firesmith 1998].

Im Konzeptuellen Schema verbinden Beziehungsknoten des Metatyps *WHOLE-PART* Knoten des Metatyps *CLASS*.

4.3.3.3 Aggregation

Die Aggregationsbeziehung ist eine Spezialisierung der Ganzes-Teil-Beziehung.[1] Mit *Aggregation* wird die Beziehung zwischen einem Aggregat und seinen Komponenten bezeichnet.[2] Dabei besteht zwischen den Teilen des Aggregats eine zeitliche, räumliche oder instrumentelle Abhängigkeit.[3] Aus diesem Grund muss ein Aggregat aus mindestens zwei Teilen bestehen.[4] Im Metamodell entspricht die Aggregation einem Knoten des Metatyps *AGGREGATION*, durch die zwei Knoten des Metatyps *CLASS* verbunden werden. Die Beziehung geht vom Aggregat aus und endet bei den Teilen des Aggregats. Nach LANO liegt dann eine Aggregationsbeziehung vor, wenn

- das Löschen des Ganzen zum Löschen der Teile führt.
- sich die Teile während der Lebensdauer des Ganzen nicht verändern können, ohne das Ganze zu verändern,
- ein Teil nur Teil *eines* Ganzen sein kann.[5]

[1] Anders in der UML. Dort wird die Assoziation als allgemeinster Fall der Beziehung angesehen und die Aggregation als Spezialfall der Assoziation verwendet (vgl. [Rumbaugh 1999: S. 146]).

[2] Mit Aggregation wird im Duden eine Anhäufung von Kenntnissen und Fakten bezeichnet, mit Aggregat ein Satz von zusammenwirkenden einzelnen Maschinen, Apparaten, Teilen. [Duden 1993].

[3] Vgl. [Firesmith 1998: S. 3], [Schienmann 1997: S. 202].

[4] Vgl. [Firesmith 1998b: S. 4]

[5] Vgl. [Lano 1995: S. 53].

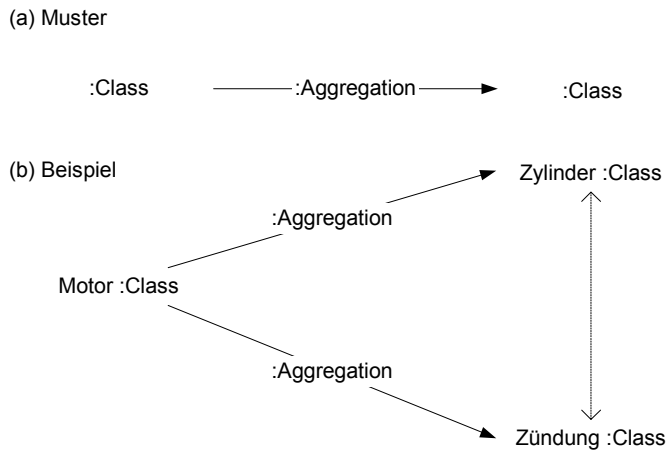


Abbildung 50: Muster und Beispiel einer Aggregationsbeziehung[1]

Die Aggregationsbeziehung wird mit unterschiedlichen Bedeutungen verwendet:[2]

- **Komposition** (*Component-Integral Object Composition*): Diese Beziehung definiert, aus welchen Teilen ein Ganzes besteht. Die Teile besitzen eine besonders starke funktionale oder strukturelle Beziehung untereinander und zum Ganzen. Struktur und Verhalten der Komposition wird durch die Anordnung der Teile bestimmt.[3] Ein Teil kann nur Element eines Ganzen sein. Die Lebensdauer eines Teils entspricht der Lebensdauer des Ganzen. Umgekehrt gilt dieser Zusammenhang nicht, da einzelne Teile durchaus entfernt werden können, ohne das Ganze zu vernichten. Beispiel: Der Film besteht aus Szenen. Wird der Film vernichtet, sind auch die Szenen vernichtet. Entnimmt man dem Film eine Szene, ist dadurch nicht gleichzeitig der Film zerstört. Ein Motor besteht (u. a.) aus Zylindern. Wird der Motor verschrottet, sind auch die Zylinder verschrottet worden, es sei denn, sie wurden vorher entfernt.

[1] In dieser Abbildung zur Aggregationsbeziehung deutet die vorhandene Kante zwischen Zylinder und Zündung an, dass eine irgendwie geartete Beziehung besteht. Diese Verbindung ist kein Element des Metamodells, sondern dient der Veranschaulichung.

[2] Die Diskussion über die präzise Semantik von Aggregationsbeziehungen kann momentan noch nicht als abgeschlossen betrachtet werden. Die hier verwendete Differenzierung gibt den aktuellen Stand der Forschung wieder. Vgl. [Henderson-Sellers 1997], [Firesmith 1998], [Bock 1998b].

[3] Vgl. [Schienmann 1997: S. 202]. Dies wird von SCHIENMANN als *Ganzheit* bezeichnet.

- **Material-Objekt Komposition** (*Material-Object Composition*): Die Teile machen das Ganze in seinem Wesen[1] aus. Sie können nicht entfernt werden, ohne das Ganze als Wesen zu verändern. Beispiel: Das Brot besteht zum Teil aus Mehl.
- **Feature-Aktivität** (*Feature-Activity*): Beispiel: Bezahlen ist ein Bestandteil vom Essen gehen.
- **Ort-Bereich** (*Place-Area*): Die Teile können nicht vom Ganzen getrennt werden. Beispiel: Wuppertal ist ein Teil von Deutschland.

4.3.3.4 Mitgliedschaft

Die Mitgliedschaft-Beziehung (*Membership*) ist die andere Spezialisierung der Ganzes-Teil-Beziehung. Sie repräsentiert eine Beziehung von einer Gruppe zu ihren Elementen. Im Unterschied zur Aggregation besitzen die Teile der Beziehung nicht unbedingt eine funktionale Abhängigkeit untereinander. Für die Existenz dieser Beziehung ist aber mindestens ein Element notwendig.[2] Im Konzeptuellen Schema entspricht die Mitgliedschaft-Beziehung einem Knoten der Metaklasse *MEMBERSHIP*, durch die zwei Knoten des Metatyps *CLASS* verbunden werden. Beispiele mit unterschiedlicher Bedeutung der Mitgliedschaft-Beziehung sind:[3]

- **Portion-Objekt-Komposition** (*Portion-Object Composition*): Das Teil ist das gleiche Konzept wie das Ganze. Beispiel: In einer Flasche Champagner sind fünf Glas Champagner.[4]
- **Kollektion** (*Member-Bunch Composition*): Das Teil ist Element einer Menge, durch die das Ganze entsteht. Beispiel: Der Baum ist Bestandteil des Waldes.[5]
- **Partnerschaft** (*Member-Partnership*): Wird ein Element der Partnerschaft entnommen, ist die Partnerschaft als Ganzes zerstört. GINGER und FRED sind ein Tanzpaar. Ehemann und Ehefrau bilden eine Ehegemeinschaft.[6] ATHOS, PORTHOS, ARAMIS und D'ARTAGNAN bilden DIE DREI MUSKETIERE.

---

[1] *Wesen* wird im folgenden Sinn verwendet: *„Das Wesen eines Dinges, Prozesses usw. durchdringt und bestimmt alle seine Eigenschaften, Merkmale usw. Es erscheint jedoch nie in reiner Form, sondern stets in spezifischer und modifizierter Gestalt, gewissermaßen als Projektion des Wesens auf der Ebene der Erscheinung."* [Klaus 1972: S. 1159].

[2] Vgl. [Firesmith 1998b: S. 4].

[3] Vgl. [Henderson-Sellers 1997].

[4] Dies wird von SCHIENMANN als *Verschmelzung* bezeichnet. Vgl. [Schienmann 1997: S. 203].

[5] Ähnlich in [Schienmann 1997: S. 202].

[6] Dies wird von SCHIENMANN auch als *Konnexion* bezeichnet. Vgl. [Schienmann 1997: S. 205].

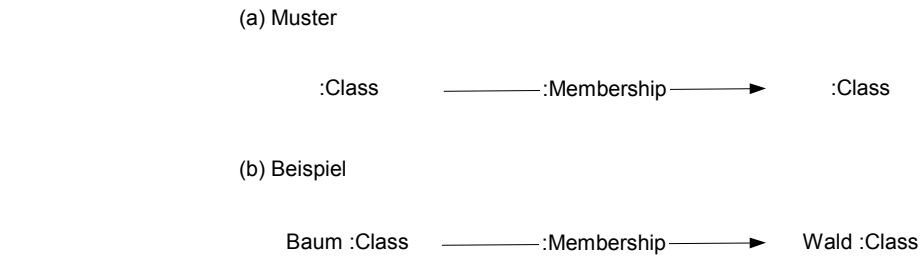


Abbildung 51: Muster und Beispiel der Mitgliedschaft-Beziehung[1]

4.3.3.5 Containerbeziehung

Ein Container ist nach STROUSTRUP ein Objekt, welches andere Objekte enthält.[2] Container-Beziehungen sind Beziehungen zwischen einem Behälter und seinem Inhalt.[3] Die Containerbeziehung[4] stellt keine Ganzes-Teil-Beziehung dar. Eine Containerbeziehung liegt dann vor, wenn ein Konzept andere Konzepte beinhaltet, ohne dass diese zu einem Bestandteil des ersten Konzepts werden.[5] Ein Beispiel: Durch das Betreten eines Geschäfts durch einen Kunden befindet sich dieser in dem Geschäft, ohne zu einem Bestandteil des Geschäfts zu werden. Der Charakter des Containerkonzepts verändert sich auch nicht durch eine Veränderung der Teile. Dies soll an dem Beispiel eines Reisebusses verdeutlicht werden: Der Charakter des Konzepts Bus verändert sich nicht durch eine Veränderung der Anzahl der Insassen. Im Metamodell entspricht die Containerbeziehung einem Knoten der Klasse *CONTAINMENT*. Sie verbindet Klassenknoten.

[1] Da die Mitgliedschaft-Beziehung eine Spezialform der Ganzes-Teil-Beziehung ist, muss sie auch deren Ausrichtung übernehmen. Dadurch ergibt sich die etwas kontraintuitive Sicht, dass die Beziehung beim Ganzen beginnt und bei den Teilen endet.

[2] [Stroustrup 2000: S. 462].

[3] Vgl. [Züllighoven 1998: S. 89 f.].

[4] Hier wurde *‚Containment‘* frei mit *‚Container‘* übersetzt, weil dies sprachlich passender erschien als die etwas umständliche Formulierung ‚Beinhaltetsein‘. Dies wird von SCHIENMANN als *Behältnis* bezeichnet. Vgl. [Schienmann 1997: S. 202].

[5] Vgl. [Henderson-Sellers 1998: Appendix E: Stichwort „Containment“]. Weit verbreitet ist die Anwendung der Containerbeziehung im objektorientierten Entwurf, wo eine Klasse *Kunden* (= Menge aller Kunden) als Container für die Klasse *Kunde* (= ein einzelner Kunde) verwendet wird. Im hier entwickelten Metamodell handelt es sich jedoch um eine Mitgliedschaft, die durch eine Beziehung des Metatyps *MEMBERSHIP* abgebildet wird.

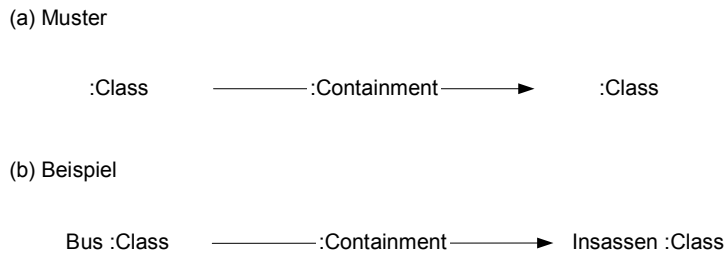


Abbildung 52: Muster und Beispiel einer Containerbeziehung

4.3.3.6 Referentielle Beziehungen im Konzeptuellen Schema

Damit ergibt sich für die referentiellen Beziehungen im Konzeptuellen Schema folgende Struktur:

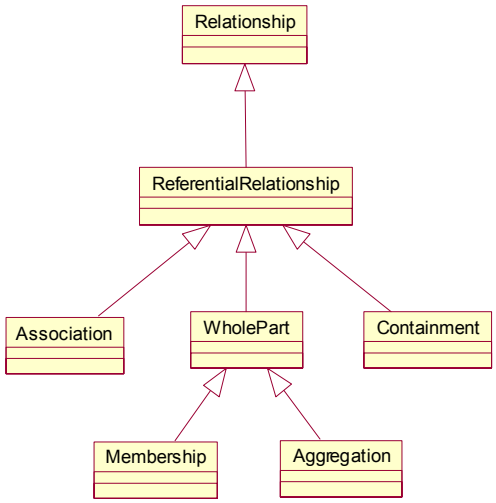


Abbildung 53: Referentielle Beziehungen im Konzeptuellen Schema

4.3.4 Intensionsbeziehung

In der OML wird die Beziehung zwischen einer Klasse und einem Attribut als interne referentielle Beziehung angesehen und unter die referentiellen Beziehungen subsumiert.[1] Dieser Beziehungstyp wird dort als *Attribution* bezeichnet. Diese Lösung ist hier nicht übertragbar, da die Attribution auf

[1] Vgl. [Firesmith 1998b].

die Verbindung von Attributen mit Klassen beschränkt ist. Es fehlt zum Beispiel ein Beziehungstyp, der Klassen mit ihren Operationen verbindet. In keinem bekannten objektorientierten Modellierungsansatz existiert ein solcher Beziehungstyp, der Klassen mit ihrer Intension (Attribute, Operationen etc.) verbindet. Durch die hier verwendete Intensionsbeziehung werden Klassenknoten (*CLASS*) mit Intensionsknoten (*INODE*) verbunden. Sie wird im Metamodell des Konzeptuellen Schemas durch den Beziehungsknotentyp *INTENSION* repräsentiert:[1]

Für den Beziehungsknotentyp *INTENSION* gilt: Ein Knoten des Metatyps *CLASS* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *INTENSION* mit mehreren Knoten des Metatyps *INODE* verbunden sein. Ein Knoten des Metatyps *INODE* ist im Konzeptuellen Schema über einen Beziehungsknoten des Metatyps *INTENSION* immer mit genau einem Knoten des Metatyps *CLASS* verbunden. Das bedeutet, dass Klassen mehrere Intensionsknoten haben können. Es gibt Klassen, die keine Intensionsknoten besitzen.[2] Ein Intensionsknoten gehört immer zur Intension exakt einer Klasse; sein Fan-In ist eins. Dies entspricht folgendem UML-Klassendiagramm:

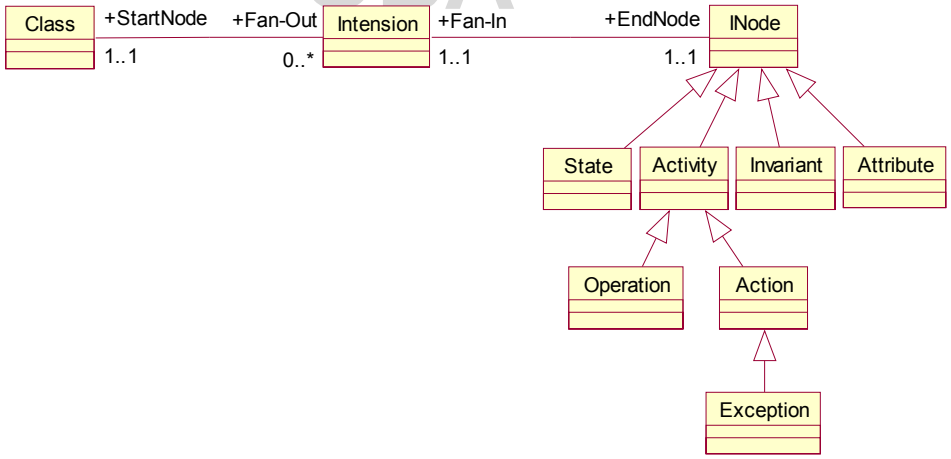


Abbildung 54: Beziehungsknotentypen des Metatyps INTENSION im Konzeptuellen Schema

[1] Der erwähnte Beziehungstyp Attribution der OML wäre eine Teilmenge der hier verwendeten Intensionsbeziehung. Es wäre möglich, ihn von *INTENSION* zu spezialisieren.

[2] Begründung: Der Fall von Klassen, die nur referentielle Beziehungen mit anderen Klassen besitzen, soll hier nicht explizit ausgeschlossen werden.

Die Intensionsbeziehung drückt eine gerichtete 1-zu-n-Merkmalsbeziehung aus. Die verbundene Eigenschaft ist danach ein Merkmal bzw. eine Charakteristik genau einer Klasse. Ist dies nicht der Fall, handelt es sich um eine referentielle Beziehung zwischen Klassen.[1] Die Beziehung geht vom Klassenknoten aus und endet beim Intensionsknoten.

Sie ist asymmetrisch und nicht reflexiv. Das bedeutet, dass ein Knoten nicht gleichzeitig Intension einer Klasse sein kann und jene Klasse zur Intension des Knotens gehört. Außerdem kann ein Knoten nicht Intension von sich selbst sein.

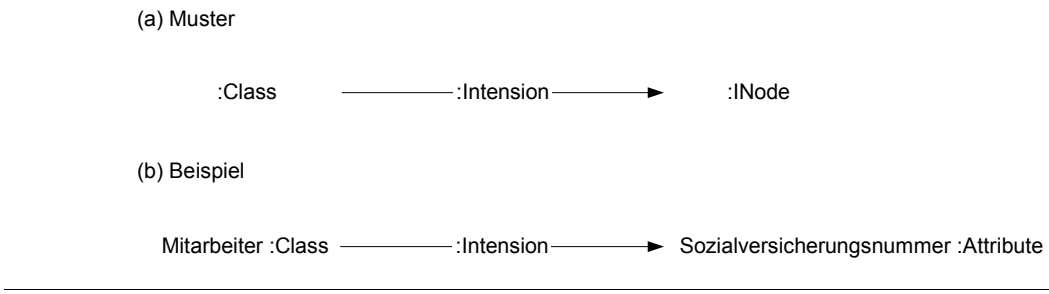


Abbildung 55: Muster und Beispiel einer Intensionsbeziehung

4.3.5 Dynamik ausdrückende Beziehungen

4.3.5.1 Interaktionsbeziehung

Objektorientierte Systeme bestehen aus einer Menge miteinander interagierender[2] Objekte. Im objektorientierten Ansatz werden nach FIRESMITH und HENDERSON-SELLERS alle Mechanismen, durch die Konzepte interagieren, als *Interaktion* bezeichnet.[3] Im Folgenden wird der Terminus

[1] Siehe hierzu die Abschnitte 4.2.2 und 4.3.3.

[2] *interagieren* = sich wechselseitig beeinflussen. *Interaktion* = aufeinander bezogenes Handeln. Im objektorientierten Ansatz wird der Begriff Interaktion dem Begriff der Kommunikation vorgezogen, da Interaktion ein allgemeineres Konzept darstellt. Vgl. hierzu auch die Darstellung von Schienmann [Schienmann 1997: S. 11 ff.], der die unterschiedlichen Arten von *Geschehnissen* einordnet und die Kommunikation unter die Interaktion subsumiert. [Schienmann 1997: Abb. 4-4].

[3] [Henderson-Sellers 1998], [Firesmith 1995: S. 219]. Andere Bezeichnungen für diesen Beziehungstyp sind *Message Connection* (vgl. [Coad 1991: S. 149]), *Verwendung* (vgl. [Booch 1995: S. 168 f.]), *Kollaboration* oder *uses*-Beziehung (vgl. [Kappel 1996: S. 40 ff.]), *Client-Server*-Beziehung (vgl. [Selic 1994: S. 55 ff.]), *Client*-Beziehung (vgl. [Waldén 1995: S. 69 ff.]), *Benutzt-Beziehung* (vgl. [Züllighoven 1998: S. 40]) oder *Kooperation* (vgl. [Wirfs-Brock 1993: S. 91 ff.]). In der UML wird das Verhalten einer Kollaboration als

Interaktion für alle Formen der Nachrichtenübertragung oder Sender-Empfänger-Beziehung verwendet, wenn

- Objekte von einem Objekt zu einem anderen Objekt bewegt werden,[1]
- Objekte aufgefordert werden, in irgendeiner Form Verhalten zu zeigen,[2]
- Objekte und Informationen von Objekten erfragt werden,
- besondere Situationen angezeigt werden, [3]
- Objekte Aufgaben an andere Objekte weiterreichen (*Delegation*),[4]
- ein Objekt seinen Zustand verändert und andere Objekte darauf reagieren,[5]
- Objekte andere Objekte erzeugen und löschen,[6]
- Objekte Ereignisse erzeugen und andere Objekte darauf reagieren,[7]
- Objekte ihre Rolle wechseln,[8]
- ein Objekt eine Verrichtung an einem anderen Objekt ausführt.

Da jede Interaktion als ein Ereignis angesehen wird, werden auf diese Weise die dynamischen Aspekte eines Systems modelliert.[9] Das Konzept, von dem die Interaktion ausgeht, heißt *Initiator*. Das andere Konzept, welches an der Interaktion beteiligt ist, heißt *Participant*.[10] Der Initiator ist der initiierende Partner der Interaktion, der Participant ist der (andere) partizipierende Teil. Bei der

---

Interaktion bezeichnet. Eine Kollaboration besteht aus einer Anzahl von miteinander verbundenen Exemplaren (Objekten), die untereinander Nachrichten austauschen. Vgl. [Rumbaugh 1999: S. 109].

[1] Vgl. [OMG 1999: S. 3-147].

[2] Vgl. [Firesmith 1995: S. 219], [Firesmith 1995: S. 53].

[3] Vgl. [Booch 1999: S. 279], [Rumbaugh 1999: S. 419], [Firesmith 1995: S. 163].

[4] Vgl. [Firesmith 1995: S. 126].

[5] In der UML wird dies nicht als Interaktion, sondern als *time event* und *change event* bezeichnet. Vgl. [Booch 1999: S. 281], [Rumbaugh 1999: S. 268 f.].

[6] Vgl. [Booch 1999: S. 206, 211].

[7] Vgl. [Henderson-Sellers 2000: S. 69].

[8] Vgl. [Booch 1999: S. 139, 166, 190, 253 f.].

[9] Vgl. [Booch 1999: S. 206].

[10] Häufig werden die beteiligten Parteien als *Sender* und als *Empfänger* bezeichnet. Vgl. [Rumbaugh 1999: S. 334], [Firesmith 1995: S. 405]. Um die technische Konnotation dieser Begriffe zu vermeiden, wurden die Begriffe Initiator und Participant aus der OBA übernommen. Vgl. [Rubin 1992]. Eine mögliche Übersetzung des Begriffs Participant wäre *Partizipant*, *Teilnehmer*, *Bearbeiter* oder *Beteiligter*. In der Methode CATALYSIS heißen beide beteiligten Objekte *Participant*. Vgl. [D'Souza 1999: S. 4].

Interaktionsbeziehung wird davon ausgegangen, dass Initiator und Participant im Rahmen der Interaktion Aktivitäten ausführen. Der Grundsatz der objektorientierten Modellierung, dass die Interaktion zwischen Objekten nur durch die Anforderung einer Operation stattfindet,[1] wird hier in der Form erweitert, dass der Initiator eine Aktion und der Participant eine Operation ausführt. Im Unterschied zu anderen objektorientierten Modellierungsmethoden wird durch den Metatyp *ACTION* in Interaktionen auch abgebildet, welche Aktivitäten die Ursache einer Operation sind. Die Interaktionsbeziehung der OBA++ betrachtet also nicht nur die Empfänger-, sondern auch die Senderseite. Darin unterscheidet sie sich von anderen objektorientierten Modellierungsansätzen, die nur die Empfängerseite betrachten.

Im Metamodell des Konzeptuellen Schemas entspricht die Interaktionsbeziehung dem Beziehungsknotentyp *INTERACTION*. Die Interaktionsbeziehung geht von der Aktion des *Initiators* aus und endet bei der Operation des *Participants,* um auszudrücken, dass der Participant und der Initiator durch ihre Aktivitäten interagieren.

Dabei ist zu beachten, dass Aktivität hier nicht im umgangssprachlichen Sinn verstanden werden kann. Die Verrichtung einer Person an einem Gegenstand beinhaltet im hier verwendeten OA eine Aktion der Person und eine Operation des Gegenstands, wobei die Operation das Verhalten bzw. die Veränderung des Gegenstands beschreibt. Das folgende Beispiel zeigt, wie ein Exemplar der Klasse Abteilungsleiter in der Rolle des Vorgesetzten eine Interaktion mit einem Sachbearbeiter in der Rolle des Untergebenen eingeht.

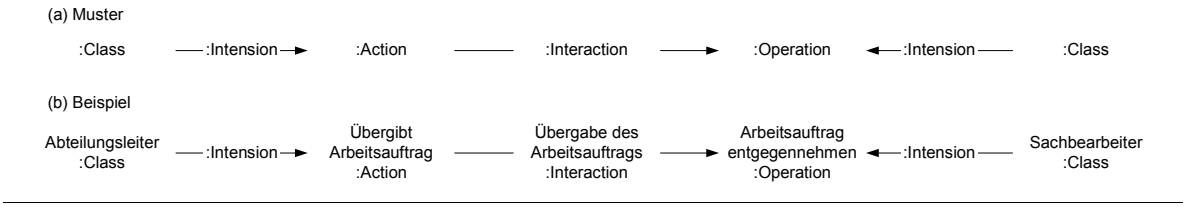


Abbildung 56: Muster und Beispiel einer Interaktionsbeziehung

Für den Beziehungsknotentyp *INTERACTION* gilt: Ein Knoten des Metatyps *ACTION* oder *EXCEPTION* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *INTERACTION* mit mehreren Knoten des Metatyps *OPERATION* verbunden sein. Ein Knoten des Metatyps *OPERATION* ist im Konzeptuellen Schema über Beziehungsknoten des Metatyps *INTERACTION* mit mindestens einem Knoten des Metatyps *ACTION* verbunden; sein Fan-In ist größer oder gleich 1. Das bedeutet, dass es keine Operationen gibt, die nicht durch mindestens eine

[1] Vgl. [Coleman 1992: S. 9].

Aktion ausgelöst werden. Im Unterschied zu den statischen Beziehungstypen (*RELATIONSHIP*) ist bei der Interaktionsbeziehung (*INTERACTION*) zu beachten, dass diese Konzeptknoten immer über Intensionsknoten verbindet. Dies entspricht folgendem UML-Klassendiagramm:

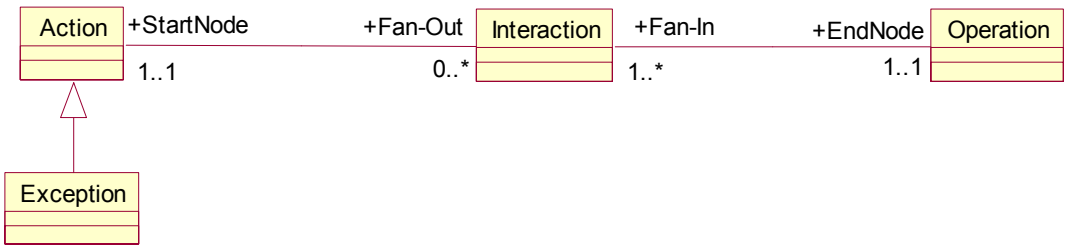


Abbildung 57: Beziehungsknotentypen des Typs INTERACTION im Konzeptuellen Schema

Die Interaktionsbeziehung ist die einzige Beziehung, durch die Objekte aufeinander einwirken können.[1] Durch sie findet die Interaktion mit einem Objekt immer über dessen Operationen statt. Es ist weder möglich, Attribute durch andere Objekte direkt zu verändern noch den abstrakten Zustand eines Objekts.

Auch die Objektflussbeziehung wird als Interaktionsbeziehung abgebildet. Der *Objektfluss* der UML ist eine Form von Kontrollfluss, durch den ein Objekt mit den Objekten und Aktivitäten verbunden wird, die es als Eingabe oder Ausgabe verwenden.[2] Durch die Aktivitäten werden Objekte be- und verarbeitet.[3] Der Objektfluss wird aktiv, wenn die Aktivität am Ausgangspunkt ausgeführt wurde. Endpunkt des Objektflusses ist eine andere Aktivität, die das fließende Objekt als Eingabe verwendet.[4] Der *Objektflusszustand* ist der Zustand, den ein Objekt zu einem

[1] Aus der Sicht der objektorientierten Programmierung mag eingewendet werden, dass die Interaktionsbeziehung untrennbar von der referentiellen Beziehung ist, da ein Objekt eine Referenz auf ein anderes Objekte besitzen muss, damit es mit ihm interagieren kann. Dem wird hier nicht widersprochen. Hier muss aber beachtet werden, dass für den Entwurf eines Metamodells zwischen der statischen und dynamischen Sichtweise zu unterscheiden ist. Im anderen Fall müsste man Interaktionen zwischen Objekten durch Assoziationsbeziehungen abbilden, was in der UML nicht zugelassen ist. [OMG 1999: S. 3-91]. Außerdem existieren referentielle Beziehungen auch, wenn zu einem bestimmten Zeitpunkt keine Interaktionen stattfinden. Daraus ergibt sich, dass beide Beziehungstypen zu trennen sind.

[2] Vgl. [Rumbaugh 1999: S. 84, 363].

[3] Vgl. [OMG 1999: S. 3-147].

[4] Vgl. [Rumbaugh 1999: S. 364], [OMG 1999: S. 2-160].

bestimmten Zeitpunkt der Bearbeitung besitzt.[1] Er stellt eine Information dar, welchen Zustand ein Objekt einer Klasse hat, wenn es über einen Objektfluss von einer Aktivität zu einer anderen Aktivität fließt. Dadurch wird sichtbar, in welcher Form ein Objekt zu einem bestimmten Zeitpunkt der Bearbeitung auftritt.[2] Der Objektflusszustand wird auch verwendet, wenn Objekte in einem bestimmten Zustand die Eingabe einer Aktivität darstellen.[3] Die Aktivität am Endpunkt der Objektflussbeziehung erhält das fließende Objekt im jeweiligen Objektflusszustand.

Das fließende Objekt stellt den Inhalt der Interaktionsbeziehung dar und fließt von der Aktion des Initiators zur Operation des Participants. Ein Objektfluss zwischen zwei Aktivitäten, die nicht zur Intension irgendeiner Klasse gehören, wird nicht unterstützt, da dies dem Prinzip des objektorientierten Ansatzes widerspricht, dass Aktivitäten immer von Objekten ausgeführt werden. Der Objektflusszustand wird über eine Zustandsveränderung abgebildet (vgl. Abschnitt 4.3.5.2).

4.3.5.2 Zustandsveränderung

Die objektübergreifende Dynamik eines Systems wird durch Interaktionen abgebildet. Zustandsveränderungen bilden dagegen die Dynamik eines Systems ab, indem die Veränderungen der Objekte einzelner Klassen isoliert betrachtet werden. Die durch einen Zustandswechsel verbundenen Zustände gehören also immer zu den Objekten einer Klasse, da das Verhalten einzelner Klassen isoliert beschrieben wird.[4] Eine Zustandsveränderung verbindet einen Start- oder Ausgangszustand mit einem End- oder Zielzustand und bildet auf diese Weise einen möglichen Wechsel des Zustands der Objekte einer Klasse ab. In objektorientierten Modellierungsansätzen werden durch Zustandsveränderungen die Übergänge zwischen abstrakten Zuständen abgebildet. Eine Abfolge von Zustandsveränderungen vom Zeitpunkt der Erzeugung eines Objekts bis zu seiner Löschung bezeichnet man als *Lebenszyklus* eines Objekts.[5] Der Beziehungsknotentyp *STATECHANGE* verbindet Intensionsknoten des Metatyps *STATE*.

---

[1] Vgl. [Rumbaugh 1999: S. 364], [OMG 1999: S. 2-154].

[2] Vgl. [OMG 1999: S. 3-147].

[3] Vgl. [Rumbaugh 1999: S. 136].

[4] Vgl. [Rumbaugh 1993: S. 110].

[5] Vgl. [Rumbaugh 1999: S. 479], [Shlaer 1992].

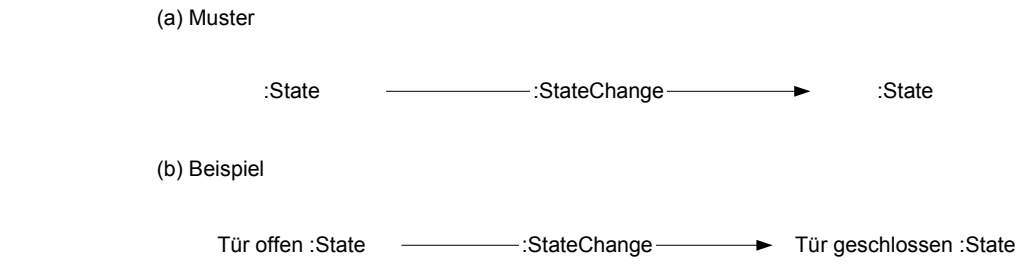


Abbildung 58: Muster und Beispiel einer Zustandsveränderung

Die Beschreibung von Zustandsveränderungen eines Systems oder eines Objekts basiert auf dem Prinzip, dass durch eine Zustandsveränderung S ein System oder ein Objekt in den Zustand B übergeht, wenn es sich im Zustand A befindet.[1] Damit die Zustandsveränderung eintritt, muss ein Ereignis stattgefunden haben, welches die Zustandsveränderung bewirkt. Dieses Ereignis wird als außerhalb des Objekts stattfindend betrachtet[2] und stellt die Verbindung des Objekts mit seiner Umwelt (den anderen Objekten) her. Der Zusammenhang zwischen Ereignis, Zustandsveränderung und Operation besteht darin, dass ein Objekt durch seine Aktionen Ereignisse erzeugt, auf die andere Objekte mit der Ausführung einer Operation reagieren. Durch diese Operation können Zustandsveränderungen herbeigeführt werden. Alle Veränderungen eines Objekts basieren also letztlich auf seinen Operationen.

Diese Sichtweise weicht von der üblichen zur Modellierung reaktiver Systeme durch die Nuance ab, dass dort die eintretenden Ereignisse direkt den Zustandswechsel bewirken.[3] Sie ergibt sich durch die Beachtung des Kapselungsprinzips, durch welches das Verhalten von Objekten immer durch Operationen herbeigeführt wird.

Für den Beziehungsknotentyp *STATECHANGE* gilt: Jeder Knoten des Metatyps *STATE* ist im Konzeptuellen Schema mit einem oder mehreren Knoten des Metatyps *STATECHANGE* verbunden, der diesen mit einem anderen Knoten des Metatyps *STATE* verbindet. Von einem Zustand können mehrere Zustandsveränderungen ausgehen und zu einem Zustand können mehrere Zustandsveränderungen hinführen. Die Knoten des Typs *STATE* müssen darüber hinaus zur Intension der gleichen Klasse gehören. Im Metamodell des Konzeptuellen Schema ist dies dadurch festgelegt, dass der Metatyp *STATE* über die vom Metatyp *INODE* geerbte Assoziation zum Metatyp *INTENSION* den zugehörigen Knoten des Typs *CLASS* referenziert:

[1] Vgl. [Coleman 1992: S. 10].

[2] Eine Ausnahme bildet die Methode SOM, die auch interne Ereignisse verwendet.

[3] Vgl. [Harel 1987: S. 232].

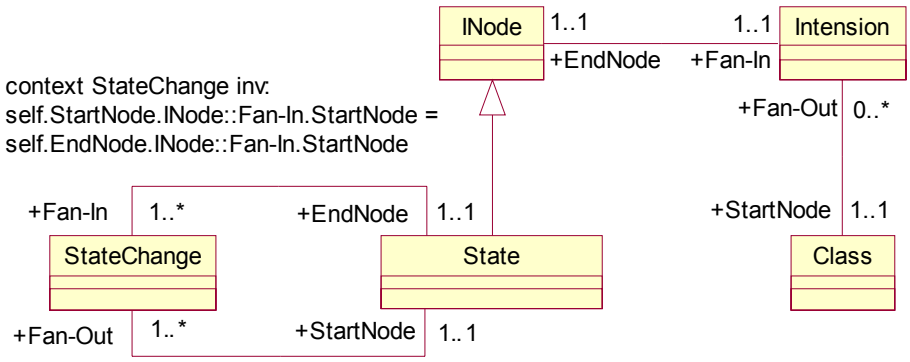


Abbildung 59: Beziehungsknoten des Typs STATECHANGE im Konzeptuellen Schema

In vielen Ansätzen werden Zustandsübergänge mit der Semantik von Transitionen verwendet. Die durch ein Ereignis ausgelöste Zustandsveränderung bzw. den -übergang stellt eine *Transition* dar, wenn sie unteilbar (atomar) ist und ohne Zeitverbrauch stattfindet.[1] Eine Transition benötigt (konzeptionell) keine Zeit und darf nicht unterbrochen werden,[2] da sonst die Situation entstehen könnte, dass ein Objekt seinen Ausgangszustand verlässt und durch einen scheiternden Zustandsübergang seinen Zielzustand niemals erreicht. Das Ergebnis wäre ein nicht definierter Zustand mit nicht vorhersehbarem Verhalten. Demzufolge dürfen auch die im Rahmen einer Transition stattfindenden Aktionen nur atomarer Natur sein.[3] Solange ein Objekt eine Aktion ausführt, reagiert es auf keine eintreffenden Ereignisse.[4] Dies hat aber zur Folge, dass komplexe Prozesse, d. h. Prozesse, an denen mehrere Objekte beteiligt sind, die wiederum nichttriviale Aktivitäten ausführen, keine „echten“ Transitionen darstellen.[5] Wenn sie im Rahmen der objektorientierten Analyse als solche abgebildet werden, dann aus erläuternder Absicht.[6] Von

[1] Vgl. [OMG 1992: S. 37], [Embley 1992: S. 10, 61], [de Champeaux 1993: S. 67, 356].

[2] [Hitz 1999: S. 128]. Hier ist anzumerken, dass diese Restriktion in der Originaldokumentation der OMG durch die Aussage ersetzt wird, dass Transitionen und Aktionen Zeit verbrauchen können. Vgl. [OMG 1999: S. 2-150].

[3] Vgl. [Rumbaugh 1999: S. 434, 482] [de Champeaux 1993: S. 356].

[4] Vgl. [Rumbaugh 1999: S. 441].

[5] *„Actions are not intended for modeling protected regions or long interruptible computations, ...“* [Rumbaugh 1999: S. 441]

[6] Mit dem Problem nicht-trivialer Zustandsübergänge befassen sich Arbeiten im Bereich sog. *langer Transaktionen*. Vgl. [Saake 1993: S. 33]. Dieser Problemkreis wird hier nicht untersucht.

diesen Einschränkungen soll hier nicht ausgegangen werden.[1] Es wird weder vorausgesetzt, dass Zustandsübergänge atomar sind, noch dass die im Rahmen eines Zustandsübergangs stattfindenden Aktivitäten trivial sind.[2] Zustandsübergänge dürfen Zeit verbrauchen. Zwischenzeitlich eintreffende Ereignisse können vom Objekt gepuffert werden, damit sie nicht verloren gehen.

4.3.5.3 Zugriff auf Interna

Der Zugriff auf Interna wird verwendet, um das Lesen oder die Veränderungen der Intension eines Objekts im Zeitverlauf abzubilden. Dieser Beziehungsknotentyp ergibt sich aus den Anforderungen des Information Hiding, dass ein Objekts nicht von außen, sondern nur über dessen Operationen erfolgen darf. Im Konzeptuellen Schema entspricht der Zugriff auf Interna dem Beziehungsknotentyp *INTERNAL*. Er verbindet Intensionsknoten. Mit seiner Hilfe wird ausgedrückt,

- dass sich durch die Aktivität eines Objekt ein Attribut verändert,
- dass sich durch die Aktivität eines Objekt sein abstrakter Zustand verändert,
- dass eine Aktivität eines Objekts eine Invariante tangiert[3].

Beispielsweise wird durch die Aktivität „heiraten“ eines Personenobjekts sein Attribut „Familienstand“ verändert:

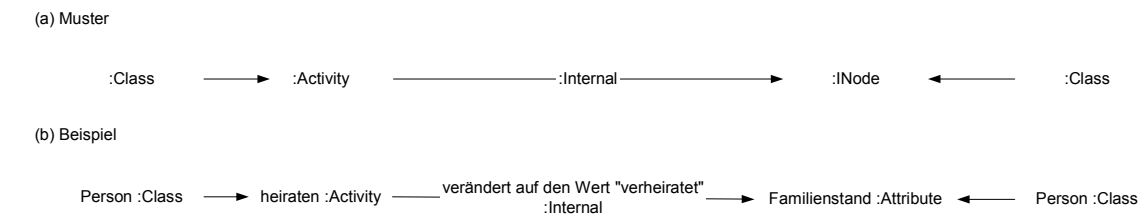


Abbildung 60: Muster und Beispiel eines Zugriffs auf Interna

[1] Wenn es notwendig ist, kann im Einzelfall geprüft werden, ob die Semantik einer Transition anwendbar ist. Gegebenenfalls werden die nicht-atomaren Aktivitäten so lange zerlegt, bis eine Granularität erreicht ist, bei der von atomaren Aktivitäten gesprochen werden kann. Dies erfolgt üblicherweise nicht in der Analyse, weil man dann auf einer Ebene sehr elementarer Aktivitäten modellieren müsste, was einen ungewollten Aufwand darstellt. Vergleiche hierzu auch DE CHAMPEAUX [de Champeaux 1993: S. 356].

[2] Die Forderung nach Trivialität von Aktivitäten ist bei der Analyse realer Systeme auch aus den genannten Gründen fast nie einzuhalten, da in der Realität nahezu jede Aktivität a) scheitern kann und b) in noch elementarere Aktivitäten zerlegbar ist.

[3] Der Begriff *tangieren* wird im Sinne von *zu beachten* verwendet.

Hierbei ist zu beachten, dass diese Beziehung nur verwendet wird, um Abhängigkeiten zwischen den Intensionsknoten eines Konzepts darzustellen. Zwischen Konzepten bestehende Beziehungen der Art *„wenn sich Konzept A verändert, führt dies zu einer Veränderung des Konzepts B“* werden durch Interaktionsbeziehungen abgebildet. Beziehungsknoten vom Metatyp *INTERNAL* verbinden *ACTION*-Knoten mit *INODE*-Knoten.

Für den Beziehungsknotentyp *INTERNAL* gilt: Ein Knoten des Metatyps *ACTION* oder *EXCEPTION* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *INTERNAL* mit mehreren Knoten des Metatyps *INODE* verbunden sein. Ein Knoten des Metatyps *INODE* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *INTERNAL* mit mehreren Knoten des Metatyps *ACTION* oder *EXCEPTION* verbunden sein. Beide Knoten müssen zur Intension der gleichen Klasse gehören:

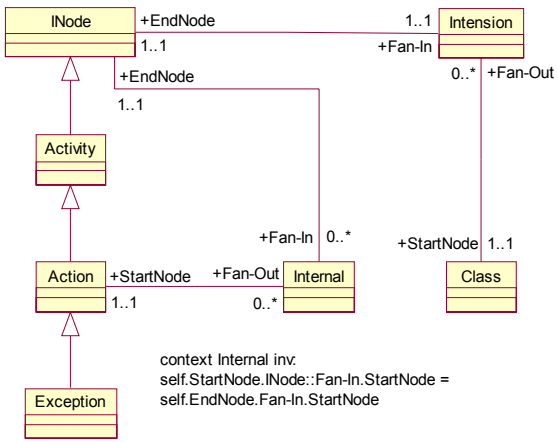


Abbildung 61: Beziehungsknotentypen des Typs INTERNAL im Konzeptuellen Schema

4.3.5.4 Dynamik ausdrückende Beziehungen im Konzeptuellen Schema

Die Beziehungen zur Modellierung der Dynamik bilden im Metamodell des Konzeptuellen Schemas folgende Vererbungsstruktur:

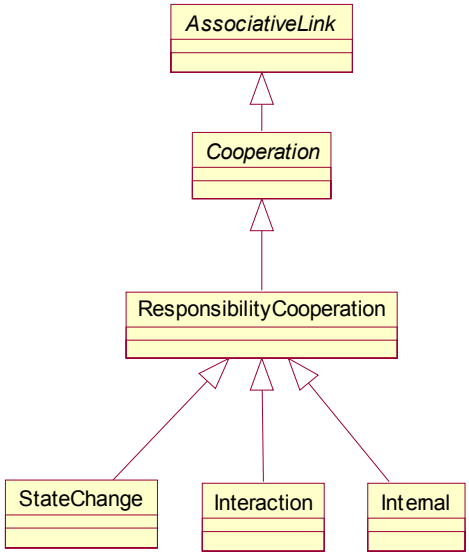


Abbildung 62: Dynamik ausdrückende Beziehungen im Konzeptuellen Schema

4.3.6 Abhängigkeitsbeziehungen

Abhängigkeitsbeziehungen liegen in der UML vor, wenn die Existenz eines Modellelements von der Existenz eines anderen Modellelement abhängt.[1] Sie stellen eine referentielle Beziehung zwischen beliebigen Modellelementen her.[2] Diese Abhängigkeiten werden im Metamodell des Konzeptuellen Schemas durch den Beziehungsknotentyp *DEPENDENCY* repräsentiert. Bei den Abhängigkeitsbeziehungen werden folgende Formen unterschieden:[3]

[1] Vgl. [OMG 1999: S. 2-30], [Rumbaugh 1999: S. 250]. Umgangssprachlich entspricht dies der Formulierung, dass ein Modellelement B von einem Modellelement A abhängig ist, wenn es wissen muss, wenn sich Modellelement A verändert. Vgl. [Hitz 1999: S. 49]. Diese Definition ist in der UML bewusst allgemein gehalten, um die unterschiedlichen Abhängigkeiten, die auftreten können, durch *DEPENDENCY*-Beziehungsknoten abbilden zu können.

[2] Dies unterscheidet die Abhängigkeitsbeziehung von den referentiellen Beziehungen, die nur Knoten des Metatyps *CLASS* verbinden.

[3] In der UML existieren eine Reihe weiterer Abhängigkeitsbeziehungen, die hier nicht benötigt werden, aber bei Bedarf in das hier entwickelte Metamodell aufgenommen werden können. Darüber hinaus lässt die UML die Definition weiterer Abhängigkeitsbeziehungen zu. Vgl. [Rumbaugh 1999: S. 250]. Dort wird auch die Instanzierungsbeziehung unter die Abhängigkeitsbeziehungen subsumiert (vgl. [Booch 1999: S. 137]) – eine Entscheidung, die bei der Ähnlichkeit zwischen Instanzenbildung und Spezialisierung nicht haltbar erscheint, da es sich in beiden Fällen um taxonomische Beziehungen handelt.

- Abläufe in einem System sind Abfolgen von Ereignissen. Zwischen den Ereignissen bestehen *Reihenfolgebeziehungen*. Beispielsweise muss vor einer Interaktion Y eine Interaktion x eingetreten sein.
- Eine Aktivität einer Klasse kommt in einer bestimmten Rolle oder durch ein bestimmtes Objekt zur Anwendung. Dies wird als *Anwendungsbereich* bezeichnet.
- Ein Modellelement wird durch eine Reihe anderer Modellelemente *präzisiert* oder *verfeinert*. Dies wird als *Abstraktionsbeziehung* bezeichnet.

Das besondere an den *DEPENDENCY*-Beziehungen ist, dass sie Verbindungen zwischen allen Metatypen von Modellelementen herstellen. Im Gegensatz zu den Beziehungen des Typs *COOPERATION* und *INTENSION* verbinden sie also auch Beziehungen miteinander. Aus diesem Grund verbinden im Metamodell des Konzeptuellen Schemas Knoten des Metatyps *DEPENDENCY* zwei Knoten des Metatyps *MODELELEMENT*:[1]

Abbildung 63: Beziehungsknotentypen des Typs DEPENDENCY im Konzeptuellen Schema

4.3.6.1 Ereignisfluss

Der Ereignisfluss stellt eine Zusicherung dar, in welcher logisch/kausalen Abfolge Ereignisse stattfinden. Er wird durch den Beziehungsknotentyp *EVENTFLOW* abgebildet. Da *EVENTFLOW*-Beziehungen in verschiedenen externen Sichten zur Dokumentation unterschiedlichster Reihenfolgebeziehungen verwendet werden, verbinden im Metamodell des Konzeptuellen Schemas Beziehungen des Metatyps *EVENTFLOW* zwei Knoten des Metatyps *MODELELEMENT*.[2]

Für den Beziehungsknotentyp *EVENTFLOW* gilt: Ein Knoten des Metatyps *MODELELEMENT* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *EVENTFLOW* mit mehreren

[1] Dies entspricht dem Metamodell der UML. Vgl. [OMG 1999: S. 2-16, Abbildung 2-7].

[2] Für die Modellierung von Skripten wird noch eine restriktivere Form des Ereignisflusses definiert.

Knoten des Metatyps *MODELELEMENT* verbunden sein. Es existieren Knoten des Metatyps *MODELELEMENT*, die nicht durch *EVENTFLOW*-Beziehungsknoten mit anderen Knoten des Metatyps *MODELELEMENT* verbunden sind. Dies entspricht folgendem UML-Klassendiagramm:[1]

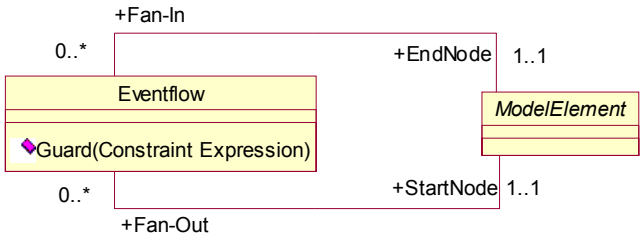


Abbildung 64: Beziehungsknotentypen des Metatyps EVENTFLOW im Konzeptuellen Schema

EVENTFLOW erhält ein Prädikat `Guard()`, durch welches eine Bedingung angegeben werden kann. Werden Knoten des Metatyps *INTERACTION* verbunden, beinhaltet `Guard()` als Term einen Ausdruck, dessen Ergebnis wahr sein muss, damit die folgende Interaktion stattfindet. Wird keine Bedingung angegeben, bedeutet dies eine sequentielle Abfolge. Die folgende Abbildung demonstriert die Abfolge von Modellelementen am Beispiel der Abfolge von Interaktionen:

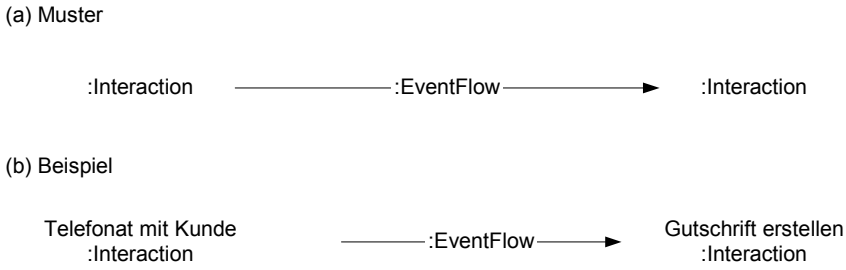


Abbildung 65: Muster und Beispiel eines Ereignisflusses

[1] In der UML existiert ein Metatyp *FLOW*, der formal ähnlich definiert ist. Dieser Metatyp wird aber nicht zur Modellierung der Abfolge von Ereignissen verwendet, sondern um die Abfolge unterschiedlicher Versionen des gleichen Objekts abzubilden. Vgl. [OMG 1999: S. 2-15, 2-32].

4.3.6.2 Anwendungsbereich

Der Anwendungsbereich ist eine Abhängigkeitsbeziehung, durch die die Verwendung eines Elements der Intension oder einer statischen Beziehung einer Klasse durch eine Rolle oder ein Objekt repräsentiert wird. Die Anwendungsbereich-Beziehung wird verwendet, um die Pseudo-Intension von Objekten und Rollen im Konzeptuellen Schema zu repräsentieren. Im Metamodell entspricht diese Beziehung einem Beziehungsknoten des Metatyps *SCOPE*. Die Beziehung geht von dem Knoten des Metatyps *ROLE* oder *OBJECT* aus und endet bei dem Knoten des Metatyps *MODELELEMENT*. Dabei werden zwei Unterklassen unterschieden:

- *SCOPEOFROLE* wird verwendet, um im Konzeptuellen Schema zu repräsentieren, welche Rolle ein Element der Intension (vgl. Abschnitt 4.2.2) oder eine statische Beziehung einer Klasse (genauer: mit einem Knoten des Typs *RELATIONSHIP* oder *INTENSION*) verwendet.
- *SCOPEOFOBJECT* wird verwendet, um im Konzeptuellen Schema zu repräsentieren, welches Objekt ein Element der Intension (vgl. Abschnitt 4.2.2) oder eine statische Beziehung einer Klasse (genauer: mit einem Knoten des Typs *RELATIONSHIP* oder *INTENSION*) verwendet.

Für die Beziehungsknotentypen *SCOPE* gilt: Jeder Knoten des Metatyps *ROLE* oder *OBJECT* muss entweder mit mindestens einem Knoten des Metatyps *RELATIONSHIP* oder mit mindestens einem Knoten des Metatyps *INTENSION* verbunden sein. Das bedeutet, dass eine Rolle oder ein Objekt mindestens in einer statischen Beziehung oder durch eine Intensionsbeziehung zur Anwendung gekommen sein muss. Das Objekt muss eine statische Beziehung oder Intensionsbeziehung der Klasse referenzieren, deren Exemplar es ist. Die Rolle muss eine statische Beziehung oder Intensionsbeziehung der Klasse referenzieren, deren Rolle sie übernimmt. Dies entspricht folgendem UML-Klassendiagramm:

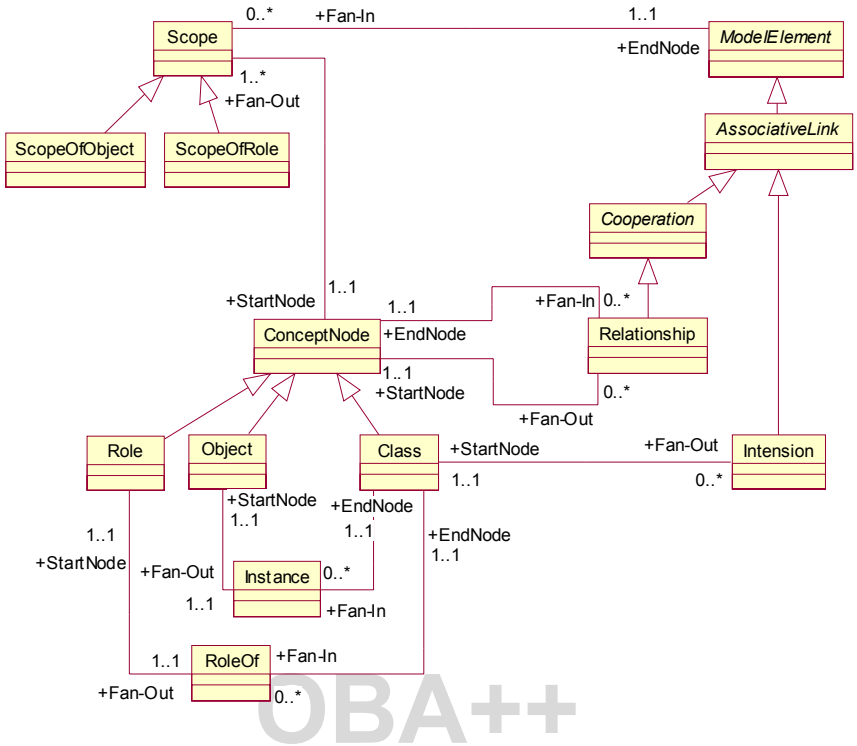


Abbildung 66: Beziehungsknotentypen zur Modellierung des Anwendungsbereichs

In der Grafik wurde aus Raumgründen die folgende OCL-Spezifikation weggelassen, die den Zusammenhang spezifiziert:[1]

```
context ScopeOfRole
inv:
(self.StartNode-> forAll(oclIsKindOf(Role)) and
self.EndNode-> forAll(oclIsKindOf(Relationship)))
-- Wenn der Ausgangspunkt der ScopeOfRole-Beziehung vom Metatyp Role
-- und der Endpunkt vom Metatyp Relationship ist, dann gilt:
implies
( (self.StartNode.Fan-Out.EndNode
-- Die Rolle ist über eine RoleOf-Beziehung
-- mit der gleichen Klasse verbunden ...
```

[1] Vgl. [OMG 1999]. Die Syntax zur Spezifikation von Beziehungen zwischen Subtypen folgt [Warmer 1999: S. 52].

```
= self.EndNode.EndNode->forAll(oclIsKindOf(Class)))
-- ... wie die referenzierte Klasse am Endpunkt der Relationship.
xor -- Oder:
(self.StartNode.Fan-Out.EndNode
-- Die Rolle ist über eine RoleOf-Beziehung
-- mit der gleichen Klasse verbunden ...
= self.EndNode.StartNode-> forAll(oclIsKindOf(Class))) )
-- ... wie die referenzierte Klasse am Ausgangspunkt der Relationship.

inv:
(self.StartNode-> forAll(oclIsKindOf(Role)) and
self.EndNode-> forAll(oclIsKindOf(Intension)))
-- Wenn der Ausgangspunkt der ScopeOfRole-Beziehung vom Metatyp Role
-- und der Endpunkt vom Metatyp Intension ist, dann gilt:
implies (( self.StartNode.Fan-Out.EndNode = self.EndNode.StartNode
-- Die Rolle ist über eine RoleOf-Beziehung
-- mit der gleichen Klasse verbunden wie
-- die referenzierte Klasse am Ausgangspunkt der Intensionsbeziehung.

context ScopeOfObject
inv:
(self.StartNode-> forAll(oclIsKindOf(Object)) and
self.EndNode-> forAll(oclIsKindOf(Relationship)))
-- Wenn der Ausgangspunkt der ScopeOfObject-Beziehung vom Metatyp Object
-- und der Endpunkt vom Metatyp Relationship ist, dann gilt:
implies
( (self.StartNode.Fan-Out.EndNode
-- Das Objekt ist über eine Instance-Beziehung
-- mit der gleichen Klasse verbunden ...
 = self.EndNode.EndNode-> forAll(oclIsKindOf(Class)))
-- ... wie die die referenzierte Klasse am Endpunkt der Relationship
xor -- Oder:
(self.StartNode.Fan-Out.EndNode
-- Das Objekt ist über eine Instance-Beziehung
-- mit der gleichen Klasse verbunden ...
= self.EndNode.StartNode-> forAll(oclIsKindOf(Class))) )
-- ... wie die die referenzierte Klasse am Ausgangspunkt der Relationship
```

```
inv:
(self.StartNode-> forAll(oclIsKindOf(Object)) and
self.EndNode-> forAll(oclIsKindOf(Intension)))
-- Wenn der Ausgangspunkt der ScopeOfObject-Beziehung vom Metatyp Object
-- und der Endpunkt vom Metatyp Intension ist, dann gilt:
implies self.StartNode.Fan-Out.EndNode = self.EndNode.StartNode
-- Das Objekt ist über eine Instance-Beziehung
-- mit der gleichen Klasse verbunden wie die
-- referenzierte Klasse am Ausgangspunkt der Intensionsbeziehung.
```

Die folgende Grafik demonstriert die Verwendung der beiden Beziehungsknotentypen:

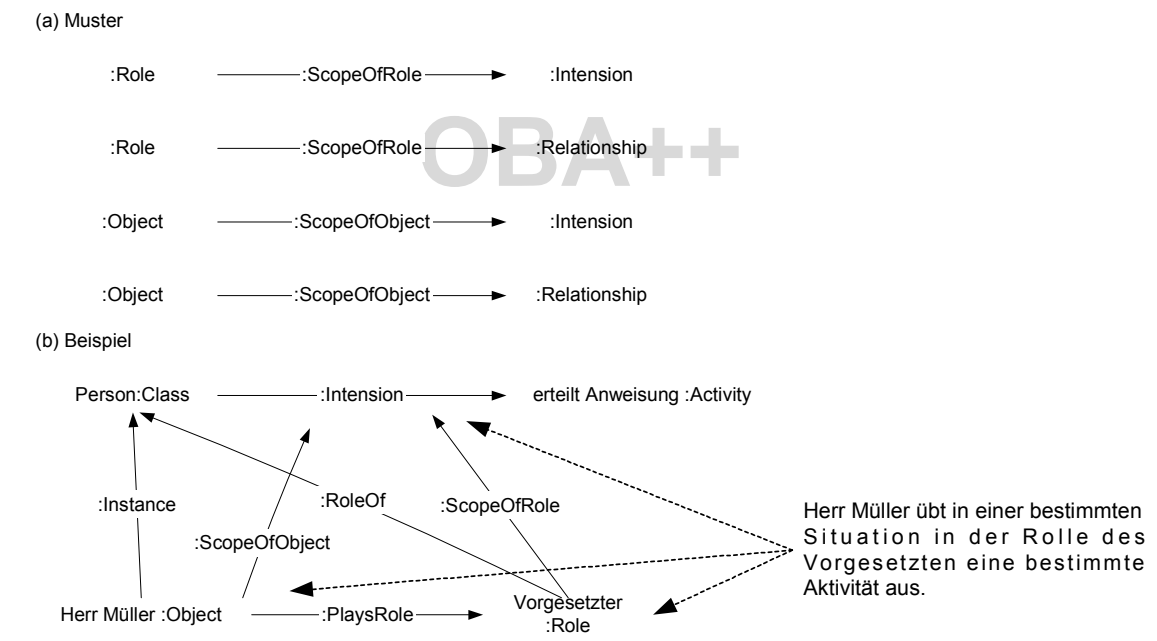


Abbildung 67: Muster und Beispiel der Anwendungsbereich-Abhängigkeit

Die *SCOPE*-Beziehung mit ihren beiden Untertypen existiert in dieser Form in keinem anderen Modellierungsansatz. Sie ist hier aber notwendig, um die Modellierung nicht nur auf der Ebene von Klassen, sondern auch auf der Ebene von Objekten und Klassen, die in einer bestimmten Rolle auftreten, zu ermöglichen.

4.3.6.3 Abstraktionsbeziehungen

Abstraktionsbeziehungen verbinden in der UML zwei Modellelemente, die den gleichen Umstand auf unterschiedlichen Abstraktionsebenen oder aus unterschiedlichen Gesichtspunkten dokumentieren.[1] Bei den Abstraktionsbeziehungen werden zwei Formen unterschieden:

- Eine *Realisierungsbeziehung*[2] liegt vor, wenn ein oder mehrere Modellelemente den Umstand eines anderen Modellelements in dem Sinne konkreter repräsentieren, dass bisher nicht dokumentierte Details dargestellt werden. Die Abstraktionsebene bleibt gleich und der Gesichtspunkt ändert sich.
- Eine *Verfeinerungsbeziehung* liegt vor, wenn die Repräsentation eines Umstands auf einem anderen Abstraktionsniveau durch ein oder mehrere andere Modellelemente repräsentiert wird. Die Abstraktionsebene wechselt und der Gesichtspunkt bleibt gleich.

Die beiden Beziehungsknotentypen lassen sich – bildhaft gesprochen – in der Form unterscheiden, dass die Realisierung dem Scharfstellen eines Bildes entspricht, die Verfeinerung einer Betrachtung mit einem kleineren Maßstab.[3] Für die Beziehungsknotentypen *ABSTRACTION* gilt: Ein Knoten des Metatyps *NODE* kann im Konzeptuellen Schema über Beziehungsknoten des Metatyps *ABSTRACTION* mit mehreren Knoten des Metatyps *NODE* verbunden sein. Die beiden Knoten dürfen aber nicht identisch sein. Dies entspricht folgendem UML-Klassendiagramm:

[1] Vgl. [OMG 1999: S. 2-18], [Rumbaugh 1999: S. 118].

[2] In der UML wird die Realisierungsbeziehung verwendet, wenn ein Modell die Implementierung eines anderen Modells darstellt. Vgl. [OMG 1999: S. 2-19], [Rumbaugh 1999: S. 405].

[3] Vgl. hierzu die ausführliche Darstellung von D'SOUZA und WILLS in [D'Souza 1999: S. 214 ff.]

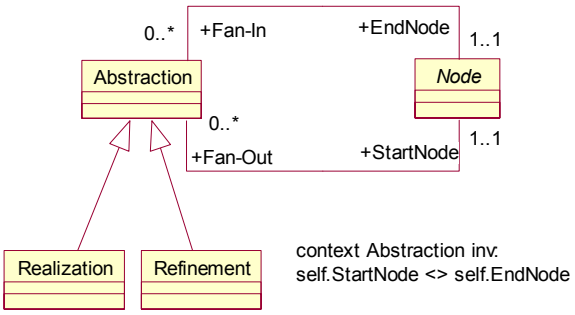


Abbildung 68: Beziehungsknotentypen zur Modellierung von Abstraktionsbeziehungen

Die Verschiedenheit ihrer Verwendung führt dazu, dass die Realisierungs- und Verfeinerungsbeziehung im Metamodell des Konzeptuellen Schemas recht allgemein definiert sind. Erst durch die Verwendung in externen Sichten wird die Verwendung präzisiert.[1]

4.3.6.4 Realisierung

Eine *Realisierungsbeziehung* ist beispielsweise die Beziehung zwischen der Operation eines Konzepts und den Interaktionen, Zustandsveränderungen und den Strukturen, die notwendig sind, um diese Operation zu realisieren: Im Rahmen der Ausführung der Operation einer Klasse finden eine Reihen von Interaktionen mit anderen Klassen statt. Realisierungsbeziehungen werden im Metamodell des Konzeptuellen Schemas durch den Beziehungsknotentyp *REALIZATION* repräsentiert.[2]

[1] Für die Modellierung von Skripten wird noch eine restriktivere Form der Abstraktionsbeziehungen definiert.

[2] In der UML werden Realisierungsbeziehungen beispielsweise verwendet, um Use Cases durch Kollaborationen zu realisieren (vgl. [Booch 1999: S. 101]). RUMBAUGH gibt ihr erstaunlicherweise – abweichend von allen anderen Abhängigkeitsbeziehungen – ein eigenes Symbol [Rumbaugh 1999: S. 118, 119, 406, 412]. BOOCH definiert davon abweichend die Realisierungsbeziehung als eigenständigen Beziehungstyp neben Abhängigkeit, Assoziation und Generalisierung (vgl. [Booch 1999: S. 23, 149]).

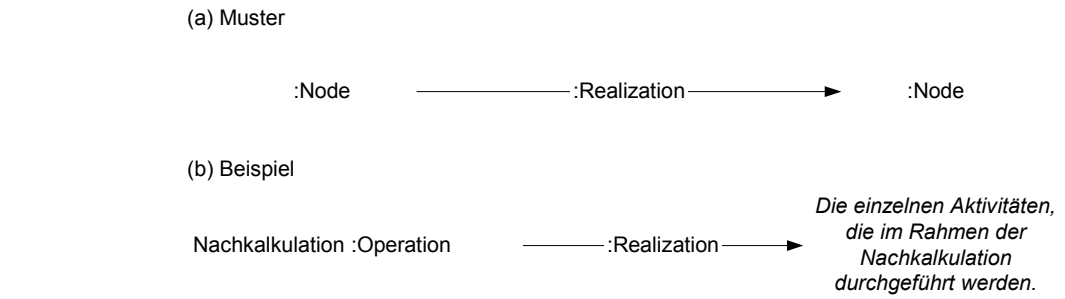


Abbildung 69: Muster und Beispiel einer Realisierungsbeziehung

4.3.6.5 Verfeinerung

Die *Verfeinerungsbeziehung* verbindet ein Modellelement mit einem anderen Modellelement, wobei letzteres eine andere Abstraktionsebene mit einem höheren Grad an Detailliertheit, Formalisierung oder Präzision besitzt.[1] Sie wird in der UML verwendet, um Modelle der Analyse mit Modellen des Entwurfs zu verbinden.[2] Sie verbindet hier zwei Knoten, von denen der Endknoten eine präzisere (im Sinne von aussagekräftiger, formaler, umfangreicher o. ä.) Beschreibung beinhaltet als der Ausgangsknoten.[3] Dies ist zum Beispiel dann der Fall, wenn die Aktivität eines Systems von einem Element des Systems ausgeführt wird. Verfeinerungsbeziehungen werden im Konzeptuellen Schema durch den Beziehungsknotentyp *REFINEMENT* repräsentiert:

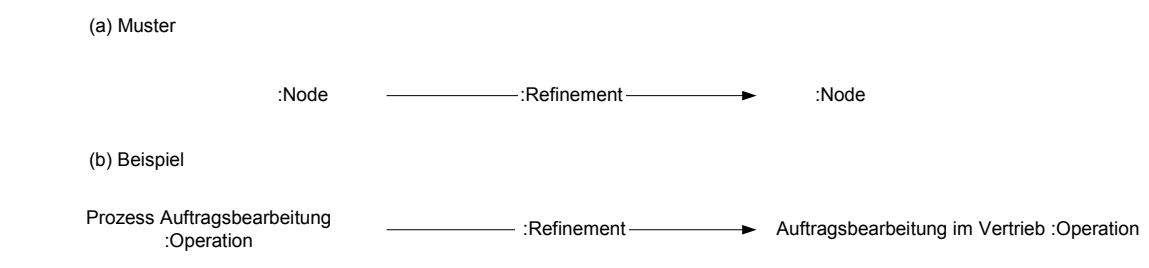


Abbildung 70: Muster und Beispiel einer Verfeinerungsbeziehung

[1] Vgl. [Rumbaugh 1999: S. 409], [OMG 1999: S. 2-19].

[2] Vgl. [Rumbaugh 1999: S. 410].

[3] Dies entspricht der Verwendung in der Methode CATALYSIS. [D'Souza 1998: S. 6-245].

4.3.6.6 Abhängigkeitsbeziehungen im Konzeptuellen Schema

Das folgende UML-Klassendiagramm fasst das Teilmodell der Abhängigkeitsbeziehungen im Konzeptuellen Schema zusammen.

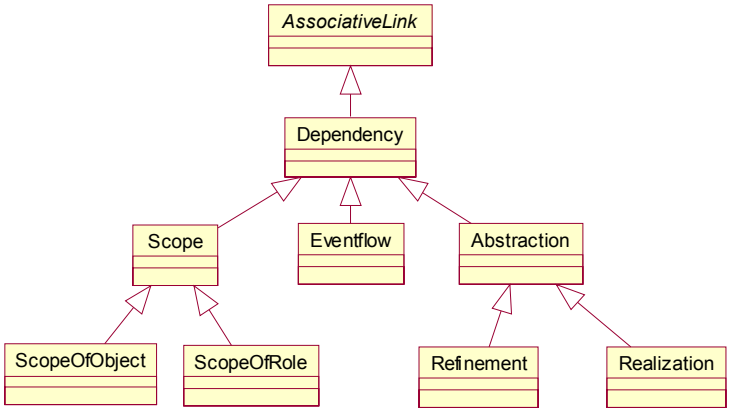


Abbildung 71: Abhängigkeitsbeziehungen im Konzeptuellen Schema

4.4 Struktur des Metamodells des Konzeptuellen Schemas

Ein Konzeptuelles Schema gilt als wohlgeformt, wenn es dem Metamodell des Konzeptuellen Schemas entspricht. In diesem Fall hat es folgende Eigenschaften:

- Es existieren keine Objekte, die nicht Exemplar einer Klasse sind.
- Es existieren keine isolierten Klassen.
- Es existieren keine isolierten Beziehungen.
- Es existieren keine Rollen, die nicht zu einer Klasse gehören.
- Es existieren keine Daten in Form von Attributen, die nicht zu einer Klasse gehören.
- Es existieren nur Aktivitäten, die einer Klasse zugeordnet sind.
- Die Interna einer Klasse können nur von der Klasse selbst verändert werden.
- Es existieren keine Intensionsknoten, die zu keiner Klasse gehören.
- Es existieren keine Intensionsknoten, die zu mehr als einer Klasse gehören.

Typisch für ein Metamodell ist sein fehlender Anwendungsbezug. Metamodelle sind auf einem hohen Abstraktionsgrad angesiedelt, um in unterschiedlichen Problembereichen Verwendung

finden zu können. Die Elemente sind als epistemische Primitive anzusehen, das heißt, durch sie werden Grundtatbestände ausgedrückt. Da das Konzeptuelle Schema auf einem domänenunabhängigen Kern basiert, kann es nicht nur zur GPM verwendet werden.[1]

4.4.1 Durch das Metamodell gebildete Abstraktionen

Durch das Metamodell des Konzeptuellen Schemas werden folgende Abstraktionen gebildet:

1. Alle Beziehungen, die zwischen Knoten geknüpft werden können, werden durch den abstrakten Metatyp *ASSOCIATIVELINK* und seine Spezialisierungen repräsentiert. Er ist die Wurzelklasse aller Beziehungsknotentypen, d. h. der Kanten des Semantischen Netzes. Von dieser abstrakten Metaklasse spezialisieren mehrere Klassen:
   1.1. Die von *ASSOCIATIVELINK* spezialisierte abstrakte Metaklasse der Kooperationsbeziehungen *COOPERATION* subsumiert alle dynamischen und statischen Beziehungen, die zwischen Konzepten bestehen.
      1.1.1. Die von *COOPERATION* spezialisierte Metaklasse *RELATIONSHIP* fasst alle statischen Beziehungstypen zwischen Konzepten zusammen.[2] Die Metaklasse *RELATIONSHIP* beinhaltet zwei Spezialformen:
         1.1.1.1. Die von *RELATIONSHIP* spezialisierte Metaklasse *TAXONOMIC-RELATIONSHIP* fasst alle klassifizierenden Beziehungstypen zwischen Konzepten zusammen.
         1.1.1.2. Die von *RELATIONSHIP* spezialisierte Metaklasse *REFERENTIAL-RELATIONSHIP* fasst alle referentiellen Beziehungstypen zwischen Konzepten zusammen.
      1.1.2. Die von *COOPERATION* spezialisierte Metaklasse *RESPONSIBILITYCOOPERATION* fasst alle dynamischen Beziehungstypen zwischen Konzepten zusammen.
   1.2. Die von *ASSOCIATIVELINK* spezialisierte Metaklasse *DEPENDENCY* repräsentiert die Abhängigkeitsbeziehungen im Konzeptuellen Schema. Diese Beziehungstypen verbinden auch Beziehungen miteinander.
   1.3. Die Intensionsbeziehungen (*INTENSION*) fassen alle Beziehungen zusammen, bei denen Knoten mit Intensionsknoten verbunden werden.

---

[1] Die Definition eines Branchen- oder Referenzmodells für Unternehmen, Organisationen oder Geschäftsprozesse wird hier nicht als Metamodell betrachtet, sondern als seine Anwendung, ggf. Anpassung für ein Anwendungsgebiet.

[2] Vgl. [Firesmith 1997: S. 367].

2. Alle Knoten werden durch den abstrakten Metatyp *NODE* und seine Spezialisierungen repräsentiert. Er ist die Wurzelklasse aller Knotentypen, d. h. der Knoten, die durch Kanten verbunden werden. Von dieser abstrakten Metaklasse spezialisieren mehrere Klassen:
   2.1. Die von *NODE* spezialisierte Metaklasse der Konzeptknoten *CONCEPTNODE* subsumiert die Klassen- und Individualkonzepte sowie die Rollen, die diese übernehmen können.
   2.2. Die von *NODE* spezialisierte Metaklasse der Intensionsknoten *INODE* subsumiert die Eigenschaften der Klassenkonzepte.
      2.2.1. Die von *INODE* spezialisierte Metaklasse der Aktivitäten *ACTIVITY* subsumiert alle Aktivitäten von Klassen.
         2.2.1.1. Die von *ACTIVITY* spezialisierte Metaklasse *OPERATION* umfasst alle Operationen, zu denen eine Klasse aufgefordert werden kann.
         2.2.1.2. Die von *ACTIVITY* spezialisierte Metaklasse *ACTION* umfasst alle Aktionen, die eine Klasse ausführt.
      2.2.2. Die von *INODE* spezialisierte Metaklasse der Attribute *ATTRIBUTE* subsumiert alle Attribute von Klassen.

Das Metamodell des Konzeptuellen Schemas bildet das UML-Paket CM.[1] Fügt man die bisher definierten Teilmodelle zusammen, ergibt sich für das Konzeptuelle Schema die in der folgenden Abbildung 72 dargestellte Vererbungsstruktur. Die zwischen den Metatypen definierten Assoziationen fehlen aus Gründen der Übersichtlichkeit. Die im weiteren Verlauf der Arbeit verwendeten Prädikate (Operationen der Metatypen) sind tabellarisch in Tabelle 7 und als UML-Klassen im Anhang (Anhang 3: Die Schnittstellen der Klassen des Metamodells) dargestellt.

[1] *Conceptual Model*.

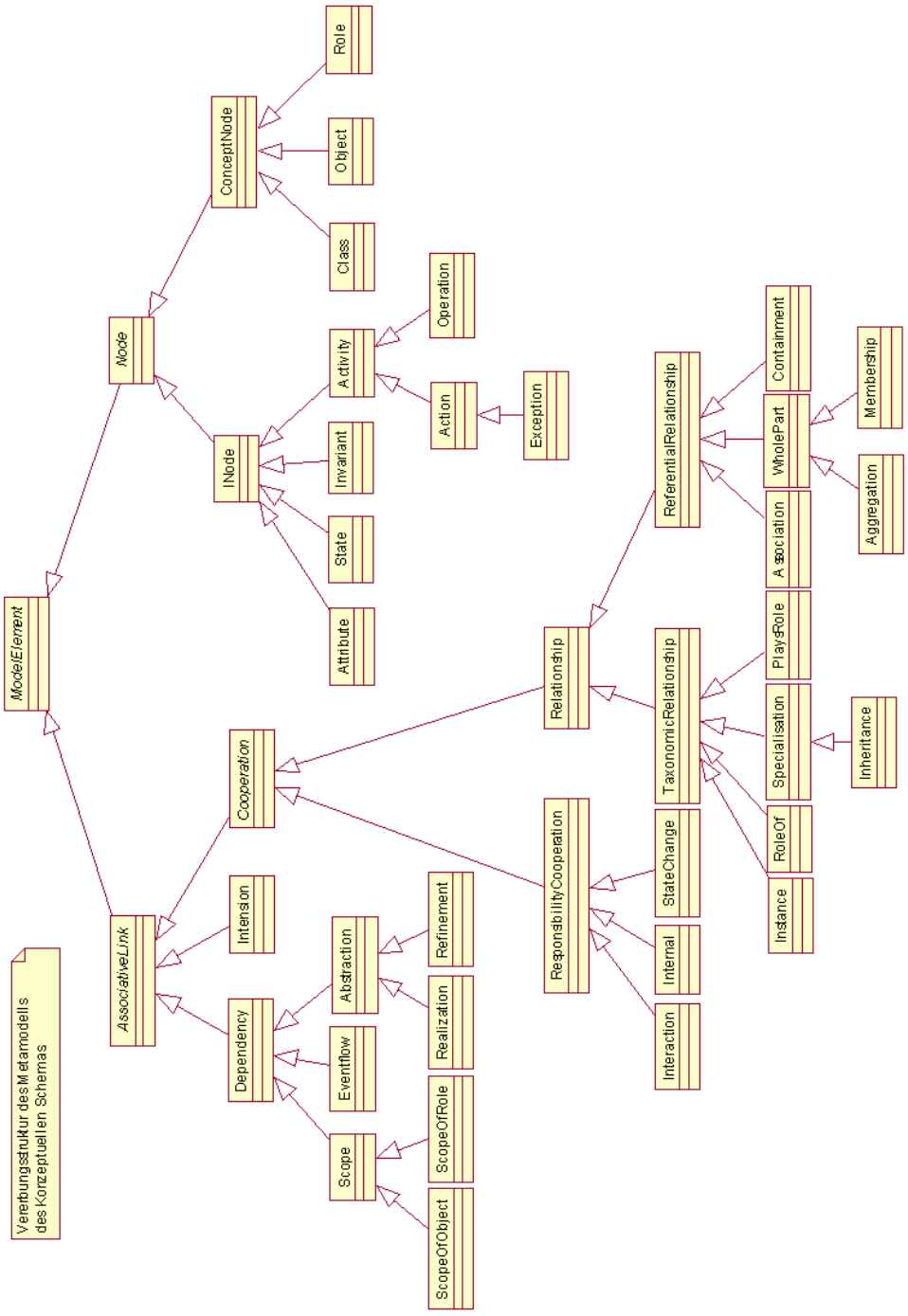


Abbildung 72: Die Vererbungsstruktur des Metamodells des Konzeptuellen Schemas: Aufbau des UML-Pakets CM

Die Vererbungsstruktur bietet folgende Vorteile:

- Sie lässt sich erweitern. Es ist möglich, zusätzliche Metatypen zu definieren, um beispielsweise andere Arten von Beziehungen im Konzeptuellen Schema abzubilden.
- Die vorhandenen Knotentypen und Beziehungsknotentypen lassen sich weiter differenzieren. Über Vererbung lassen sich die bestehenden Knotentypen durch speziellere präzisieren.
- Das Metamodell ist durch die Nutzung der Vererbungsbeziehung kompakt. Die für übergeordnete Typen definierten Eigenschaften werden im Rahmen der Vererbung an alle spezialisierten Typen weitergegeben. Alle Metatypen spezialisieren die für den Metatyp *MODELELEMENT* definierten Eigenschaften. Wenn definiert wird, dass Klassenknoten durch Knoten des Typs *INTENSION* mit Knoten des Typs *INODE* verbunden werden, gilt dies auch für alle abgeleiteten Metatypen, also *OPERATION*, *ATTRIBUTE*, *INVARIANT* usw.

Beim Aufbau dieses Metamodells wurde die verbreitete Modellierungskonvention angewendet, dass abstrakte Klassen nur von abstrakten Klassen erben sollten. Die UML erzwingt dies nicht. Der Metatyp *SCOPE* wird beispielsweise deshalb nicht abstrakt modelliert, weil der Metatyp DEPENDENCY nicht abstrakt ist. Andere Metatypen (z. B. *DEPENDENCY*) sind aus Gründen der Kompatibilität zur UML nicht abstrakt. Schließlich sind einige Metatypen aus dem pragmatischen Grund nicht abstrakt, um möglichst große Freiheiten bei der Definition anderer Externer Schemas zu haben.

4.4.2 Namensräume im Metamodell

Ein Namensraum ist ein bestimmter Bereich, innerhalb dessen ein Bezeichner eindeutig sein muss. Da im Metamodell des Konzeptuellen Schemas Modellelemente teilweise über ihren Namen identifiziert werden, ist die Definition von Namensräumen notwendig, um Modellelemente eindeutig referenzieren zu können.

Bei einem wohlgeformten Konzeptuellen Schema gilt, dass innerhalb eines Namenraums nicht zwei Modellelemente gleichen Metatyps mit dem gleichen Namen existieren können. Für zwei unterschiedliche Namensräume gilt, dass in jedem Namensraum ein Modellelement existieren kann, dessen Name und Metatyp mit einem Modellelement in einem anderen Namensraum übereinstimmt.

In der objektorientierten Modellierung gelten implizit unterschiedliche Namensräume, die hier für das Metamodell des Konzeptuellen Schemas übernommen wurden und explizit festgelegt werden.

Der Namensraum für Exemplare des Metatyps *CLASS* ist ein Konzeptuelles Schema, das heißt, der Name einer Klasse muss im Konzeptuellen Schema eindeutig sein. Der Namensraum für Exemplare der Metatypen *OBJECT* und *ROLE* ist *CLASS*. Das bedeutet, Exemplare unterschiedlicher Klassen können den gleichen Namen besitzen oder die gleiche Rolle einnehmen. Wenn dies nicht der Fall wäre, wäre die Aussage *Busse und Bahnen übernehmen die Rolle von Fortbewegungsmitteln* nicht abzubilden. Der Namensraum für Exemplare der Metatypen *INODE* (resp. der definierten Subtypen) ist ebenfalls *CLASS*. Das bedeutet, dass unterschiedliche Klassen z. B. Attribute und Operationen mit dem gleichen Namen besitzen können. Wäre dies nicht der Fall, könnte kein Polymorphismus realisiert werden, durch den zwei Klassen Operationen mit gleichem Namen, aber unterschiedlicher Semantik besitzen.[1]

Beziehungsknotentypen besitzen keinen Namensraum. Entweder sie erhalten keinen Namen (wie beispielsweise Intensions- oder Instanzierungsknoten), oder sie sind – unabhängig von ihrem Namen – durch ihre Objektidentität eindeutig. Das bedeutet, dass beispielsweise beliebig viele Interaktionsknoten mit einem Namen X existieren können.

4.4.3 Erweiterung des Metamodells

Insgesamt stehen sechs Möglichkeiten der Erweiterung zur Verfügung:

1. Zusätzliche Terme für vorhandene Prädikate.
2. Definition von Stereotypen für Knoten und Kanten.
3. Zusätzliche Prädikate.
4. Definition neuer Beziehungsknotentypen.
5. Definition neuer Intensionsknotentypen.
6. Neue Knotentypen.

Die Nummerierung gibt in zweierlei Hinsicht eine Reihenfolge vor. Zunächst stellt sie die Reihenfolge dar, mit der das Metamodell an eine Problemstellung angepasst werden sollte. Außerdem stellt diese Reihenfolge auch eine Steigerung des Umfangs des Eingriffs dar.

- Zu 1: Zu vielen Prädikaten werden die möglichen Terme vordefiniert. Zu diesen können weitere Werte hinzugefügt werden, was keine Veränderung des Metamodells bedeutet.

[1] Das in vielen Programmiersprachen verwendete Konzept überladener Operationen [Budd 1997: S. 255] müsste in der Form realisiert werden, dass im Konzeptuellen Modell die Namen der Operationen so umgeformt werden, dass sie eindeutig sind. Compiler verwenden ein ähnliches Verfahren.

- Zu 2: Je nach Anwendungsbereich können die Knoten des Semantischen Netzes einen Stereotyp erhalten. Auf diese Weise werden problemspezifische Typen definiert, ohne das Metamodell verändern zu müssen. Dies ist die Ebene, auf der in der UML problemspezifische Erweiterungen des Metamodells vorgenommen werden sollen.[1] Es wäre zum Beispiel möglich, einen Stereotyp *Organisationseinheit* zu definieren, um zu verdeutlichen, dass es sich bei den Knoten um Elemente der Aufbauorganisation handelt.[2]
- Zu 3: Sollen Knotentypen mit zusätzlichen Eigenschaften versehen werden, ist die Definition weiterer Prädikate notwendig. Dies stellt eine Veränderung des bestehenden Metamodells dar. Beispielsweise könnte ein Prädikat für im Rahmen von Kommunikationsprozessen auftretenden Informationsverlust[3] eingeführt werden.
- Zu 4: Die Definition neuer Beziehungsknotentypen ist dann notwendig, wenn zusätzliche Modellierungskonstrukte eingeführt werden sollen, die sich nicht durch vordefinierte Terme, Prädikate oder Stereotype abbilden lassen. Dies kann z. B. der Fall sein, wenn man die unterschiedlichen Abhängigkeitsbeziehungen der UML oder die differenzierten Vererbungsbeziehungen der OML oder die von MEYER[4] übernehmen möchte. Neue Beziehungstypen sollten aber als Unterklassen vorhandener Beziehungstypen definiert werden.
- Zu 5: Neue Intensionsknotentypen wird man beispielsweise dann definieren, wenn zusätzliche Eigenschaftstypen benötigt werden. Diese stellen dann Spezialisierungen von *INODE* dar.[5]
- Zu 6: Neue Knotentypen wird man dann definieren, wenn es beispielsweise erforderlich ist, die Knotentypen der UML zu übernehmen. Es ist dann im Einzelfall zu prüfen, ob diese als Spezialisierungen von *NODE* oder *CONCEPTNODE* definiert werden. [6]

---

[1] Dies wird auch als *leichtgewichtige Metamodellierung* bezeichnet. Vgl. [Atkinson 2000: S. 34].

[2] Im weiteren Verlauf wird hierauf noch eingegangen.

[3] Vgl. [Steinbuch 1974: S: 92 f.], [Rechenberg 1999: S. 192, 198].

[4] Vgl. [Meyer 1997: S. 822 ff.].

[5] Weitere Möglichkeiten zur Beschreibung der Intension, die sich im Bereich objektorientierter formaler Spezifikationssprachen wie TROLL [Saake 1993], TAOS [Schienmann 1997] oder VDM++ [IACS 1996] finden lassen, wurden nicht verwendet. Sie können für solche Erweiterungen verwendet werden.

[6] Vgl. [OMG 2001: S. 2-17 ff.]. Dies sind die Metatypen *Interface*, *Node*, *DataType*, *Component* und *Artifact*. Artefakte sind – nach der Definition der UML – physische Informationsstücke. Komponenten sind austauschbare Teile eines Systems. Datentypen sind primitive Datentypen ohne eigene Identität. Knoten sind physische Objekte, die über einen Arbeitsspeicher verfügen und Berechnungen ausführen können. Eine Schnittstelle ist eine Menge von Operationen, die das Verhalten eines Elements beschreibt. Dem Metamodell der UML ist zu entnehmen, dass diese Knotentypen als Spezialisierung des Metatyps *CONCEPTNODE* zu definieren wären. Vgl. [OMG 2001: S. 2-17, Abb. 2-8].

4.4.4 Metamodell des Konzeptuellen Schemas im Vergleich zur UML

Ein wesentlicher Unterschied zwischen dem hier vorgestellten Metamodell und dem der UML besteht darin, dass nicht zwischen statischer und dynamischer Sicht unterschieden wird.

Die dynamische Sicht – wobei hier in erster Linie die Interaktionen zwischen Objekten gemeint sind – wird in der Weise integriert, dass die Objekte über *INSTANCEOF*-Beziehungsknoten mit den Klassen verbunden werden. Auf diese Weise wird durch die Zuordnung von Objekten zu Klassen eine Abbildung der dynamischen auf die statische Sicht durchgeführt. Durch die Beziehungsknotentypen *ROLEOF*, *PLAYSROLE* und *SCOPE* kann über das Konzeptuelle Schema außerdem abgebildet werden, welches Objekt in welcher Rolle eine Aktivität ausgeführt hat oder eine Beziehung eingegangen ist.

Im Gegensatz zu anderen Metamodellen ist es durch diese Verbindung von dynamischer und statischer Sicht nicht nur möglich, abzubilden, welche Intension eine Klasse besitzt und in welchen Beziehungen sie zu anderen Klassen steht, sondern auch, ob und in welchem Zusammenhang die Eigenschaften und Beziehungen verwendet bzw. benötigt werden. Alle durch Objekte und Rollen verwendeten Eigenschaften und Beziehungen werden als Eigenschaften und Beziehungen der jeweiligen Klassen repräsentiert. Zudem kann für jede Eigenschaft und Beziehung einer Klasse festgestellt werden, durch welche Objekte oder in welchen Rollen diese Eigenschaft oder Beziehung verwendet wird.

4.5 Zusammenfassung

Gegenstand dieses Kapitels ist das Metamodell des Konzeptuellen Schemas. Dieses Metamodell bildet das Begriffssystem der im vorherigen Kapitel eingeführten Metapher. Es kann als erweiterbares Fundament für unterschiedliche Externe Schemata verwendet werden. Das im nächsten Kapitel entwickelte Externe Schema der Skripte ist eine Sicht auf dieses Konzeptuelle Schema und stellt die externe Repräsentation des Modellierungsansatzes dar.

5 Der Aufbau des Skriptmodells der OBA++

„I do not see the object-oriented method as a mere fad;
I think it is not trivial [...].“
[Meyer 1997: S. v]

Das Ziel dieses Kapitels ist es, den Aufbau der externen Repräsentationsform Skript und die Abbildung von Skripten auf das Konzeptuelle Schema zu entwickeln.

5.1 Aufbau der Skripte

Als externe Repräsentation eines Skripts wird eine Tabelle mit sechs Spalten verwendet. Die Anordnung der Spalten ist festgelegt. Die Spalten müssen nicht beschriftet sein.[1] Dies entspricht folgendem Aufbau:

Number	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.

Skript 1: Äußere Form eines Skripts

Die Bezeichnungen der Tabellenspalten Initiator und Participant wurden aus der OBA übernommen. Die Tabellenspalte Initiator Responsibility entspricht in der OBA der Spalte Action, die Tabellenspalte Participant Responsibility entspricht der Tabellenspalte Service.[1] Zwei Tabellenspalten sind hinzugekommen: Die neu eingeführte Tabellenspalte Connection wird dazu verwendet, um die Art der Beziehung, ihren Inhalt usw. abbilden zu können. Die neu eingeführte Tabellenspalte Number wird dazu verwendet, um zwischen den Zeilen Anordnungsbeziehungen festlegen zu können. Darauf wird später noch eingegangen.

[1] Im weiteren Verlauf der Arbeit verbleibt die Tabellenspalte Number aus typographischen Gründen in der Regel unbeschriftet.

Ein Skript besteht aus keiner, einer oder mehreren Zeilen. Jede Zeile gehört zu genau einem Skript. Die Zeilen besitzen innerhalb des Skripts eine Reihenfolge. Jede Zeile besteht aus sechs Spalten mit den Namen *Initiator*, *Initiator Responsibility*, *Connection*, *Number*, *Participant* und *Participant Responsibility*. Jede Tabellenspalte hat dabei eine bestimmte Bedeutung. Die einzelnen Zeilen des Skript werden in unterschiedlicher Art und Weise ausgefüllt. Diese unterschiedlichen Formen werden verwendet, um unterschiedliche Sachverhalte auszudrücken. Der logische Aufbau entspricht folgendem UML-Klassendiagramm:

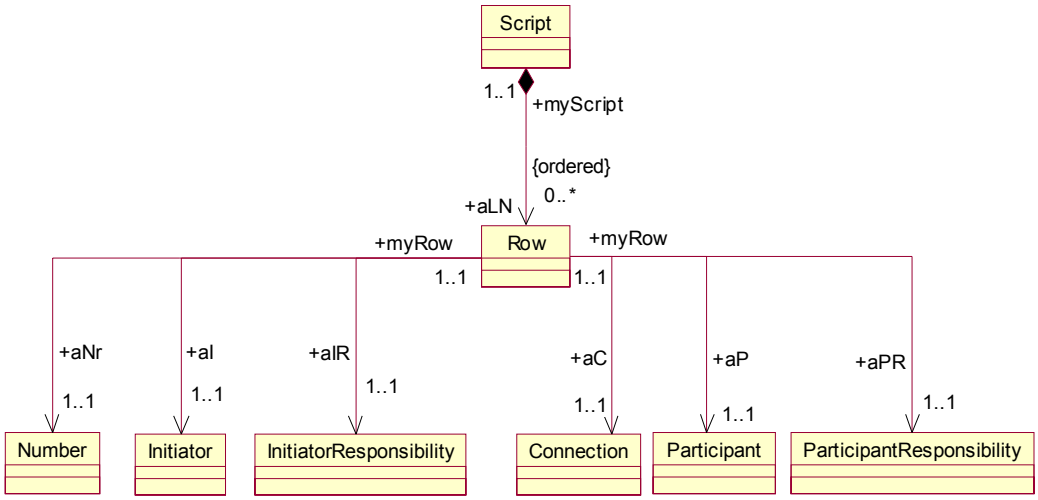


Abbildung 73: Aufbau der Skripte als UML-Klassendiagramm

Skripte werden durch den Metatyp *SCRIPT* repräsentiert, der in einer UML-Aggregationsbeziehung zum Metatyp *ROW* steht. Der Metatyp *ROW* repräsentiert jeweils eine Zeile des Skripts. An der UML-Aggregationsbeziehung zwischen *SCRIPT* und *ROW* nehmen 0 bis n Exemplare von *ROW* und genau ein Exemplar von *SCRIPT* teil. Jedes *ROW*-Exemplar steht zu exakt einem Exemplar der Metatypen *NUMBER*, *INITIATOR*, *INITIATOR RESPONSIBILITY*, *CONNECTION*, *PARTICIPANT* und *PARTICIPANT RESPONSIBILITY* in einer UML-Assoziationsbeziehung. Das sind die jeweiligen Spalten des Skripts. Das Löschen eines *SKRIPT*-Exemplars zieht das kaskadierende Löschen der beteiligten *ROW*-Exemplare und der zugehörigen Exemplare der Klassen *INITIATOR*, *INITIATOR RESPONSIBILITY* usw. nach sich. Das bedeutet, dass alle Zeilen eines Skripts gelöscht werden, wenn das Skript gelöscht wird. Die Exemplare dieser Metatypen

[1] Vgl. [Rubin 1993].

gehören zu exakt einem *ROW*-Exemplar. Das bedeutet, dass eine Zelle einer Tabelle niemals gleichzeitig zu einer weiteren Zeile oder Tabelle gehören kann.

Der Aufbau der einzelnen Tabellenspalten eines Skripts wird im weiteren Verlauf durch eine Grammatik in Form einer erweiterten BACKUS-NAUR-Form (eBNF) festgelegt. Diese beginnt mit folgender Produktionsregel:[1]

```
<Skriptzeile>::=   <Inhalt Spalte Number> +
                  <Inhalt Spalte Initiator> +
                  <Inhalt Spalte Initiator Responsibility> +
                  <Inhalt Spalte Connection> +
                  <Inhalt Spalte Participant> +
                  <Inhalt Spalte Participant Responsibility>;
```

In den Produktionsregeln werden folgende Symbole verwendet:

`::=`	Ist äquivalent zu
`[a | b]`	Auswahl (entweder a oder b)
`+`	Sequenz
`Ø`	Leer
`* *`	Kommentar
`„ “`	Terminalsymbol
`< >`	Nicht-Terminalsymbol
`M{ }N`	Wiederholung (von M bis N)
`( )`	Option, gleichwertig zu 0{ }1
`;`	Ende einer Produktionsregel
`.`	Ende der Produktionsregeln

5.2 Skriptmodell als Externes Schema

Das Skriptmodell ist ein Externes Schema, welches eine bestimmte Sicht auf das Konzeptuelle Schema darstellt. Das Metamodell des Skriptmodells ist in der Notation der UML ein Paket mit dem Namen SKRM, welches das Metamodell des Konzeptuellen Schemas im Paket CM verwendet:

[1] Vgl. [Kopp 1988: S. 6 ff.], [Aho 1992: S. 32 ff.].

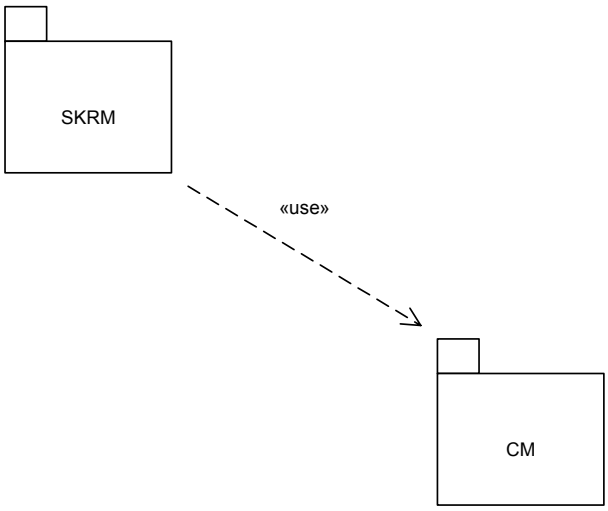


Abbildung 74: Beziehung zwischen dem Metamodell des Konzeptuellen und des Externen Schemas

Diese Konzeption wird aus folgenden Gründen verwendet:

- Bei der Definition Externer Schemata kann das Konzeptuelle Schema wiederverwendet werden.
- Es können beliebige weitere Externe Schemata definiert werden, solange sie sich auf das Konzeptuelle Schema abbilden lassen.
- Externe Schemata können voneinander isoliert definiert werden.

Bisher wurden im Metamodell des Konzeptuellen Schemas Knoten (*NODE*) und Kanten bzw. Beziehungsknoten (*ASSOCIATIVELINK*) unterschieden, die die Gesamtheit der Modellelemente (*MODELELEMENT*) ausmachen. Darüber hinaus wurden bei den Knoten Konzeptknoten (*CONCEPTNODE*) und ihre beschreibenden Eigenschaften (*INODE*) unterschieden. Es stellt sich nun die Frage, wie die Metatypen des Skriptmodells (also die Metaklassen *SCRIPT*, *ROW* usw. des Pakets SKRM[1]) einzuordnen sind.

Skripte stellen eine bestimmte Sicht auf das Konzeptuelle Schema dar. Sie besitzen eine gleichförmige Struktur, zwischen ihnen bestehen Beziehungen, und sie besitzen einen Namen, durch den sie identifiziert werden können. Auf dieser Grundlage ließe sich der Metatyp *SCRIPT* als eine spezielle Form von Klassen einordnen und vom Metatyp *CLASS* vererben. Da von Skripten

[1] SKRiptModell

aber keine Exemplare erzeugt werden und sie keine Rollen übernehmen, wird eine andere Struktur gewählt:

Skripte sind Knoten, die durch Beziehungen mit anderen Knoten verbunden werden. Außerdem stellen sie Konzepte dar. Darüber hinaus sind sie nur eine bestimmte Sicht auf das Konzeptuelle Schema, neben der noch weitere existieren könnten. Sie sollen aber weder als Klassen noch als Rollen oder Objekte abgebildet werden. Aus diesem Grund erbt der Metatyp *SCRIPT* von einem neu eingeführten Metatyp *VIEW*, der die Wurzelklasse aller Externen Sichten darstellen soll. Er erbt vom Metatyp *CONCEPTNODE* des Konzeptuellen Schemas. Von *VIEW* erben darüber hinaus alle Modellelemente einer Externen Sicht. In diesem Fall sind dies die Klassen *ROW*, *INITIATOR*, *PARTICIPANT* usw. Das folgende Diagramm verbindet die Vererbungsstruktur des Metamodells im Paket SKRM mit dem Metamodell des Konzeptuellen Schemas im Paket CM.

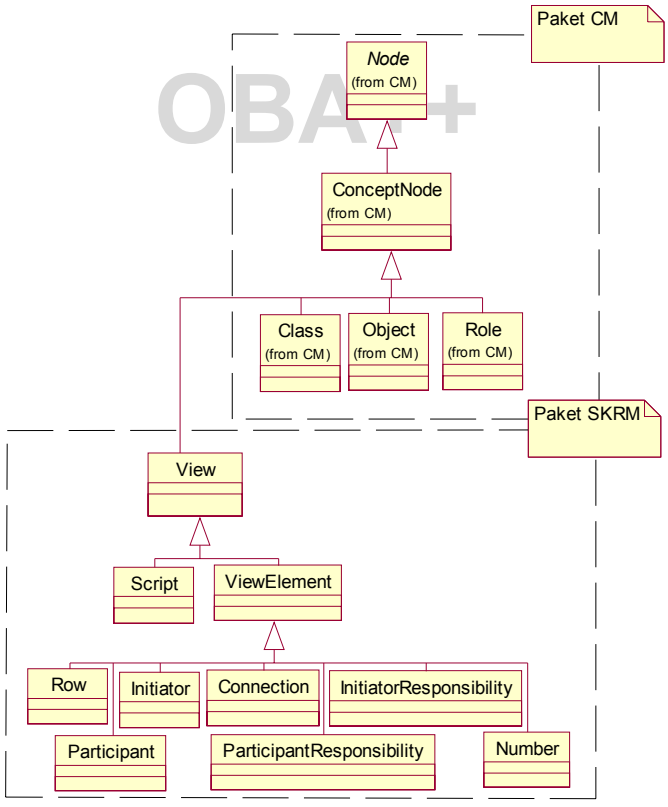


Abbildung 75: Einordnung der Modelle im Metamodell des Konzeptuellen Schemas als UML-Klassendiagramm

5.2.1 Die Tabellenspalten Initiator und Participant

5.2.1.1 Aufbau der Tabellenspalten

Die Tabellenspalten Initiator und Participant beinhalten die Namen einer Klasse, eines Objekts und einer Rolle. Objekt- und Rollennamen können nur angegeben werden, wenn auch ein Klassenname angegeben wurde. Wenn die Tabellenspalten nicht leer sind, muss ein Klassenname angegeben werden. Wird ein Klassenname angegeben, kann genau ein Objektname und bzw. oder genau ein Rollenname angegeben werden. Der Klassenname wird mit dem Zeichen „:“ eingeleitet, der Rollenname wird mit dem Zeichen „/“ eingeleitet. Eine Zeichenkette ohne das Präfix „:“ oder „/“ wird als Klassenname angesehen. Optional kann eine Spezifikation angegeben werden, die aus einem oder mehreren Prädikaten besteht. Der Aufbau der Tabellenspalten ist durch folgende Produktionsregeln festgelegt:[1]

```
<Inhalt Spalte Initiator> ::= [ <Initiator> | Ø ];
<Inhalt Spalte Participant> ::= [ <Participant> | Ø ];
<Initiator> ::= [(<Objektbez>)+(<Rollenbez>)+<Klassenbez>+
             (<Spezifikation>) | <Klassenname>+(<Spezifikation>)] ;
<Participant>:::=[(<Objektbez>)+(<Rollenbez>)+<Klassenbez>+
             (<Spezifikation>)| <Klassenname>+(<Spezifikation>)];
<Objektbez> ::= <Objektname>;
<Rollenbez> ::= „/”+ <Rollenname>;
<Klassenbez> ::= „:” + <Klassenname>;
<Spezifikation> ::= 1{<Prädikat>}M;
*<Prädikat> ::= Siehe Abschnitt 5.2.6;*
*<Rollenname> ::= Name eines Knotens der Metaklasse ROLE im
Konzeptuellen Schema;*
*<Objektname> ::= Name eines Knotens der Metaklasse OBJECT im
                Konzeptuellen Schema;*
*<Klassenname> ::= Name eines Knotens der Metaklasse CLASS im
                Konzeptuellen Schema;*
```

Das Metamodell der Tabellenspalten Initiator und Participant entspricht folgendem UML-Klassendiagramm. Dabei wurden die Prädikate der Spezifikation ausgeblendet, weil sie im Moment noch nicht von Interesse sind:

[1] Die Produktionsregeln sollen der Erläuterung dienen. Aus diesem Grund sind sie nicht so detailliert formuliert worden, dass sie für einen Generator verwendet werden könnten.

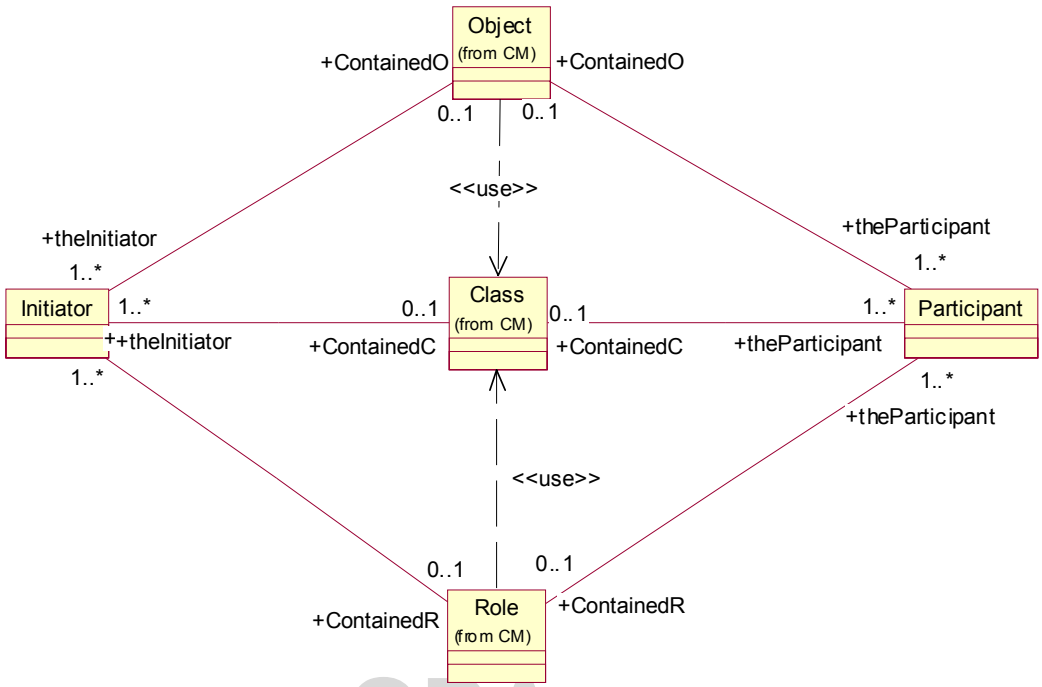


Abbildung 76: Aufbau der Tabellenspalte Initiator und Participant als UML-Klassendiagramm

Jede Rolle, jedes Objekt oder jede Klasse kann mehrfach in unterschiedlichen Zeilen eines oder mehrerer Skripte als Initiator oder Participant verwendet werden. Die Spalten Initiator und Participant (abgebildet durch die Metatypen *INITIATOR* und *PARTICIPANT*) können je eine Klassenbezeichnung, einen Objektnamen und eine Rolle beinhalten, die im Konzeptuellen Schema als ein Knoten der Metatypen *ROLE*, *OBJECT* und *CLASS* repräsentiert werden. Bei bestimmten Beziehungen kann diese Tabellenspalte leer sein (vgl. Abschnitt 5.2.3.6). Aus diesem Grund ist die Beziehung zwischen *PARTICIPANT* und *INITIATOR* auf der einen Seite und *ROLE*, *OBJECT* und *CLASS* auf der anderen Seite optional. Durch die beiden UML-Dependency-Beziehungen mit dem Stereotyp «use» wird festgehalten, dass in den Tabellenspalten Initiator und Participant nur dann ein Objekt oder eine Rolle angegeben werden kann, wenn auch eine Klasse angegeben wurde: Die Verwendung eines Objekt- oder Rollenknotens ist von der Existenz eines Klassenknotens abhängig. Wenn die Spalte ausgefüllt ist, muss ein Klassenname angegeben werden.

In der tabellarischen Darstellung der Skripte entspricht dies – unter Verwendung der definierten eBNF – folgendem Aufbau:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	[(<Objektname>) + („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]	...	...	[(<Objektname>) + („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]	...

Skript 2: Muster für die Tabellenspalten Initiator und Participant

5.2.1.2 Abbildung von Initiator und Participant im Konzeptuellen Schema

Der Inhalt der Tabellenspalten Initiator und Participant wird in identischer Weise im Konzeptuellen Schema abgebildet. Der dort vorgefundene Inhalt wird zerlegt, wobei die Zeichen „/“ und „:“ als Trennsymbole für die sich jeweils anschließenden Textketten verwendet werden. Die mit dem Doppelpunkt beginnende Textkette wird als Klassenbezeichner verwendet, die mit dem Schrägstrich beginnende Textkette wird als Rollenbezeichner verwendet. Die Textkette, die von keinem Trennsymbol eingeleitet wird, wird als Objektbezeichner verwendet, wenn außerdem eine weitere Textkette vorhanden ist, die mit einem „:“ beginnt. Ist in der Tabellenspalte Initiator oder Participant eine Textkette enthalten, die von keinem der genannten Trennsymbole eingeleitet wird, wird sie als Klassenname angesehen.

Ist die Textkette zerlegt, findet die Abbildung auf das Konzeptuelle Schema wie folgt statt: Ein Knoten des Metatyps *CLASS* wird im Konzeptuellen Schema mit dem verwendeten Klassenbezeichner als Namen angelegt, wenn er nicht schon existiert. Der Knoten des Metatyps *OBJECT* und dem verwendeten Objektnamen wird im Konzeptuellen Schema angelegt, wenn nicht schon ein Knoten dieses Metatyps mit diesem Objektnamen existiert, welcher über eine *INSTANCE*-Beziehung mit dem Klassenknoten verbunden ist. Ein solcher neu angelegter Objektknoten wird dann über einen Beziehungsknoten des Metatyps *INSTANCE* mit dem Klassenknoten verbunden. Ein Knoten des Metatyps *ROLE* wird angelegt, wenn im Konzeptuellen Schema nicht schon ein Knoten dieses Metatyps mit diesem Rollennamen existiert, der über einen *ROLEOF*-Beziehungsknoten mit dem Klassenknoten verbunden ist. Der neu erzeugte Rollenknoten wird dann über einen Beziehungsknoten des Metatyps *ROLEOF* mit dem Klassenknoten verbunden. Als Ergebnis existiert folgendes Semantische Netz im Konzeptuellen Schema:

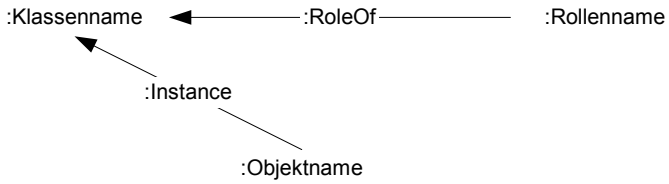


Abbildung 77: Repräsentation von Initiator und Participant im Konzeptuellen Schema

Da Objekte immer die Extension einer Klasse sind, ist für das Erzeugen von Rollen- und Objektknoten die Existenz des Klassenknotens notwendig. Aus diesem Grund ist es nicht möglich, Objekte oder Rollen ohne Klassenbezug anzugeben.

5.2.2 Die Tabellenspalte Connection

Die in den Skripten in der Tabellenspalte Connection verwendeten Beziehungstypen entsprechen Beziehungsknotentypen des Konzeptuellen Schemas. Folgende Beziehungstypen werden verwendet:

Im Skript abgebildeter Umstand:	**Verwendeter Beziehungs-typ im Skript:**	**Beziehungsknotentyp im Metamodell des Konzeptuellen Schemas:**
Interaktion zwischen Initiator und Participant	Interaction	INTERACTION
Referentielle Beziehung zwischen Initiator und Participant	Association, Membership, Whole-Part, Containment, Aggregation	ASSOCIATION, MEMBERSHIP, WHOLE-PART, CONTAINMENT, AGGREGATION
taxonomische Beziehung zwischen Initiator und Participant	Specialisation, Inheritance	SPECIALISATION, INHERITANCE
Zugriff auf Objektinterna	Internal	INTERNAL
Zustandsveränderungen	StateChange	STATECHANGE
Beziehungen zwischen Zeilen eines Skripts	Eventflow	EVENTFLOW

Tabelle 6: Abbildung Skript auf Konzeptuelles Schema

5.2.2.1 Aufbau der Tabellenspalte

Jede Zeile eines Skripts drückt eine Beziehung aus und besitzt einen Typ. Dies geschieht in der Tabellenspalte Connection nach folgender eBNF:[1]

```
<Inhalt Spalte Connection> ::= <Connection>;
<Connection> ::=(<Beziehungsname>) + <Beziehungstyp> +
(<Spezifikation>);
<Beziehungstyp> ::= „:” + <Typ der Beziehung>;
<Spezifikation> ::= 1{<Prädikat>}M;
*<Prädikat> ::= Siehe Abschnitt 5.2.6;*
*<Beziehungsname> ::= Zeichenkette;*
*<Typ der Beziehung> ::= Knotentyp des Typs ASSOCIATIVELINK im
                    Konzeptuellen Schema;*
```

[1] Einige Produktionsregeln werden aus Gründen der besseren Lesbarkeit wiederholt.

Die Einträge in der Tabellenspalte Connection entsprechen in der tabellarischen Darstellung der Skripte – unter Verwendung der definierten eBNF – folgendem Muster:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	[(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]		(<Beziehungsname>)+ „:“+<Typ der Beziehung>+ 0{<Prädikat>}M	[(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]	

Skript 3: Aufbau des Eintrags der Tabellenspalte Connection

Die Tabellenspalte Connection beinhaltet zwingend einen Beziehungstyp und optional einen Beziehungsnamen, durch den die Zeile innerhalb des Skripts identifiziert wird.[1] Der Name wird vor dem Doppelpunkt angegeben. Dieser Name wird als Name des Beziehungsknotens im Konzeptuellen Schema abgebildet. Der Anwender entscheidet über die Vergabe der Namen und ob diese eindeutig sind. Optional kann eine Spezifikation angegeben werden, die aus einem oder mehreren Prädikaten besteht.

5.2.2.2 Modell der Tabellenspalte

Die Tabellenspalte Connection dient als Container für einen Knoten des Metatyps *ASSOCIATIVELINK* im Konzeptuellen Schema:

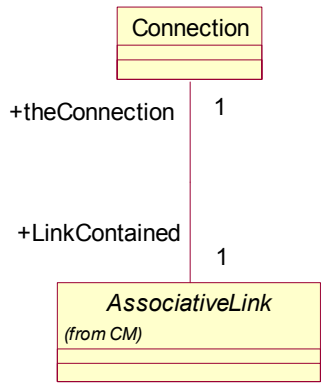


Abbildung 78: Aufbau der Tabellenspalte Connection als UML-Klassendiagramm

[1] Da die Angabe eines Namens optional ist, kann er nicht als (identifizierender) Schlüssel verwendet werden. Er muss optional sein, weil in der objektorientierten Modellierung Vererbungsbeziehungen üblicherweise keinen Namen erhalten.

Der Inhalt dieser Tabellenspalte – im Metamodell des Skriptmodells die Klasse *CONNECTION* – steht in einer UML-Assoziationsbeziehung zum entsprechenden Objekt der Unterklassen von *ASSOCIATIVELINK*.[1] Das Löschen einer Zeile im Skript hat das Löschen des entsprechenden Beziehungsknotens im Konzeptuellen Schema zur Folge. Für jede Zeile eines Skripts wird im Konzeptuellen Schema ein separater Beziehungsknoten erzeugt. Beziehungstypen erben im Konzeptuellen Schema vom abstrakten Beziehungsknotentyp *ASSOCIATIVELINK*. Die Abbildung des Inhalts der Tabellenspalte Connection in das Konzeptuelle Schema ist vom verwendeten Beziehungstyp abhängig. Dies wird in den folgenden Abschnitten dargestellt.

5.2.3 Die Tabellenspalten Initiator Responsibility und Participant Responsibility

5.2.3.1 Aufbau der Tabellenspalten

Der Aufbau der Tabellenspalten Initiator Responsibility und Participant Responsibility (abgebildet durch *INITIATOR RESPONSIBILITY* und *PARTICIPANT RESPONSIBILITY*) ist vom Typ der Beziehung abhängig, der in der Tabellenspalte Connection angegeben wird. Hier sind folgende Fälle zu unterscheiden:

1. Wenn die Beziehung ein von *RESPONSIBILITYCOOPERATION* abgeleiteter Subtyp ist, beinhalten Initiator Responsibility und Participant Responsibility Intensionsknotentypen.
2. Wenn die Beziehung ein von *RELATIONSHIP* abgeleiteter Subtyp ist, bleiben Initiator Responsibility und Participant Responsibility leer.
3. Wenn die Beziehung vom Typ *EVENTFLOW* ist, beinhalten Initiator Responsibility und Participant Responsibility Verweise auf andere Zeilen des Skripts. Nun bleiben die Tabellenspalten Initiator und Participant leer.

Daraus ergibt sich folgende eBNF:[2]

```
<Inhalt Spalte Initiator Responsibility> ::= [<Responsibility> | Ø ];
<Inhalt Spalte Participant Responsibility> ::= [<Responsibility> | Ø ];
<Responsibility> ::= [<Intension> + (<Spezifikation>) | <Verweis> +
(<Spezifikation>) | Ø ];
<Verweis> ::= <Beziehungsname> + <Beziehungstyp>;
```

[1] *ASSOCIATIVELINK* ist eine abstrakte Klasse, deren Extension leer ist. Inhalt der Tabellenspalte ist ein Objekt der nicht abstrakten Subklassen von *ASSOCIATIVELINK*.

[2] Einige Produktionsregeln werden aus Gründen der besseren Lesbarkeit wiederholt.

```
<Beziehungstyp> ::= „:" + <Typ der Beziehung>;
*<Beziehungsname> ::= *Zeichenkette;*
<Spezifikation> ::= 1{<Prädikat>}M;
*<Prädikat> ::= Siehe Abschnitt 5.2.6;*
*<Typ der Beziehung> ::= Knoten des Metatyps EVENTFLOW im Konzeptuellen
                         Schema;*
<Intension> ::= [<InitiatorIntension>|<ParticipantIntension>] ;
<InitiatorIntension> ::= (<Intensionsname>)+ <Intensionstyp>;
<ParticipantIntension> ::= <Intensionsname> + <Intensionstyp>;
<Intensionstyp> ::= „:" + <Typ der Intension>;
<Intensionsname> ::= *Zeichenkette*;
*<Typ der Intension> ::= Knoten des Metatyps INODE im Konzeptuellen
Schema;*
```

Das Metamodell der Tabellenspalten Initiator Responsibility und Participant Responsibility entspricht folgendem UML-Klassendiagramm:

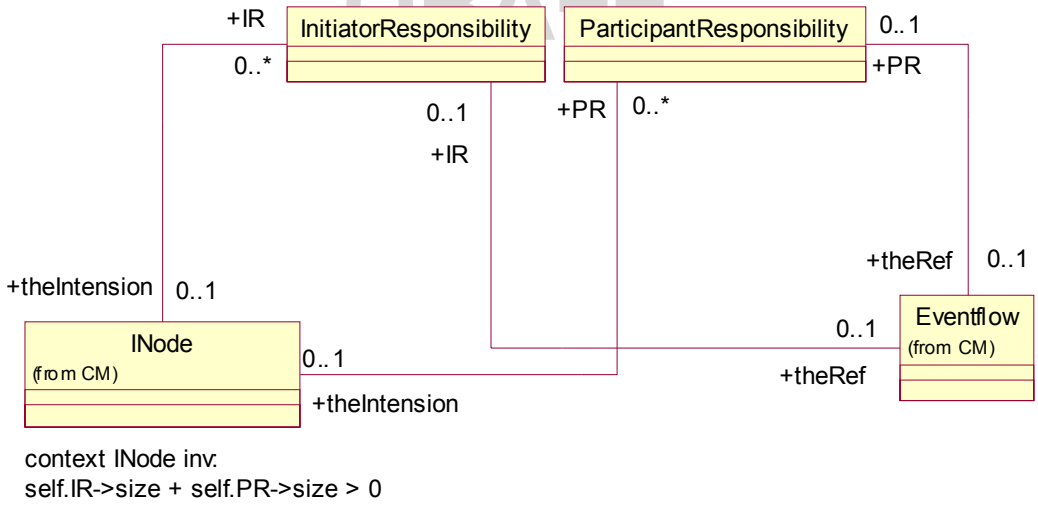


Abbildung 79: Aufbau der Tabellenspalte Initiator Responsibility und Participant Responsibility als UML-Klassendiagramm

Die beiden Responsibility-Tabellenspalten beinhalten entweder

- ein Element der Intension einer Klasse, welche im Konzeptuellen Schema durch einen Knoten der Klasse *INODE* abgebildet wird, oder
- einen Verweis auf eine andere Zeile im Skript, welche im Konzeptuellen Schema durch einen Knoten des Metatyps *EVENTFLOW* abgebildet wird, oder

- beide Tabellenspalten bleiben leer.

Ein Objekt der Klasse *INODE* kann mehrfach als Initiator Responsibility oder Participant Responsibility verwendet werden. Es muss aber entweder mindestens einmal als Initiator Responsibility oder mindestens einmal als Participant Responsibility verwendet werden. Daraus folgt, dass es keine Objekte der Klasse *INODE* gibt, die nicht mindestens in einem Skript verwendet wurden. Man beachte, dass dies in der UML nicht durch die Kardinalitäten der Beziehungen erkennbar ist, sondern durch OCL-Ausdrücke (`context INode inv: self.IR -> size + self.PR->size > 0`) formuliert werden muss. Bei Objekten der Klasse *EVENTFLOW* ist die Situation anders, da sie in zwei unterschiedlichen Situationen verwendet werden. Sie besitzen nur dann einen Verweis auf ein Objekt der Klasse *INITIATOR* oder *PARTICIPANT RESPONSIBILITY,* wenn sie in einem Skript verwendet werden, um zwischen den Zeilen eines Skripts Abhängigkeitsbeziehungen festzuhalten (vgl. Abschnitt 5.2.3.6). Bei der Abbildung von Reihenfolgebeziehungen zwischen Skripten (vgl. Abschnitt 5.2.7) haben Objekte der Klasse *EVENTFLOW* keinen solchen Verweis.

Für die Verwendung von Namen in den Tabellenspalten Initiator Responsibility und Participant Responsibility gilt: Wird in den Tabellenspalten Initiator Responsibility und Participant Responsibility ein Element der Intension einer Klasse angegeben, hat dies die Form `Intensionsname :Intensionstyp`. Dabei ist für die Tabellenspalte Participant Responsibility die Angabe eines Namens zwingend. Für die Tabellenspalte Initiator Responsibility ist die Angabe eines Namens zu empfehlen[1], aber nicht zwingend.

5.2.3.2 Abbildung der Interaktionsbeziehung im Konzeptuellen Schema

Das folgende Skript stellt das Muster einer Interaktionsbeziehung zwischen einem Initiator und einem Participant dar. Im Skript wird der Beziehungstyp Interaction verwendet, der im Konzeptuellen Schema durch einen Beziehungsknotentyp *INTERACTION* repräsentiert wird. Der Aufbau der Tabellenspalte Initiator Responsibility ist `Aktionsname :Action` oder `Ausnahmebezeichnung :Exception` sowie weitere Prädikate zur Spezifikation der Aktion oder Ausnahme.[2] Der Aufbau der Tabellenspalte Participant Responsibility ist `Operationsname :Operation` sowie weitere Prädikate zur Spezifikation der Operation. Man beachte, dass bei Interaktionsbeziehungen die Spalten Initiator und Participant nicht leer sein dürfen:

---

[1] Dies ist notwendig, weil sich sonst Interaktionsdiagramme der UML nicht in Skripte übertragen ließen.

[2] Im Metamodell des Konzeptuellen Schemas sind Ausnahmen als Subtyp von Aktionen definiert. Aus diesem Grund wird im weiteren Verlauf nicht mehr auf den Sonderfall von Ausnahmen eingegangen.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	(<Aktionsname>) + “:Action” + 0{<Prädikat>}M	(<Beziehungs-name>)+ „:Interaction” + 0{<Prädikat>}M	(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	<Operationsname> + “:Operation” + 0{<Prädikat>}M

Skript 4: Muster der Interaktionsbeziehung

Eine Interaktionsbeziehung wird nach folgendem Verfahren in das Konzeptuelle Schema abgebildet:

1. Der Inhalt der Tabellenspalte Connection wird zerlegt, wobei der Doppelpunkt als Trennzeichen zur Identifikation des Beziehungstyps verwendet wird. Der in der Tabellenspalte Connection angegebene Beziehungstyp der Skriptzeile wird als entsprechender Beziehungsknotentyp im Konzeptuellen Schema abgebildet (hier: *INTERACTION*). Wird in der Tabellenspalte Connection ein Beziehungsname verwendet, dann wird dieser als Name des Beziehungsknotens im Konzeptuellen Schema verwendet. Eventuell verwendete Prädikate werden als Attributwerte des Beziehungsknotens gespeichert.[1]

2. Der Inhalt der Tabellenspalten Initiator und Participant wird nach dem im Abschnitt 5.2.1.2 geschilderten Verfahren erzeugt.

3. Es wird überprüft, ob die an der Beziehung teilhabenden Klassen über eine Intensionsbeziehung (Beziehungsknotentyp *INTENSION*) mit Intensionsknoten verbunden sind, die den verwendeten Typ (hier: *ACTION, EXCEPTION* und *OPERATION*) und die verwendeten Namen besitzen. Ist dies nicht der Fall, werden sie erzeugt und mit den in Schritt 1 erzeugten Klassenknoten durch eine Intensionsbeziehung verbunden. Fehlt für *ACTION* oder *EXCEPTION* ein Name, wird in jedem Fall ein neuer Intensionsknoten angelegt.[2]

4. Der in Schritt 1 erzeugte Beziehungsknoten verbindet die in Schritt 3 erzeugten Intensionsknoten. Ausgangspunkt der Beziehung ist der *ACTION*-Knoten aus der Tabellenspalte Initiator Responsibility, Endpunkt ist der *OPERATION*-Knoten aus der Tabellenspalte Participant Responsibility.

[1] Auf die Details der Speicherung der Prädikate wird hier nicht eingegangen, da dies abhängig von einem ggf. verwendeten Softwaresystem ist und somit ein Detail der Realisierung des hier entwickelten Modellierungsansatzes darstellt.

[2] Dies ist notwendig, weil sich sonst Interaktionsdiagramme der UML nicht in Skripte übertragen ließen.

5. Wenn in Schritt 2 für den Initiator ein Rollenknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFROLE* mit den in Schritt 3 erzeugten Beziehungsknoten des Typs *INTENSION* verbunden. Das gleiche Verfahren wird für den Participant verwendet.
6. Wenn in Schritt 2 für den Initiator ein Objektknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFOBJECT* mit dem in Schritt 3 erzeugten Beziehungsknoten des Typs *INTENSION* verbunden. Das gleiche Verfahren wird für den Participant verwendet.

Das Ergebnis ist im Konzeptuellen Schema ein Semantisches Netz der folgenden Form:

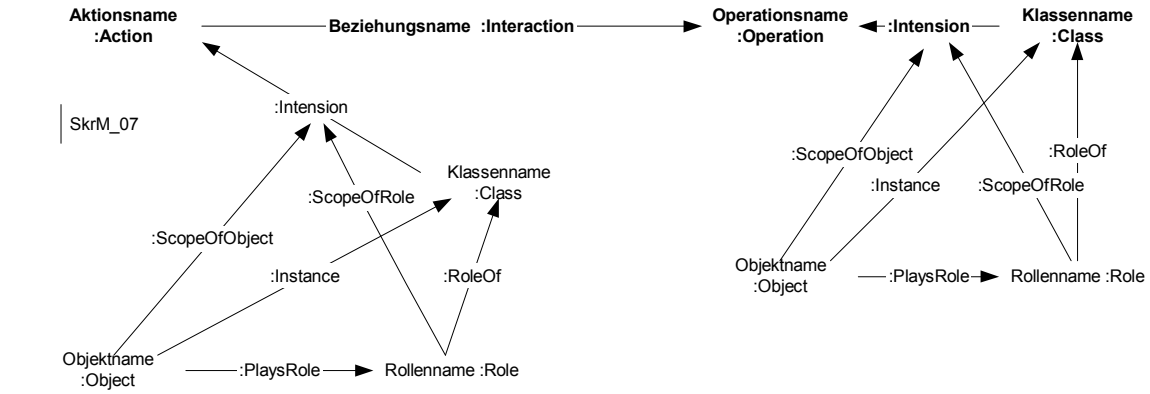


Abbildung 80: Repräsentation einer Interaktionsbeziehung im Konzeptuellen Schema

5.2.3.3 Abbildung der internen Beziehung im Konzeptuellen Schema

Wird durch die Aktivität eines Objekts ein Attribut (d. h. der konkrete Zustand) oder der abstrakte Zustand des Objekts verändert, wird dies im Skript durch eine interne Beziehung abgebildet, die im Konzeptuellen Schema durch einen Beziehungsknotentyp *INTERNAL* repräsentiert wird. Der Aufbau der Tabellenspalte Initiator Responsibility ist `Aktionsname :Action` oder `Ausnahmebezeichnung :Exception` sowie weitere Prädikate zur Spezifikation der Aktion oder Ausnahme.[1] Der Aufbau der Tabellenspalte Participant Responsibility ist `Attributname :Attribute`, `Zustandsname :State` oder `Invariantenname :Invariant` sowie weitere Prädikate zur Spezifikation. Man beachte, dass bei internen Beziehungen die Spalten Initiator und Participant nicht leer sein dürfen:

[1] Im Metamodell des Konzeptuellen Schemas sind Ausnahmen als Subtyp von Aktionen definiert. Aus diesem Grund wird im weiteren Verlauf nicht mehr auf den Sonderfall von Ausnahmen eingegangen.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
[(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]	(<Aktionsname>) + “:Action” + 0{<Prädikat>}M	(<Beziehungs-name>) + „:Internal” + 0{<Prädikat>}M	[(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M | Ø]	[<Attributname> + „:Attribute“ + 0{<Prädikat>}M | <Zustandsname> + „:State“ + 0{<Prädikat>}M | <Invariantenname> + „:Invariant“ + 0{<Prädikat>}M]

Skript 5: Muster für interne Beziehungen

Die Abbildung im Konzeptuellen Schema erfolgt ähnlich wie für Interaktionsbeziehungen.

1. Der in der Tabellenspalte Connection angegebene Beziehungstyp der Skriptzeile (hier: *INTERNAL*) wird nach dem im Abschnitt 5.2.3.2, Punkt 1, geschilderten Verfahren als entsprechender Beziehungsknoten im Konzeptuellen Schema abgebildet.

2. Der Inhalt der Tabellenspalten Initiator und Participant wird nach dem im Abschnitt 5.2.1.2 geschilderten Verfahren erzeugt. Die in Initiator und Participant verwendeten Objekt- und Klassennamen haben identisch zu sein. Die Rollennamen dürfen sich unterscheiden.

3. Es wird überprüft, ob die an der Beziehung teilhabenden Klassen über eine Intensionsbeziehung (Beziehungsknotentyp *INTENSION*) mit Intensionsknoten verbunden sind, die den in Initiator Responsibility und Participant Responsibility verwendeten Typ (hier: *ACTION* oder *EXCEPTION* auf der Seite des Initiators und *ATTRIBUTE*, *STATE* oder *INVARIANT* auf der Seite des Participants) und die verwendeten Namen besitzen. Ist dies nicht der Fall, werden sie erzeugt und mit den in Schritt 2 erzeugten Klassenknoten durch eine Intensionsbeziehung verbunden. Fehlt für *ACTION* oder *EXCEPTION* ein Name, wird in jedem Fall ein neuer Intensionsknoten angelegt.[1]

4. Der in Schritt 1 erzeugte Beziehungsknoten verbindet die in Schritt 3 erzeugten Intensionsknoten. Ausgangspunkt der Beziehung ist der *ACTION*- oder *EXCEPTION*-Knoten aus der Tabellenspalte Initiator Responsibility, Endpunkt ist der *STATE*, *ATTRIBUTE* oder *INVARIANT*-Knoten aus der Tabellenspalte Participant Responsibility.

5. Wenn in Schritt 2 für den Initiator ein Rollenknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFROLE* mit dem in Schritt 3 erzeugten Intensionsknoten verbunden. Das gleiche Verfahren wird für den Participant verwendet.

[1] Dies ist notwendig, weil sich sonst Interaktionsdiagramme der UML nicht in Skripte übertragen ließen.

6. Wenn in Schritt 2 ein Objektknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFOBJECT* mit dem in Schritt 3 erzeugten Intensionsknoten verbunden.

Für den Fall einer Aktion auf Seiten des Initiators und einem Attribut auf Seiten des Participants ist das Ergebnis ein Semantisches Netz der folgenden Form:

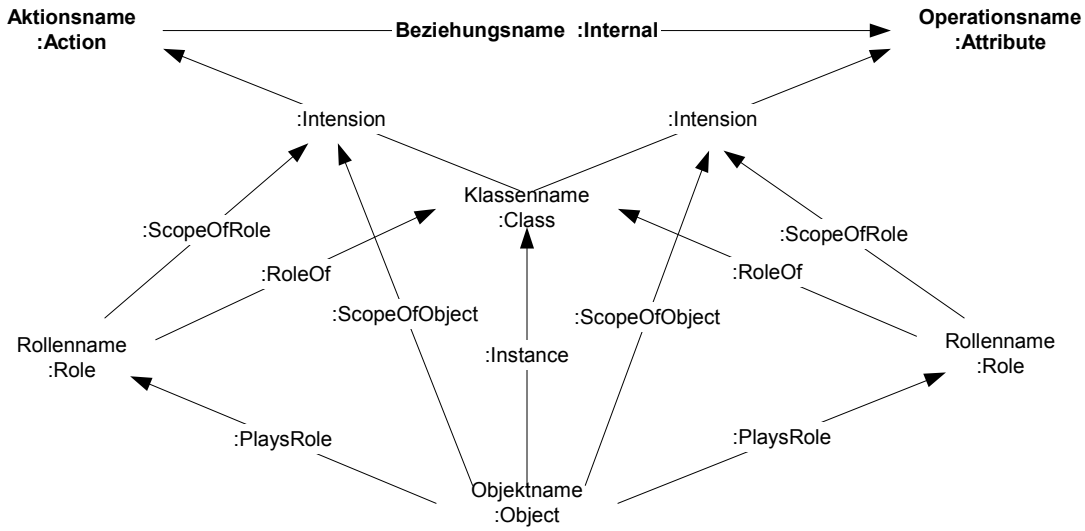


Abbildung 81: Repräsentation einer internen Beziehung im Konzeptuellen Schema

5.2.3.4 Abbildung von Zustandsübergängen im Konzeptuellen Schema

Werden Zustandsübergänge der Objekte einer Klasse abgebildet, wird dies im Skript durch eine Beziehung des Typs Statechange abgebildet, die im Konzeptuellen Schema durch einen Beziehungsknotentyp *STATECHANGE* repräsentiert wird. Der Aufbau der Tabellenspalten Initiator Responsibility und Participant Responsibility ist identisch und entspricht `Zustandsname :State` sowie weiteren Prädikaten zur Spezifikation des Zustands. Man beachte, dass bei Zustands-übergängen die Spalten Initiator und Participant nicht leer sein dürfen:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	<Zustandsname> + “:State” + 0{<Prädikat>}M	(<Beziehungsname>) + “:Statechange” + 0{<Prädikat>}M	(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	<Zustandsname> + “:State” + 0{<Prädikat>}M

Skript 6: Muster eines Zustandsübergangs

Die Abbildung im Konzeptuellen Schema ähnelt der von Interaktionsbeziehungen, wobei folgende Unterschiede zu beachten sind:

1. Wenn für die angegebene Klasse noch keine Intensionsknoten vom Typ *STATE* mit dem verwendeten Zustandsnamen existieren, werden sie erzeugt.
2. Die in Initiator und Participant verwendeten Objekt- und Klassennamen haben identisch zu sein. Die Rollennamen dürfen sich unterscheiden. Existieren die entsprechenden Rollen, Objekte und Klassen im Konzeptuellen Schema noch nicht, werden sie – wie unter Abschnitt 5.2.1.2 beschrieben – erzeugt.
3. Der in der Tabellenspalte Connection angegebene Beziehungstyp der Skriptzeile (hier: *STATECHANGE*) wird nach dem im Abschnitt 5.2.3.2, Punkt 1, geschilderten Verfahren als entsprechender Beziehungsknoten im Konzeptuellen Schema abgebildet.
4. Dieser verbindet die unter Initiator und Participant Responsibility genannten Intensionsknoten des Typs *STATE*. Ausgangspunkt der Beziehung ist der *STATE*-Knoten aus der Tabellenspalte Initiator Responsibility, Endpunkt ist der *STATE*-Knoten aus der Tabellenspalte Participant Responsibility.

Das Ergebnis ist im Konzeptuellen Schema ein Semantisches Netz der folgenden Form:

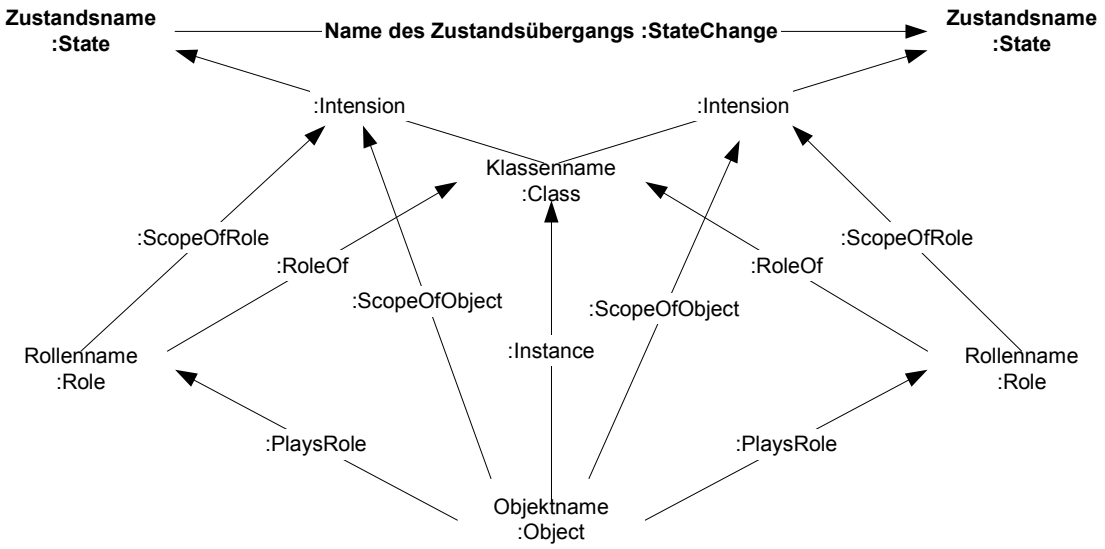


Abbildung 82: Abbildung eines Zustandsübergangs im Konzeptuellen Schema

5.2.3.5 *Abbildung referentieller und taxonomischer Beziehungen im Konzeptuellen Schema*

Statische Beziehungen werden im Konzeptuellen Schema durch Beziehungsknoten des Typs *RELATIONSHIP* bzw. den definierten Untertypen abgebildet. Die gleichen Bezeichnungen werden im Skriptmodell verwendet. Bei statischen Beziehungen bleiben im Skriptmodell die Tabellenspalten Initiator Responsibility und Participant Responsibility leer.[1] Man beachte, dass bei referentiellen und taxonomischen Beziehungen die Spalten Initiator und Participant nicht leer sein dürfen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	Ø	(<Beziehungsname>) + :Relationship + 0{<Prädikat>}M	(<Objektname>)+ („/“ + <Rollenname>) + „:“+<Klassenname> + 0{<Prädikat>}M	Ø

Skript 7:Aufbau einer statischen Beziehung als Skript

Alle statischen Beziehungen werden nach folgendem Verfahren im Konzeptuellen Schema abgebildet:

1. Der in der Tabellenspalte Connection angegebene Beziehungstyp der Skriptzeile (z. B. Association) wird nach dem im Abschnitt 5.2.3.2, Punkt 1, geschilderten Verfahren als entsprechender Beziehungsknoten im Konzeptuellen Schema abgebildet.
2. Der Inhalt der Tabellenspalten Initiator und Participant wird nach dem im Abschnitt 5.2.1.2 geschilderten Verfahren erzeugt.
3. Der in Schritt 1 erzeugte Beziehungsknoten verbindet die in Schritt 2 erzeugten Klassenknoten. Ausgangspunkt der Beziehung ist der *CLASS*-Knoten aus der Tabellenspalte Initiator, Endpunkt ist der *CLASS*-Knoten aus der Tabellenspalte Participant.
4. Wenn in Schritt 2 für den Initiator ein Rollenknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFROLE* mit dem in Schritt 1 erzeugten Beziehungsknoten verbunden. Das gleiche Verfahren wird für den Participant verwendet.
5. Wenn in Schritt 2 für den Initiator ein Objektknoten erzeugt wird, wird er über eine Beziehung des Typs *SCOPEOFOBJECT* mit dem in Schritt 1 erzeugten Beziehungsknoten verbunden. Das gleiche Verfahren wird für den Participant verwendet.

Das Ergebnis ist im Konzeptuellen Schema ein Semantisches Netz der folgenden Form:

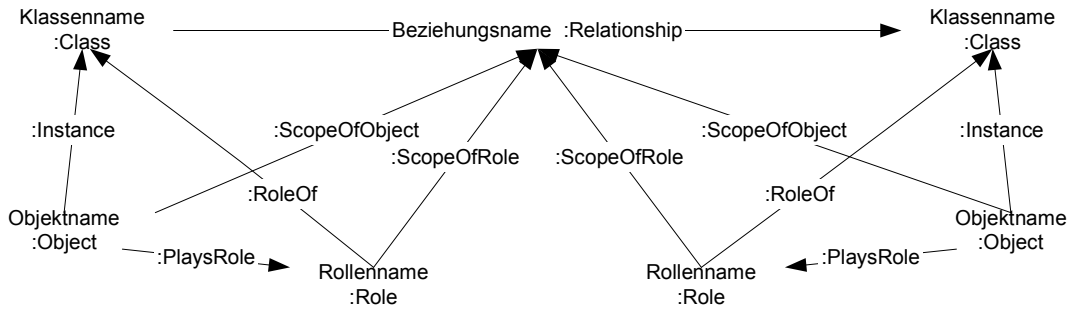


Abbildung 83: Muster einer statischen Beziehung im Konzeptuellen Schema

5.2.3.6 Abbildung von Abhängigkeitsbeziehungen im Konzeptuellen Schema

Zwischen den Zeilen eines Skripts können Abhängigkeitsbeziehungen definiert werden. In diesem Fall stehen in den Tabellenspalten Initiator Responsibility und Participant Responsibility Verweise auf andere Zeilen innerhalb des Skripts. Ein Verweis auf eine andere Zeile liegt vor, wenn der Beziehungsname und -typ aus der Tabellenspalte Connection einer anderen Zeile des Skripts übernommen wird. Man beachte, dass bei Abhängigkeitsbeziehungen die Tabellenspalten Initiator und Participant leer bleiben. Diese Anordnung wurde gewählt, da sonst der entsprechende Eintrag als Klassen- und Objektbezeichner missverstanden werden könnte. Im Skriptmodell wird der Beziehungstyp Eventflow verwendet, der im Konzeptuellen Schema durch einen Beziehungsknoten des Metatyps *EVENTFLOW* repräsentiert wird. Eine Zeile hat damit folgenden Aufbau:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Ø	<Verweis>	<Beziehungsname> :Eventflow 0{<Prädikat>}M	Ø	<Verweis>

Skript 8: Aufbau einer Abhängigkeitsbeziehung in der Skriptnotation

Im folgenden Beispiel ist durch den Ereignisfluss die Abhängigkeit modelliert worden, dass die Interaktion Int2 nach der Interaktion Int1 stattfindet:

[1] Siehe Abschnitt 5.2.3.1.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:A	a :Action	Int1 :Interaction	:B	b :Operation
	:C	c :Action	Int2 :Interaction	:D	d :Operation
		Int1 :Interaction	Abhängigkeit_X :Eventflow		Int2 :Interaction

Skript 9: Beispiel für eine Abhängigkeit zwischen Interaktionen

Abhängigkeitsbeziehungen werden nach folgendem Verfahren im Konzeptuellen Schema abgebildet:

1. Der in der Tabellenspalte Connection angegebene Beziehungstyp der Skriptzeile (hier: *EVENTFLOW*) wird nach dem im Abschnitt 5.2.3.2, Punkt 1, geschilderten Verfahren als entsprechender Beziehungsknoten im Konzeptuellen Schema abgebildet.
2. Dieser Beziehungsknoten wird mit dem in der Tabellenspalte Initiator Responsibility angegebenen Modellelement und dem in der Tabellenspalte Participant Responsibility angegebenen Modellelement verbunden. Dafür wird in jeder Zeile des Skripts überprüft, ob der Beziehungsname und der Beziehungstyp mit dem Namen und Beziehungstyp in der Tabellenspalte Initiator Responsibility der aktuellen Zeile übereinstimmt. Der zugehörige Beziehungsknoten im Konzeptuellen Schema wird als Ausgangspunkt eines Beziehungsknotens des Typs *EVENTFLOW* verwendet. Anschließend wird in jeder Zeile des Skripts überprüft, ob der Beziehungsname und der Beziehungstyp mit dem Namen und Beziehungstyp in der Tabellenspalte Participant Responsibility der aktuellen Zeile übereinstimmt. Der zugehörige Beziehungsknoten im Konzeptuellen Schema wird als Endpunkt des Beziehungsknotens verwendet.

Das Ergebnis ist im Konzeptuellen Schema ein Semantisches Netz der folgenden Form:

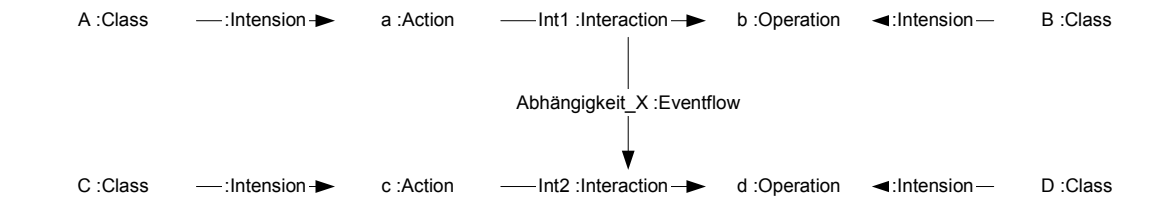


Abbildung 84: Beziehungstyp EVENTFLOW *im Konzeptuellen Schema*

Während im Konzeptuellen Schema Beziehungsknotentypen keinen Namensraum besitzen, müssen die Namen der referenzierten Beziehungen innerhalb des Skripts eindeutig sein, wenn in einem Skript Abhängigkeitsbeziehungen verwendet werden. Bei Beziehungen zwischen Zeilen eines Skripts handelt es sich bei den verbundenen Modellelementen in jedem Fall um Knoten des Typs

RESPONSIBILITYCOOPERATION bzw. um einen der im Metamodell des Konzeptuellen Schemas definierten Subtypen. Das hat zur Folge, dass nur zwischen dynamischen Beziehungen Ereignisflüsse definiert werden können. Insofern stellt die Verwendung des Ereignisflusses im Skriptmodell eine Einschränkung des Metamodells des Konzeptuellen Schemas dar, weil im Konzeptuellen Schema Ereignisflüsse alle Modellelemente verbinden dürfen:[1]

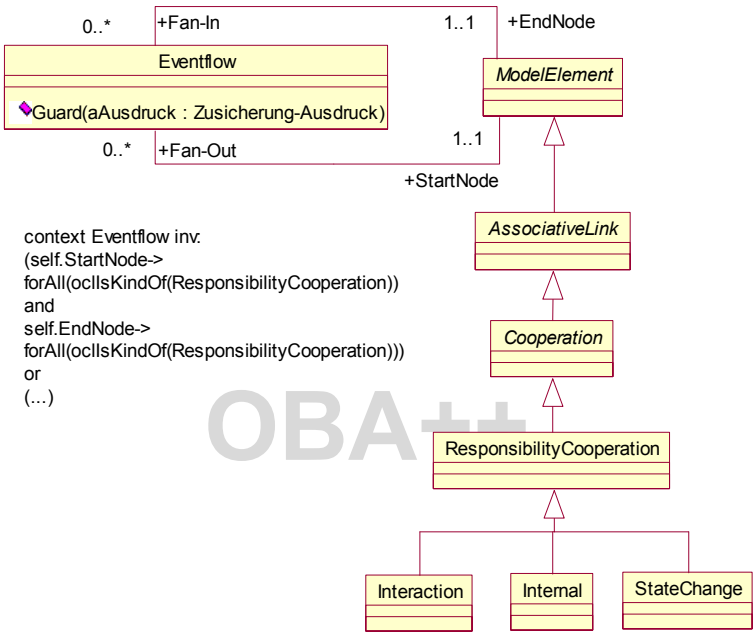


Abbildung 85: Metamodell des Ereignisflusses als UML-Klassendiagramm

5.2.4 Die Zeilennummer

5.2.4.1 Aufbau der Tabellenspalte

Die Tabellenspalte Zeilennummer enthält einen *Sequenzausdruck* mit dem folgenden Aufbau:

```
<Inhalt Spalte Number> ::= (<Sequenzausdruck>);
<Sequenzausdruck> ::= (<Vorgängerausdruck> + "/") + <Sequenznummer>;
<Vorgängerausdruck>::= [<Sequenznummernausdruck> | <Sequenznummer>];
<Sequenznummer> ::= 0{<Sequenznummerebene>+"."}M +
```

[1] Die Invariante ist noch nicht vollständig. Sie wird im Abschnitt über Beziehungen zwischen Skripten vervollständigt.

```
                    <Sequenznummerebene> + ”.” ;
<Sequenznummerebene> ::= (<Präfix>) + <LaufendeNummer> + (<Postfix>);
<Präfix> ::= ”A”|”B”|...|”Z”;
<Postfix> ::= ”a”|”b”|...|”z”;
*<Sequenznummernausdruck> ::= Aus den logischen Operatoren XOR, AND,
                              NOT, OR, Klammern und Sequenznummern
                              gebildeter Ausdruck;*
<LaufendeNummer> ::= *Natürliche Zahl und die Null*;
```

5.2.4.2 Abbildung der Tabellenspalte Zeilennummer im Konzeptuellen Schema

Im Skriptmodell werden neben Abhängigkeitsbeziehungen zusätzlich auch sog. Sequenznummern verwendet, um Reihenfolgebeziehungen zwischen den Zeilen von Skripten festlegen zu können.[1] Im Konzeptuellen Schema wird diese Abfolge der Skriptzeilen durch Beziehungsknoten des Typs *EVENTFLOW* abgebildet, die zwei Beziehungsknoten des Typs *RESPONSIBILITY-COOPERATION* verbinden.

- Die Sequenznummern werden in Sequenznummernebenen zerlegt. Dabei werden Punkte als Trennzeichen verwendet. Innerhalb einer Sequenznummernebene gilt:
  - In einer Sequenznummernebene wird der Knoten des Typs *RESPONSIBILITY-COOPERATION* der Zeile mit der laufenden Nummer N mit dem Knoten des Typs *RESPONSIBILITYCOOPERATION* verbunden, der die laufende Nummer N+1 besitzt. Beispiel: Zeile `2.` wird mit Zeile `3.` verbunden.
  - In einer Sequenznummernebene wird der Knoten des Typs *RESPONSIBILITY-COOPERATION* der Zeile mit der laufenden Nummer N und dem Präfix x mit dem Knoten des Typs *RESPONSIBILITYCOOPERATION* verbunden, der die laufende Nummer N+1 und den gleichen Präfix x besitzt. Beispiel: Zeile `A2.` wird mit Zeile `A3.` verbunden.
  - In einer Sequenznummernebene wird der Knoten des Typs *RESPONSIBILITY-COOPERATION* der Zeile mit der laufenden Nummer N und dem Postfix x mit dem Knoten des Typs *RESPONSIBILITYCOOPERATION* verbunden, der die laufende Nummer N+1 und den gleichen Postfix x besitzt. Beispiel: Zeile `2A.` wird mit Zeile `3A.` verbunden.
  - In einer Sequenznummernebene wird der Knoten des Typs *RESPONSIBILITY-COOPERATION* der Zeile mit der laufenden Nummer N mit dem Knoten des Typs *RESPONSIBILITYCOOPERATION* verbunden, der auf dieser Ebene ebenfalls die Nummer

[1] Auf die Verwendung von Nummern wird hier noch nicht eingegangen. Die Details hierzu finden sich in dem Kapitel, welches sich mit der Anwendung des Skriptmodells befasst.

N besitzt und eine zusätzliche Ebene, auf der er die laufende Nummer 1. besitzt. Beispiel: Zeile `2.` wird mit der Zeile `2.1.` verbunden.

- Darüber hinaus wird der Knoten des Typs *RESPONSIBILITYCOOPERATION* der Zeile mit der Sequenznummer X mit den Knoten des Typs *RESPONSIBILITYCOOPERATION* verbunden, die im Vorgängerausdruck aufgeführt sind. Der Knoten des Typs *RESPONSIBILITYCOOPERATION* der Zeile mit der Sequenznummer X ist dabei Endpunkt des Beziehungsknotens. Im Vorgängerausdruck werden die Zeichen "!" (Negation bzw. NOR), "&" (Konjunktion bzw. AND), "^" (Disjunktion bzw. XOR), "|" (Adjunktion bzw. AND) und die Klammern ("(" und ")") verwendet.

Die folgende Abbildung dient zur Veranschaulichung des Verfahrens. In einem Skript sind in der Tabellenspalte Number die Sequenzausdrücke 1., 2., 3., 3.| 5. / 7., 1. / 5., 2.1., 2.2., 2.1.1., 2.1.2., 2.1.3A., 2.1.3B., 2.1.3A. & 2.1.3B. / 2.1.5., 2.1.6., 1. / B2., B3., B4. verwendet worden. Nur aus Gründen der Anschaulichkeit haben die Interaktionen den jeweiligen Sequenzausdruck als Namen erhalten. Die übrigen Tabellenspalten bleiben unausgefüllt, da sie keinen Einfluss auf das Verfahren besitzen:

Number			Connection		
1.			1. :Interaction		
2.			2. :Interaction		
3.			3. :Interaction		
3.| 5. / 7.			3.| 5. / 7. :Interaction		
1. / 5.			1. / 5. :Interaction		
2.1.			2.1. :Interaction		
2.2.			2.2 :Interaction		
2.1.1.			2.1.1. :Interaction		
2.1.2.			2.1.2. :Interaction		
2.1.3A.			2.1.3A. :Interaction		
2.1.3B.			2.1.3B. :Interaction		
2.1.3A. & 2.1.3B. / 2.1.5.			2.1.3A. & 2.1.3B. / 2.1.5. :Interaction		
2.1.6.			2.1.6. :Interaction		
1. / B2.			1. / B2. :Interaction		
B3.			B3. :Interaction		
B4.			B4. :Interaction		

Skript 10: Beispielskript zur Demonstration von Zeilennummern

Das Ergebnis ist im Konzeptuellen Schema ein Semantisches Netz der folgenden Form:

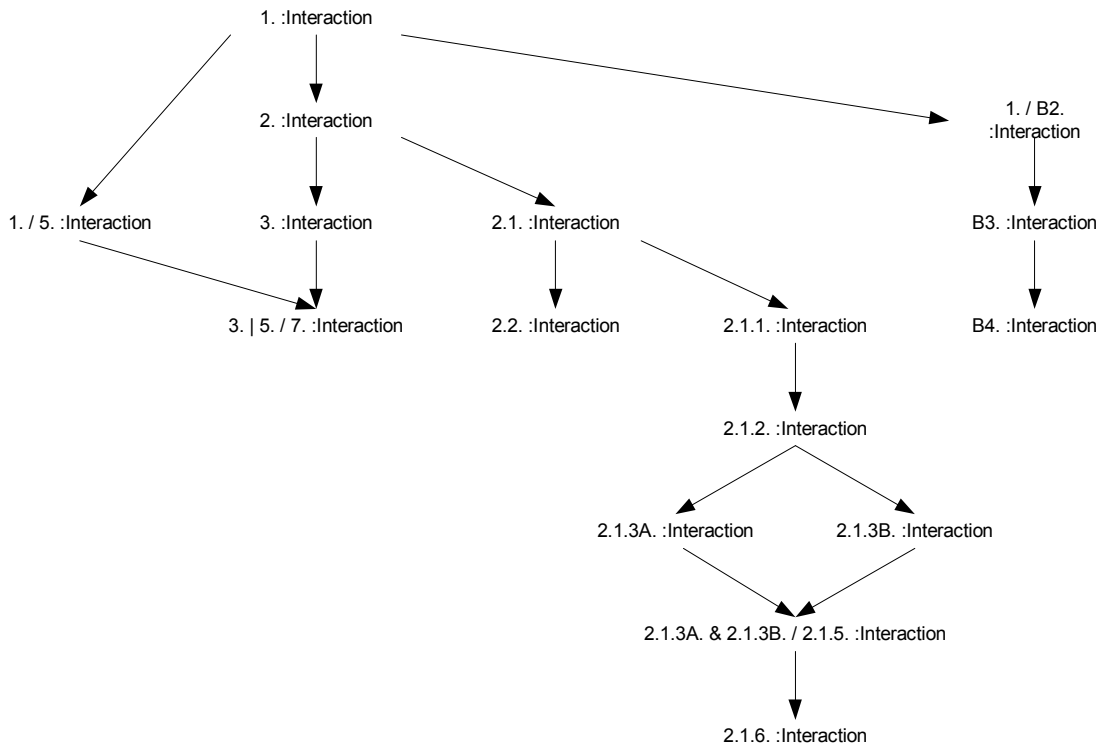


Abbildung 86: Abbildung von Reihenfolgebeziehungen im Konzeptuellen Schema

Die folgenden Beispiele sollen den Abbildungsprozess exemplarisch darstellen.

5.2.5 Beispiele der Abbildung von Skripten auf das Konzeptuelle Schema

Im folgenden Beispiel besteht eine Abteilung Projektbetreuung aus einem Abteilungsleiter und Mitarbeitern. Abteilungsleiter erteilen den Mitarbeitern Anweisungen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Projektbetreuung		:Whole-Part	:Mitarbeiter	
	:Projektbetreuung		:Whole-Part	:Abteilungsleiter	
	:Abteilungsleiter		Ist Vorgesetzter :Association	:Mitarbeiter	
	:Abteilungsleiter	erteilt Anweisung :Action	Erteilt Anweisung :Interaction	:Mitarbeiter	nehmen Anweisung entgegen :Operation

Skript 11: Skript mit Interaktionen und statischen Beziehungen

Daraus entsteht folgendes Semantische Netz:

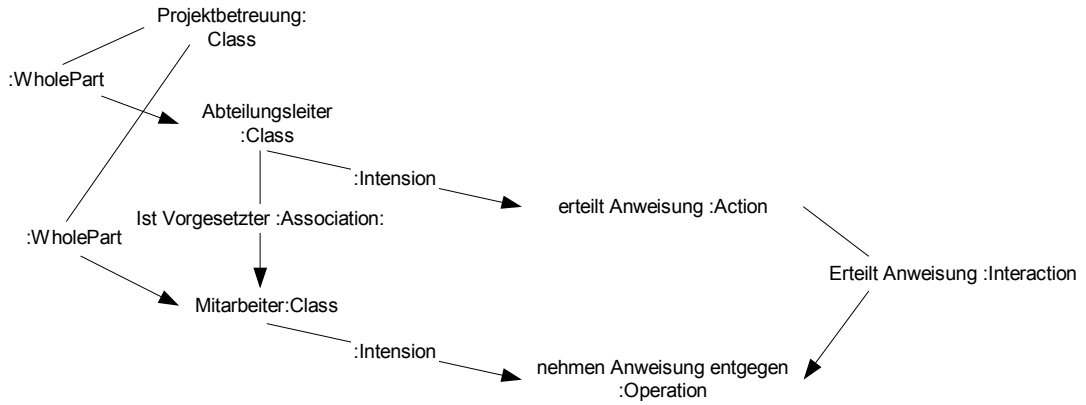


Abbildung 87: Resultierendes Semantisches Netz (1)

Das folgende Skript stellt einen ähnlichen Sachverhalt für benannte Objekte dar. Auf diese Weise werden Beobachtungen auf Objektebene abgebildet. Zusätzlich haben zwei Interaktionen eine Zeilennummer erhalten:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Projektbetreuung		:Whole-Part	:Mitarbeiter	
	:Projektbetreuung		:Whole-Part	:Abteilungsleiter	
	:Abteilungsleiter		Ist Vorgesetzter :Association	:Mitarbeiter	
1.	Herr Müller :Abteilungsleiter	erteilt Anweisung :Action	Erteilt Anweisung :Interaction	Herr Müller :Mitarbeiter	nehmen Anweisung entgegen :Operation
2.	Herr Müller :Abteilungsleiter	erteilt Anweisung :Action	Erteilt Anweisung :Interaction	Herr Schmitz :Mitarbeiter	nehmen Anweisung entgegen :Operation

Skript 12: Skript mit benannten Objekten

Die Klasse Mitarbeiter besitzt nun eine Extension mit zwei Elementen (Herr Müller und Herr Schmitz). Ein zweiter Herr Müller ist aber ein Exemplar der Klasse `Abteilungsleiter`. Die Operation `nehmen Anweisung entgegen` der Klasse Mitarbeiter und die Aktion `erteilt Anweisung` werden je zweimal verwendet. Im Konzeptuellen Schema ist die Information erhalten geblieben, dass zwei Objekte mit gleichem Namen Exemplare unterschiedlicher Klassen sind und unterschiedliche Aktivitäten zeigen. Darüber hinaus wird der Ereignisfluss durch Kanten des Typs *EVENTFLOW* abgebildet.

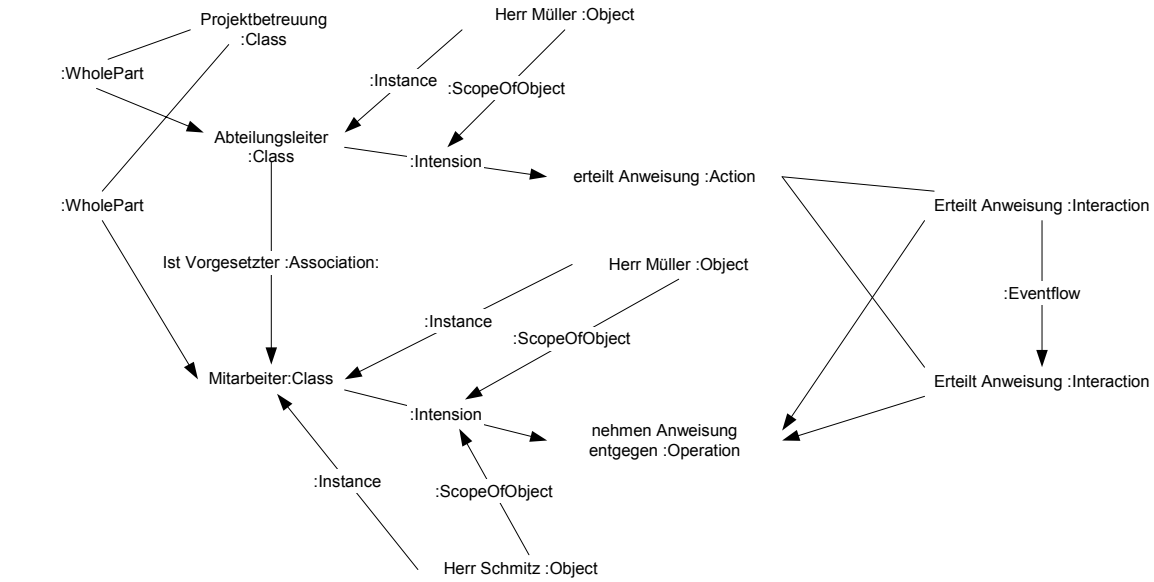


Abbildung 88: Resultierendes Semantisches Netz (2)

Das folgende Skript demonstriert die Abbildung von unterschiedlichen Rollen, die Objekte und Klassen übernehmen. Außerdem wird deutlich, wie Aktivitäten von Klassen in Skripten mehrfach verwendet werden, aber nur einmal im Konzeptuellen Schema existieren:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	Herr Dorn /Vorgesetzter :Geschäftsführer	erteilt Anweisungen :Action	:Interaction	Herr Müller /Untergebener :Abteilungsleiter	nehmen Anweisung entgegen :Operation
2.	Herr Müller /Vorgesetzter :Abteilungsleiter	erteilt Anweisungen :Action	:Interaction	Herr Schmitz /Untergebener :Mitarbeiter	nehmen Anweisung entgegen :Operation

Skript 13: Skript mit Objekten und Rollen

Sowohl die Klasse Geschäftsführer wie auch die Klasse Abteilungsleiter verfügt in der Rolle des Vorgesetzten über die Aktivität `erteilt Anweisungen`. Der Abteilungsleiter nimmt in der Rolle des Untergebenen Anweisungen entgegen und erteilt in der Rolle des Vorgesetzten Anweisungen. Sowohl die Klasse Abteilungsleiter wie auch die Klasse Mitarbeiter besitzt die Eigenschaft, Anweisungen entgegenzunehmen.

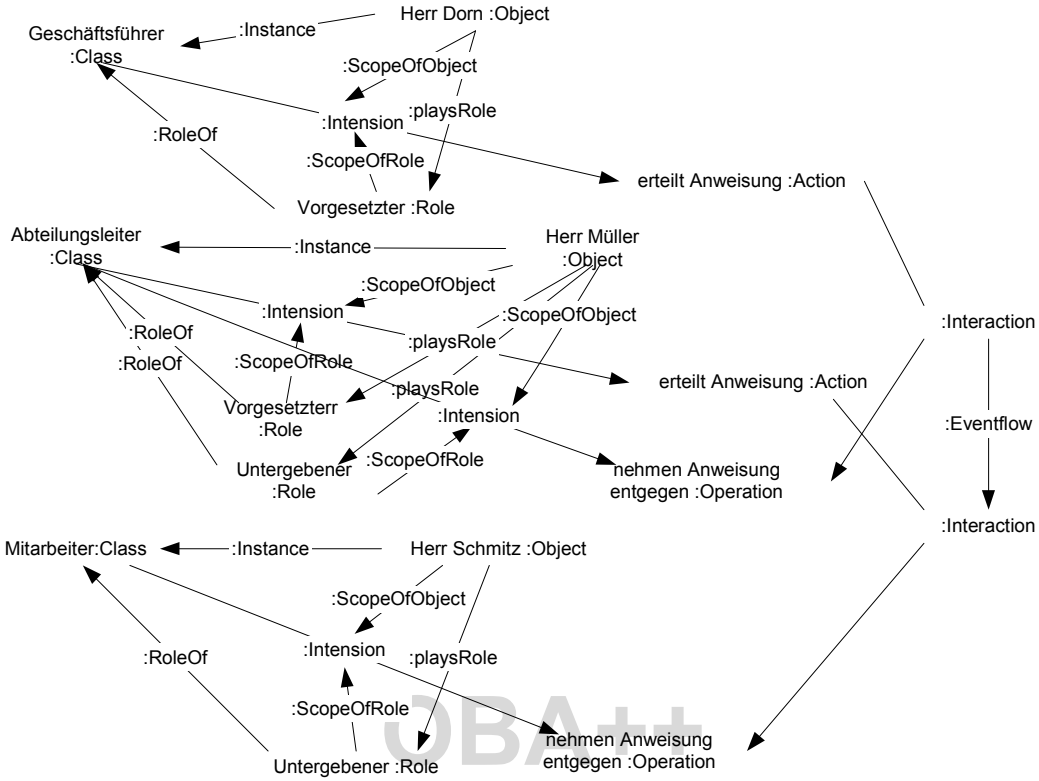


Abbildung 89: Resultierendes Semantisches Netz (3)

5.2.6 Abbildung der Spezifikation

Abgesehen von der Tabellenspalte Zeilennummer kann in den übrigen Tabellenspalten eine Spezifikation angegeben werden, die aus einem oder mehreren Prädikaten besteht.[1] Für jede Tabellenspalte sind eine Reihe von Prädikaten definiert. Auf diese Prädikate wird hier eingegangen.

Die Prädikate der Tabellenspalten Initiator und Participant im Skriptmodell stellen im Konzeptuellen Schema Prädikate des Metatyps *CLASS* dar. Die Prädikate der Tabellenspalte Connection im Skriptmodell stellen im Konzeptuellen Schema Prädikate des jeweils verwendeten Beziehungsknotentyps dar. Auch die Prädikate der Tabellenspalte Initiator Responsibility und Participant Responsibility im Skriptmodell stellen ebenfalls im Konzeptuellen Schema Prädikate des jeweils verwendeten Beziehungsknotentyps dar.

[1] Siehe die entsprechenden Regeln der eBNF.

Die folgende Tabelle beinhaltet alle in den Skripten verwendete Prädikate des Konzeptuellen Schemas. Die erste Tabellenspalte beinhaltet den Namen des Prädikats. Die zweite Tabellenspalte gibt an, ob dieses Primitiv auch in der UML definiert ist. Die dritte Tabellenspalte gibt die Bedeutung des Prädikats an. Die vierte Tabellenspalte gibt an, in welchen Tabellenspalten eines Skripts das Prädikat verwendet werden kann. Die fünfte Tabellenspalte gibt an, für welchen Metatyp im Konzeptuellen Schema das Prädikat definiert ist. Die letzte Tabellenspalte beinhaltet, welche Arten von Termen für die Prädikate verwendet werden.[1] Auf die Verwendung wird hier nicht eingegangen. Dies ist Inhalt des folgenden Kapitels über die Anwendung des Skriptmodells. Das Zeichen "–" bedeutet, dass das Prädikat keinen Term besitzt.

Name	UML	Bedeutung	Spalte	Metatyp im Konzeptuellen Schema	Art des Terms
`Act`	(✓)	Aktivität im Zustandsübergang	C	*STATECHANGE*	`String`
		Aktivität, die eine Referentielle Bez. verändert.	C	*REFERENTIAL-RELATIONSHIP*	`String`
`ActExpr`	✓	Inhalt der Aktion	IR	*ACTION*	`String`
`after`		Zeitereignis	IR	*ACTION*	`Period, Point Of Reference`
	✓	Zeitraum	C	*STATECHANGE*	
`Answer`		Rückgabe	C	*INTERACTION*	`String`
`Card`	✓	Kardinalität	PR	*ATTRIBUTE*	`Cardinality Expression`
	✓		I, P	*CLASS*	
`Callb`		Rückruf	C	*INTERACTION*	`Operation Name`
`Changeable`	✓	Veränderbarkeit	PR	*ATTRIBUTE*	`Changeability Identifier`
`Concur-rency`	(✓)	Nebenläufigkeit, Synchronisation	C	*INTERACTION*	`Concurrency Kind`
	✓		IR, PR	*ACTIVITY*	
`Content`		Inhalt	C	*INTERACTION*	`String`
				INTERNAL	`String`
				STATECHANGE	`String`
`Continuity`		Stetigkeit	C	*INTERACTION*	`Continuity Identifier`
`Coord`		Koordination	C	*INTERACTION*	`Coordination Identifier`
`Cost`		Kosten	IR, PR	*ACTIVITY*	`Cost Expression`
`Dimension`		Maßeinheit	PR	*ATTRIBUTE*	`String`
`Discrim`		Diskriminator von Spezialisierungs-	C	*SPECIALISATION*	`String`

[1] Vgl. Seite 102.

Name	UML	Bedeutung	Spalte	Metatyp im Konzeptuellen Schema	Art des Terms
		beziehungen			
Desc		Beschreibung	IR, PR, C, I, P	*MODELELEMENT*	String
Err		Fehler-bedingungen	PR	*OPERATION*	Error Condition
execution-Time	(✓)	Ausführungs-dauer	IR, PR	*ACTIVITY*	–
	(✓)		C	*INTERACTION*	
Exp		Qualifizierter Export	PR	*OPERATION*	Identifier List
Expression	✓	Ausdruck	PR	*INVARIANT*	Expression
Flow		Strom	C	*INTERACTION*	Flow Type
Form		Form der Interaktion	C	*INTERACTION*	Form Identifier
Goal		Ziel der Operation	PR	*ACTIVITY*	Expression
Guard	✓	Überwachungs-bedingung	C	*RESPONSIBILITY-COOPERATION*	Constraint Expression
In	(✓)	Argumente	PR	*OPERATION*	Identifier List
Init	✓	Initialisierung eines Attributs	PR	*ATTRIBUTE*	Expression
Kind		Art	C	*INTERACTION*	Interaction Kind
Lifespan		Lebensdauer	C	*REFERENTIAL-RELATIONSHIP*	Lifespan Identifier
Mode		Modus	C	*INTERNAL*	Access Mode
			C	*INTERACTION*	Interaction Mode
			IR, PR	*ACTIVITY*	Activity Mode
			C	*STATECHANGE*	Statechange Mode
			C	*REFERENTIAL-RELATIONSHIP*	Change Mode
Name		Name des Modellelements	IR, PR, C, I, P	*MODELELEMENT*	String
Reply		Benachrichtigung	C	*INTERACTION*	Acknowledgement Identifier
RngCard	✓	Kardinalität des Bilds	C	*REFERENTIAL-RELATIONSHIP*	Cardinality Expression
DomCard	✓	Kardinalität des Urbilds	C	*REFERENTIAL-RELATIONSHIP*	
Out	(✓)	Rückgabe	PR	*OPERATION*	Identifier List
Periodic		Periodische Aktion	IR	*ACTION*	t, In, Int, n
Post	✓	Nachbedingung	PR	*OPERATION*	Expression
Pre	✓	Vorbedingung		*OPERATION*	
Prop		Ausbreitung der Interaktion	C	*INTERACTION*	Propagation Kind, Selector Expression, Order Kind
Qualifier		Qualifizierer	PR	*ATTRIBUTE,*	Identifier List
			PR	*OPERATION*	Identifier List

Name	UML	Bedeutung	Spalte	Metatyp im Konzeptuellen Schema	Art des Terms
receive-Time	(✓)	Empfangszeit-punkt	C	*INTERACTION*	–
Ref		Externe Referenzen	PR	*OPERATION*	Identifier List
sendTime	(✓)	Sendezeitpunkt	C	*INTERACTION*	–
Spec		Formale Spezifikation mit OCL	I, P, IR, PR, C	*MODELELEMENT*	OCLExpression
startTime	(✓)	Startzeitpunkt	IR, PR	*ACTIVITY*	–
stopTime	(✓)	Endzeitpunkt	IR, PR	*ACTIVITY*	–
Stpe	✓	Stereotyp von Modellelementen	I, P, IR, PR, C	*MODELELEMENT*	String
Subrange		Wertebereich eines Attributs	PR	*ATTRIBUTE*	Expression
Subtype		Form der mereologischen Beziehung	C	*WHOLEPART*	Relationship Type
Suspend		Abbruch nach Fristablauf	C	*INTERACTION*	Time Expression, Point Of Reference
System		Systemzu-gehörigkeit	I, P	*CLASS*	System Identifier
Timing-Constraint	(✓)	Zeitliche Einschränkung	IR, PR	*ACTIVITY*	Time Expression
	(✓)		C	*RESPONSIBILITY-COOPERATION*	
Type	✓	Datentyp	PR	*ATTRIBUTE*	Type Identifier
Var		Variabilität	C	*REFERENTIAL-RELATIONSHIP*	Variability Identifier
Vis	✓	Sichtbarkeits-kategorie	PR	*OPERATION*	Visibility Kind
When		Veränderungs-ereignis	IR	*ACTION*	Expression
	✓	Zeitpunkt	C	*STATECHANGE*	

Tabelle 7: Vordefinierte Prädikate

Im Kapitel 11 (Anhang 3: Die Schnittstellen der Klassen des Metamodells) sind für die Metaklassen aus dem Kapitel 4 (Das Konzeptuelle Schema) die definierten Prädikate dargestellt. In der Notation der UML handelt es sich um Operationen der Metaklassen.[1] Exemplarisch sei dies am Beispiel der Klasse *INTERNAL* erläutert:

[1] Vgl. die Darstellung auf Seite 102, wo der Zusammenhang zwischen Operationen und Prädikaten und Argumenten und Termen dargestellt ist.

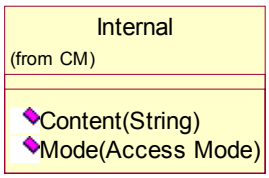


Abbildung 90: Die Prädikate der Metaklasse INTERNAL

Die beiden Operationen `Internal::Content()`[1] und `Internal::Mode()` entsprechen den beiden in der Tabelle 7 angegebenen Prädikaten der Metaklasse *INTERNAL*.[2] Hinzu kommen die Prädikate, die die Metaklasse *INTERNAL* von den Metaklassen *MODELELEMENT*, *ASSOCIATIVELINK*, *COOPERATION* und *RESPONSIBILITYCOOPERATION* erbt. Dies wird aus der Abbildung 72 auf Seite 164 ersichtlich. Verfolgt man die Vererbungsbeziehungen vom Blatt des Vererbungsbaums bis zu seiner Wurzel, ergibt sich unter zu Hilfenahme von Kapitel 11 (Anhang 3: Die Schnittstellen der Klassen des Metamodells), dass für die Metaklasse *INTERNAL* insgesamt folgende Prädikate definiert sind:

- `Content()`
- `Mode()`
- `TimingConstraint()` von *RESPONSIBILITYCOOPERATION*
- `Guard()` von *RESPONSIBILITYCOOPERATION*
- `Name()` von *MODELELEMENT*
- `Desc()` von *MODELELEMENT*
- `Spec()` von *MODELELEMENT*
- `Stpe()` von *MODELELEMENT*

Entsprechendes gilt für die Prädikate der übrigen Metaklassen des Konzeptuellen Schemas. Im Kapitel 11 (Anhang 3: Die Schnittstellen der Klassen des Metamodells) finden sich für alle Metaklassen des Konzeptuellen Schemas die definierten Prädikate in der Notation der UML.

Dieses Verfahren, Prädikate zu vererben, bietet den Vorteil, dass sie – wo immer es sinnvoll ist – an zentraler Stelle definiert sind und alle erbenden Metaklassen über diese Prädikate verfügen. Wenn dies nicht sinnvoll ist, sind die Prädikate für mehrere Metaklassen definiert. Dies sei am

[1] Der *Bereichsauflösungsoperator* „::“ stammt aus der Programmiersprache C++. Auf diese Weise wird festgelegt, dass die Operation im Namensraum der Klasse definiert ist. Vgl. [Stroustrup 2000: S. 29, 89].

[2] Operationen, die sich aus der Realisierung dieses Metamodells ergeben, würden als zusätzliche Operationen hinzugefügt werden. Sie sind hier nicht Gegenstand der Betrachtung.

Beispiel des Prädikats `TimingConstraint()` erläutert. Es ist für die Metaklassen *ACTIVITY* und *RESPONSIBILITYCOOPERATION* definiert, weil es in den erbenden Metaklassen *INTERACTION*, *INTERNAL*, *STATECHANGE*, *ACTION* und *EXCEPTION* verwendet wird. Die Alternative hätte darin bestanden, es in einer gemeinsamen Oberklasse zu definieren. Dies wäre in diesem Fall die Metaklasse *MODELELEMENT*. Die Verwendung von zeitlichen Restriktionen ist zwar für dynamische Beziehungen und Aktivitäten sinnvoll, nicht jedoch für alle Modellelemente.[1] Daraus ergibt sich die Notwendigkeit der Definition für mehrere Metaklassen. Eine andere Situation liegt im Fall des Prädikats `Mode()` vor, welches für die Metaklassen *INTERNAL*, *INTERACTION*, *ACTIVITY*, *STATECHANGE*, *REFERENTIALRELATIONSHIP* definiert ist.[2] Für jede Metaklasse sind andere Modi (`Access Mode`, `Interaction Mode`, `Activity Mode`, `StateChange Mode`, `Change Mode`) vordefiniert. Diese zulässigen Terme der Prädikate sind auf (Meta-)Modellebene unterschiedliche Argumenttypen der Operationen. Dadurch ergibt sich für jede der genannten Metaklassen eine andere *Signatur*[3] der Operation `Mode()`.[4]

Wenn für die Terme eines Prädikats Werte vordefiniert sind, handelt es sich auf der Metamodellebene um einen Aufzählungsdatentyp. Dieser Aufzählungsdatentyp wird als Typ des Arguments der Operation verwendet. Nur die vordefinierten Werte des Aufzählungsdatentyps dürfen bei der Modellierung verwendet werden. In der UML ist dies eine *Enumeration*, die als Klasse mit dem Stereotyp «`enumeration`» gekennzeichnet wird. Alle zulässigen Werte werden als Attribute der Klasse aufgeführt.[5] Dies sei am Beispiel des Prädikats `Mode()` des Metatyps *INTERACTION* erläutert (`Interaction::Mode()`). Die vordefinierten Werte sind durch den Aufzählungstyp `Interaction Mode` festgelegt:

---

[1] Darauf wird noch in Kapitel 6 eingegangen.

[2] Vgl. Tabelle 7 und Kapitel 11.

[3] Die *Signatur* ist der Name einer Operation, der Typ ihres Rückgabewertes und der Typ ihrer Argumente. Vgl. [Wirfs-Brock 1993: S. 21].

[4] Die Definition einer abstrakten Operation `ModelElement::Mode()` ist nicht sinnvoll, da dann in allen erbenden Klassen diese Operation realisiert werden muss.

[5] Dies ist so in der UML festgelegt.

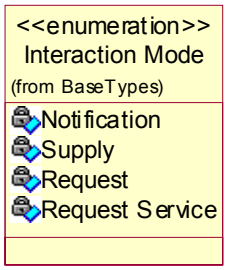


Abbildung 91: Die zulässigen Werte für das Prädikat `Interaction::Mode()`*, festgelegt durch den Aufzählungsdatentyp Interaction Mode*

Für den Interaktionsmodus dürfen folglich nur die Werte `Notification`, `Supply`, `Request` und `Request Service` verwendet werden. In allen anderen Fällen handelt es sich um kein wohldefiniertes Modell (vgl. Abschnitt 4.4). Auf diese Weise kann über das Metamodell festgestellt werden, ob ein Modell syntaktisch „richtig“ ist. Die vordefinierten Aufzählungsdatentypen finden sich in Abschnitt 11.16.

5.2.7 Beziehungen zwischen Skripten

Operationen werden im Skriptmodell mit Skripten (in der Rolle eines Subskripts) über Abhängigkeitsbeziehungen verknüpft. Dabei kann es sich um Verfeinerungs- oder Realisierungsbeziehungen handeln. Für die Verwendung der von *ABSTRACTION* erbenden Beziehungsknotentypen *REALIZATION* und *REFINEMENT* gilt: Ein Knoten des Metatyps *OPERATION* kann durch Beziehungsknoten der beiden Metatypen mit mehreren Knoten des Metatyps *SCRIPT* verbunden sein. Jeder Knoten des Metatyps *SCRIPT* kann über Beziehungsknoten der beiden Metatypen mit mehreren Knoten des Metatyps *OPERATION* verbunden sein. Das bedeutet, dass ein Skript eine Realisierung oder Verfeinerung mehrerer Operationen sein kann und dass eine Operation mehrere Realisierungen oder Verfeinerungen in Form von Skripten besitzen kann. Das folgende Modell stellt diese Verfeinerung des im Metamodells des Konzeptuellen Schemas postulierten Zusammenhangs dar:

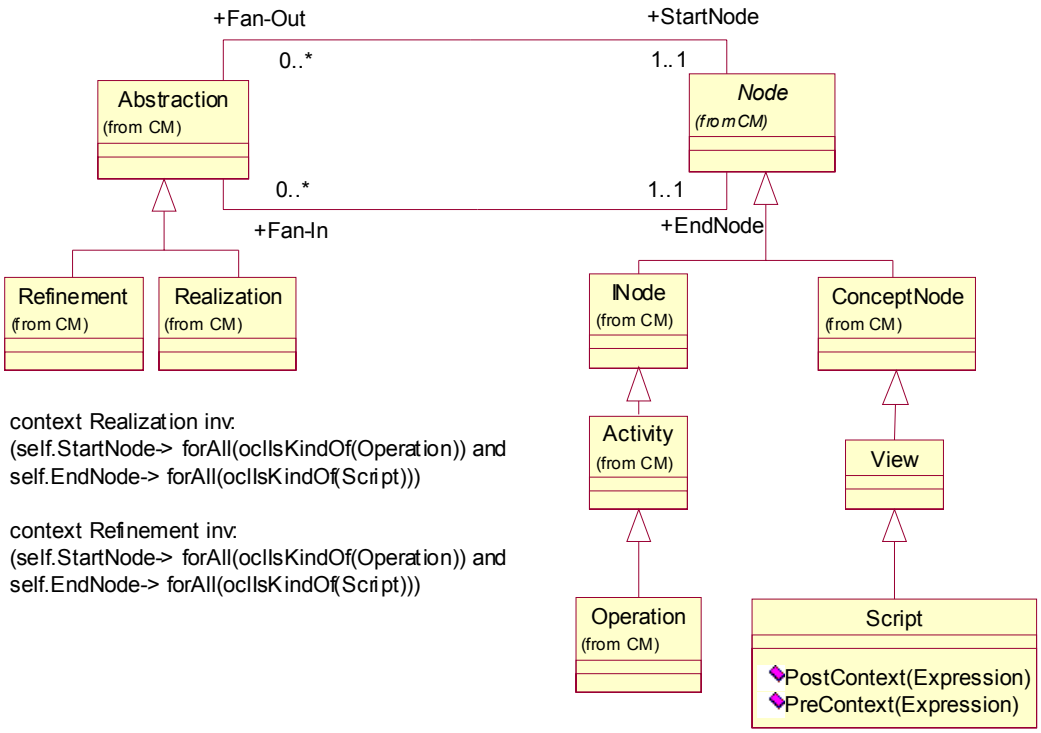


Abbildung 92: Metamodell für Subskripte als UML-Klassendiagramm

Durch dieses Metamodell wird festgelegt, dass im Skriptmodell nur Operationen durch Skripte realisiert oder verfeinert werden. In anderen Externen Sichten auf das Konzeptuelle Modell wäre es durchaus möglich, auch andere Knotentypen durch Abstraktionsbeziehungen zu verbinden.

Die Reihenfolgebeziehungen zwischen Skripten werden durch den Metatyp *EVENTFLOW* des Konzeptuellen Schemas abgebildet. Ein Skript kann keinen, einen oder mehrere Vorgänger und Nachfolger besitzen, die wiederum Skripte sind. Szenarien werden im Metamodell des Skriptmodells durch den Metatyp *SCENARIO* dargestellt, der ebenfalls von *VIEW* erbt. *SCENARIO* besitzt – wie die Klasse *SCRIPT* – einen Vor- und einen Nachkontext. Der Vorkontext beschreibt die Ausgangssituation des Szenarios, der Nachkontext beschreibt die Endsituation, die zum Endzeitpunkt des Szenarios gilt. Da das Szenario aus den Skripten besteht, gilt folgendes UML-Klassendiagramm:

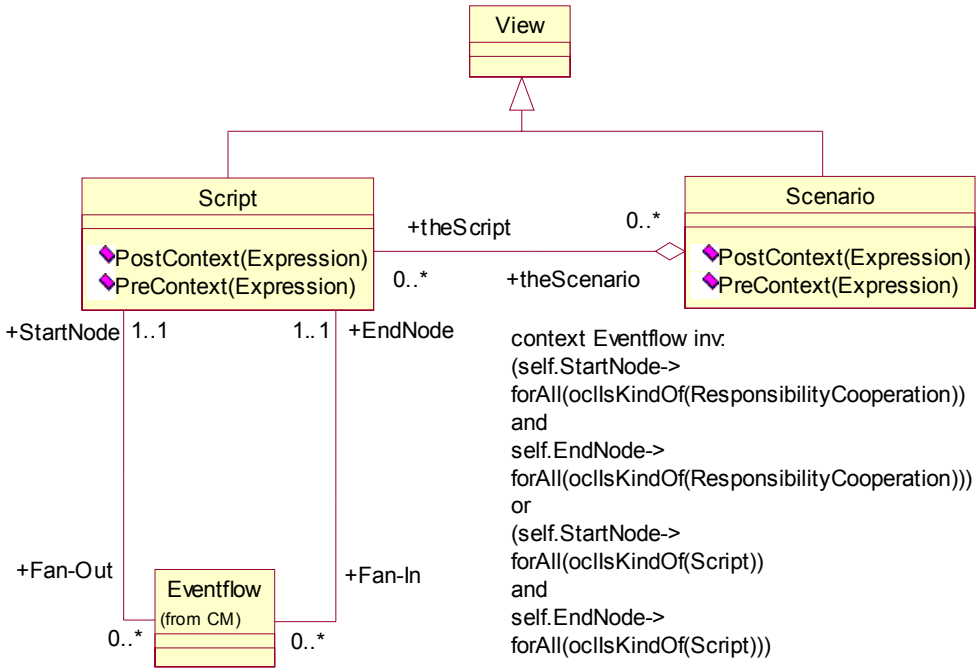


Abbildung 93: Zusammenhang zwischen Skripten und Szenarien als UML-Klassendiagramm

Aus dem Diagramm wird deutlich, dass folgende Modellierungsmöglichkeiten bestehen:

- Mehrere Skripte bilden ein Szenario.
- Ein Skript kann ein Szenario bilden.
- Ein Skript kann zu mehreren Szenarien gehören.
- Ein Skript muss zu keinem Szenario gehören.
- Es lassen sich leere Szenarien anlegen.

Für die praktische Anwendung bedeutet dies, dass unterschiedliche Versionen bzw. Alternativen eines Ablaufs aus einzelnen Skripten zusammengefügt werden können. Die einzelnen Skripte werden als Bausteine in unterschiedlichen Abläufen verwendet.

5.3 Verwendungsmöglichkeiten anderer Externer Schemata

Auch andere Diagramme können als Externes Schema verwendet werden. In diesem Abschnitt wird demonstriert, wie sie auf das Konzeptuelle Schema abgebildet werden. Die Darstellung ist auf

Beispiele beschränkt. Dadurch wird auch demonstriert, dass die Skripte die Diagramme anderer Methoden ersetzen können.

5.3.1 UML-Klassendiagrammen

Auch UML-Klassendiagramme können als Externes Schema verwendet werden. Es wird gezeigt, wie aus Skripten über die Abbildung auf das Konzeptuelle Schema die Schnittstellen von Klassen bzw. ein UML-Klassendiagramm erzeugt wird. Das folgende Beispiel geht von der Situation aus, dass UML-Klassendiagramme als weiteres Externes Schema definiert sind:

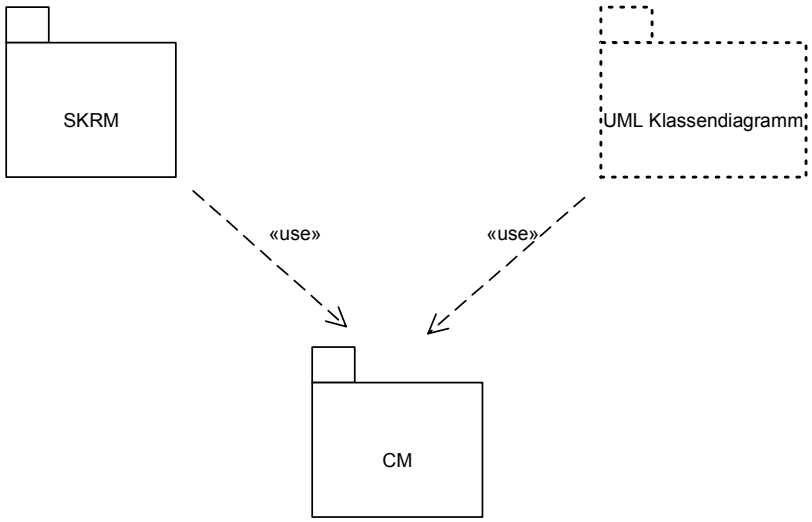


Abbildung 94: UML-Klassendiagramm als weiteres Externes Schema

Die folgenden drei Skripte stellen ein Beispiel für die Abbildung auf das Konzeptuelle Schema dar:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/xRole :A		:Association DomCard(1) RngCard(0..1)	/yRole :B	
1.	:A	Aktion1 :Action	:Interaction	:B	Op1 :Operation In(parA :Atyp) Vis(public)
	:B	Aktion3 :Action	:Internal	:B	Attr1 :Attribute Type(Integer)
2.	:A	Aktion2 :Action	:Interaction	:C	Op2 :Operation

Skript 14: Erstes Beispielskript

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:C		:Inheritance	:B	
	:D	Aktion2 :Action	:Interaction	:B	Op1 :Operation In(parA :Atyp) Vis(public)

Skript 15: Zweites Beispielskript

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:C		:Membership DomCard(1) RngCard(n)	:D	
	dasC :C	Aktion2 :Action	:Interaction Content("xxx" :char)	das D /u :D	Op1 :Operation In(Wert :char) Out(Ergebnis :Boolean)
	:D	Z1 :State	:StateChange Act(Op1 :Operation)	:D	Z2 :State

Skript 16: Drittes Beispielskript

Die Beispiele zeigen sehr gut, dass in den Skripten simultan sowohl die statische wie auch die dynamische Modellierung möglich ist: Es werden dynamische und statische Beziehungstypen verwendet. Nach dem in diesem Kapitel definierten Abbildungsprozess entsteht im Konzeptuellen Schema folgendes Semantische Netz:

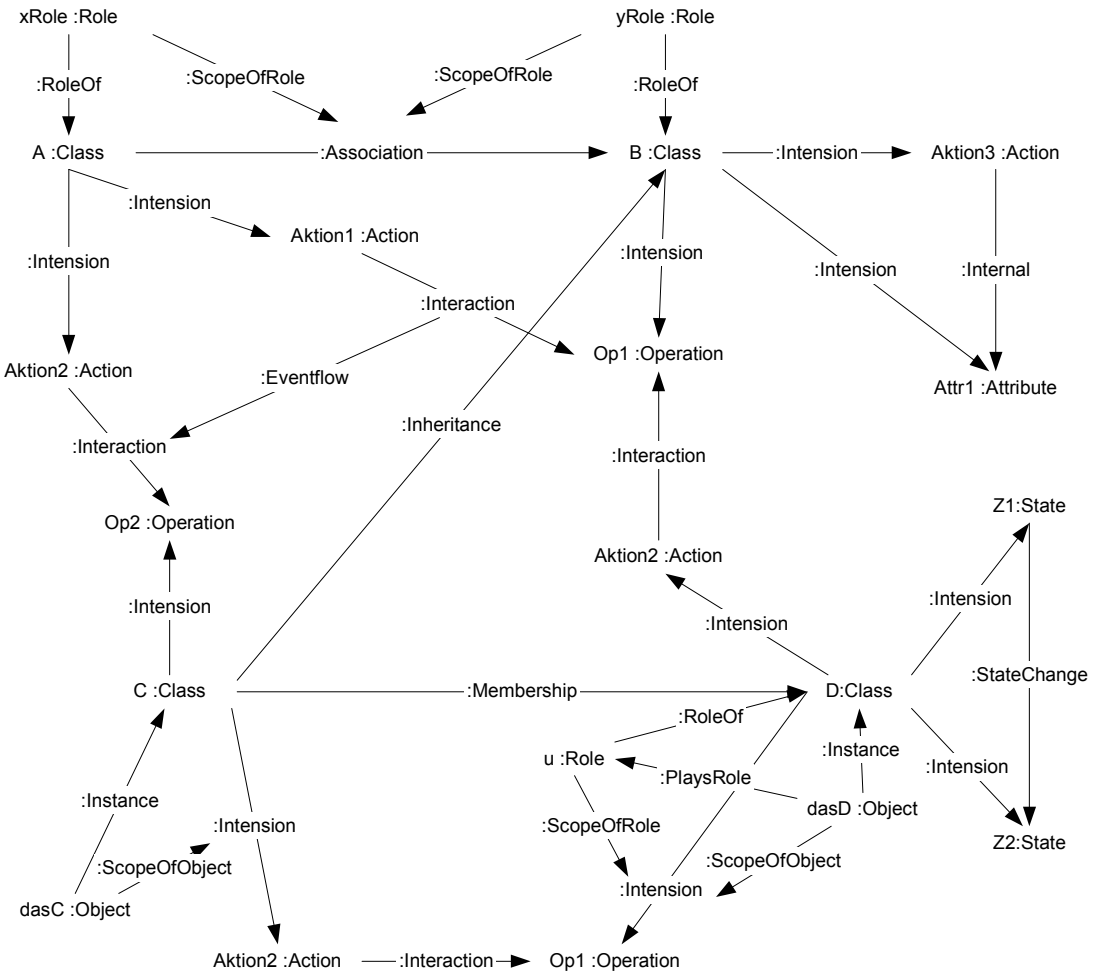


Abbildung 95: Resultierendes Konzeptuelles Schema

Aus diesem Konzeptuellen Schema lässt sich ein UML-Klassendiagramm als Externes Schema erzeugen. Im Paket UML-Klassendiagramm wird für jeden *CLASS*-Knoten des Konzeptuellen Schemas eine Klasse im Klassendiagramm angelegt. Zu jeder Klasse werden die zugehörigen *ATTRIBUTE*- und *OPERATION*-Knoten gesucht, indem man die vom *CLASS*-Knoten ausgehenden *INTENSION*-Beziehungsknoten der Klasse untersucht. Datentypen von Attributen, Argumente und die Sichtbarkeit von Operationen sind im Konzeptuellen Schema durch Prädikate abgespeichert. Die durch *ACTION*-Knoten dokumentierten Aktionen von Klassen werden im Klassendiagramm nicht übernommen. Vererbungsbeziehungen und Assoziationsbeziehungen (mit ihren Kardinalitäten) können direkt übernommen werden. Allerdings ist hierbei zu beachten, dass jede Verwendung einer referentiellen oder taxonomischen Beziehung in einem Skript im Klassendiagramm erscheint. Die zwangsläufig auftretenden Dubletten müssten in der Darstellung des

Klassendiagramms unterdrückt werden.[1] Beziehungstypen, die in der UML nicht existieren, müssen durch stereotypisierte UML-Beziehungen abgebildet werden. Auf diese Weise würde aus dem *MEMBERSHIP*-Beziehungsknoten des Konzeptuellen Schemas eine stereotypisierte UML-Aggregationsbeziehung entstehen. Rollen können nur für referentielle Beziehungen übernommen werden.

Die *INTERACTION*-Beziehungsknoten werden in UML-Klassendiagrammen als *DEPENDENCY*-Beziehungen mit dem Stereotyp «`call`» dargestellt. Dabei wird jedoch nicht mehr sichtbar, durch welche Aktion die Operation ausgelöst wurde. Die Zustandsveränderungen in Form von *STATECHANGE*-Beziehungsknoten werden ignoriert, da sie nicht im UML-Klassendiagramm darzustellen sind. Die durch *EVENTFLOW*-Beziehungsknoten abgebildeten Reihenfolgebeziehungen werden ebenfalls ignoriert, da in Klassendiagrammen keine Reihenfolgen existieren. *OBJECT*-Knoten sind im UML-Klassendiagramm ebenfalls nicht sichtbar. Auf diese Weise ergibt sich folgendes Diagramm:

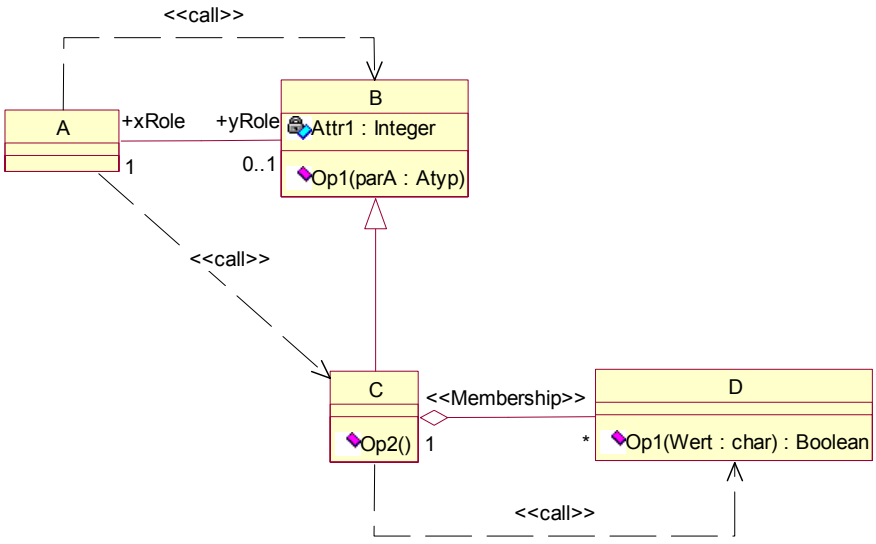


Abbildung 96: Resultierendes UML-Klassendiagramm

Da das Konzeptuelle Schema sowohl die dynamischen wie auch die statischen Aspekte der objektorientierten Modellierung beinhaltet, sind UML-Klassendiagramme eine Projektion der

[1] Dies ist ein Thema, welches die Realisierung der Methode durch ein Werkzeug betrifft. Hier soll auf solche Fragen der technischen Realisierung nicht weiter eingegangen werden.

insgesamt im Konzeptuellen Schema enthaltenen Informationen. Es wird aber deutlich, dass die bei der ooSe wichtigen Schnittstellen von Klassen sich direkt aus den Skripten herleiten lassen. Auf diese Weise ersetzen die UML-Klassendiagramme die Objektmodell-Karten der OBA.[1] Im Gegensatz zur OBA, die den Unterschied zwischen Klassen und ihrer Extension verwischt, existiert nun aber durch das Konzeptuelle Schema eine definierte Abbildung von Objekten und Rollen auf Klassen.

Durch den Umstand, dass *ROW*- und *SKRIPT*-Knoten kein Bestandteil des Konzeptuellen Schemas sind, sondern ihr Inhalt auf Klassen abgebildet wird, ergibt sich die Synthese bzw. die Integration[2] der Skripte durch das Semantische Netz des Konzeptuellen Schemas: Unabhängig davon, welche und wie viele Skripte zur Modellierung verwendet werden, ist die Existenz eines konsistenten Gesamtmodells gewährleistet.[3] Dies ermöglicht die Analyse der Abläufe in einem System durch eine beliebige Anzahl von Skripten und stellt einen erheblichen Vorteil gegenüber dem in der UML propagierten Verfahren dar, Abläufe und funktionales Verhalten durch Use Cases und Interaktionsdiagramme zu untersuchen,[4] denn dort fehlt eine solche Integration von Modellierungsergebnissen.

Die folgenden Beispiele demonstrieren, dass auch andere Diagramme direkt oder mit geringen Veränderungen in Skripte umgeformt werden können. Daraus ergeben sich zwei Folgerungen: Erstens können die in den Beispielen erwähnten Diagramme auch als Externes Schema verwendet werden, was die Leistungsfähigkeit des Konzeptuellen Schemas unterstreicht. Zweitens wird auf diese Weise deutlich, dass die Skripte die genannten Notationsformen ersetzen können, was die Leistungsfähigkeit des Skriptmodells unterstreicht.

5.3.2 SOM-Vorgangs-Ereignis-Schema

Die folgende Abbildung stellt die Abwicklung eines Geschäftsvorfalls in der Notation eines Vorgangs-Ereignis-Schemas (VES) bzw. -Diagramms der Methode SOM von FERSTL und SINZ dar.[5] Diese Notation wurde gewählt, weil sie die am meisten verbreitete objektorientierte Notation zur GPM darstellt.

---

[1] Vgl. Abschnitt 3.1.3, [Rubin 1992: S. 57 f.].

[2] Im Bereich der Softwareentwicklung spricht man auch von Integrationsmodell.

[3] In der UML ist die Synthese mehrerer Diagramme nicht definiert. Dem Autor sind nur zwei Methoden bekannt, die sich mit diesem Problem befassen: OOram [Reenskaug 1996] und OSA [Embley 1992].

[4] Vgl. die Kritik in [Linssen 1999b].

[5] Vgl. [Ferstl 1994], [Ferstl 1994c], [Ferstl 1995], [Ferstl 1998]. Vgl. auch Abschnitt 2.7.

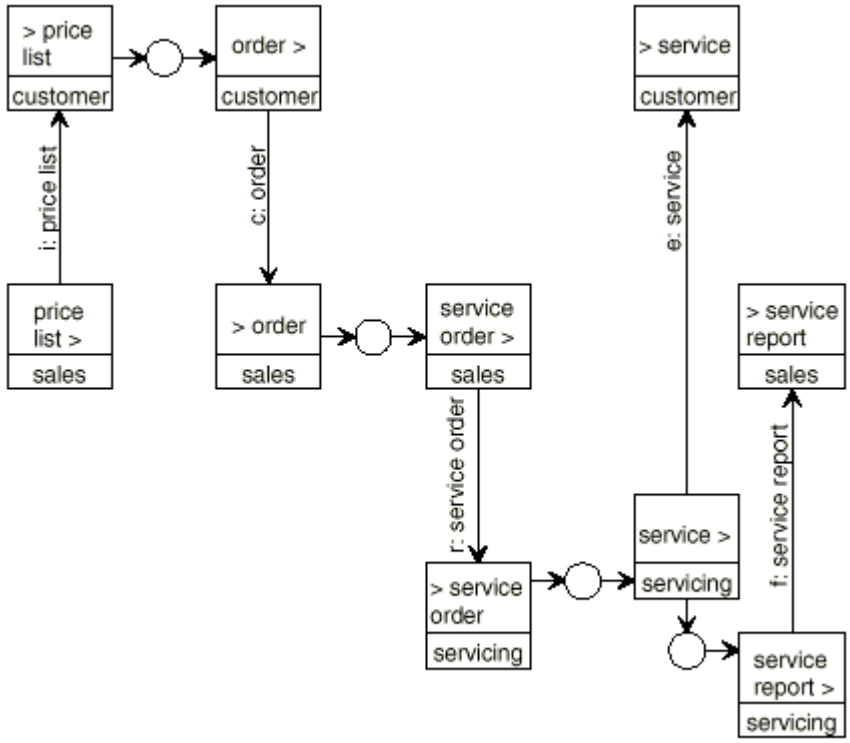


Abbildung 97: SOM-Vorgangs-Ereignis-Schema[1]

Die Rechtecke beinhalten im unteren Abschnitt den Namen des Objekts und im oberen Abschnitt die Bezeichnung der vom Objekt ausgeführten Aufgaben. Die senkrechten Pfeile stellen die zwischen Objekten ablaufenden Interaktionen dar, die waagerechten Pfeile stellen Reihenfolgebeziehungen dar. Die Buchstaben an den Interaktionsbeziehungen geben die Art der Koordination zwischen den beteiligten Objekten an. Das im Konzeptuellen Schema für den Metatyp *INTERACTION* definierte Prädikat `Coord()` wird verwendet, um diesen Koordinationstyp anzugeben.[2] Im Diagramm werden die Abkürzung i (*initiating*) für Anbahnung, c (*contracting*) für Vereinbarung, e (*service*) für Durchführung, r (*controlling*) für Steuerung und f (*feedback*) für Kontrolle verwendet.[3] Die hinter diesen Buchstaben zu sehende Textkette stellt den Inhalt der Interaktion dar. Dieser Inhalt wird durch das Prädikat `Content()` abgebildet. Zur besseren

[1] [Ferstl 1997: S. 14].

[2] Weitere Erläuterungen zu der Bedeutung dieser Prädikate finden sich in dem Kapitel über die Anwendung des Skriptmodells. Der Leser möge diesen Vorgriff entschuldigen, der aus methodischen erfolgte.

[3] [Ferstl 1994c: S. 8]. In der OBA++ werden allerdings andere Bezeichner verwendet, nämlich *request* für Anbahnung, *commitment* für Vereinbarung, *performance* für Durchführung, *control* für Steuerung und *feedback* für die Kontrolle.

Vergleichbarkeit wird die Bezeichnung redundant als Name der Interaktion übernommen.[1] Außerdem werden die Namen der Objekte und der Aufgaben übersetzt.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Verkauf	versendet Preisliste:Action	Preisliste :Interaction Coord(request) Content(:Preisliste)	:Kunde	erhält Preisliste :Operation
2.	:Kunde	bestellt:Action	Bestellung :Interaction Coord(commitment) Content(:Bestellung)	:Verkauf	erhält Bestellung :Operation
3.	:Verkauf	fordert Leistung an :Action	Leistungsauftrag :Interaction Coord(control) Content(:Leistungsauftrag)	:Leistungs-erbringer	erhält Leistungsauftrag :Operation
4.	:Kunden-dienst	erbringt Leistung :Action	Leistung :Interaction Coord(performance) Content(:Leistung)	:Kunde	erhält Leistung :Operation
5.	:Leistungs-erbringer	überträgt Bericht :Action	Bericht :Interaction Coord(feedback) Content(:Bericht)	:Verkauf	erhält Bericht :Operation

Skript 17: Skript zu SOM-Vorgangs-Ereignis-Schema

Offensichtlich ist eine Übertragung auf die hier verwendeten Skripte möglich. Das folgende Vorgangs-Ereignis-Schema stellt nach FERSTL und SINZ eine Verfeinerung bzw. eine Weiterentwicklung des vorherigen Diagramms dar:[2]

[1] Im weiteren Verlauf der Darstellung wird noch deutlich, warum.

[2] [Ferstl 1997: S. 15].

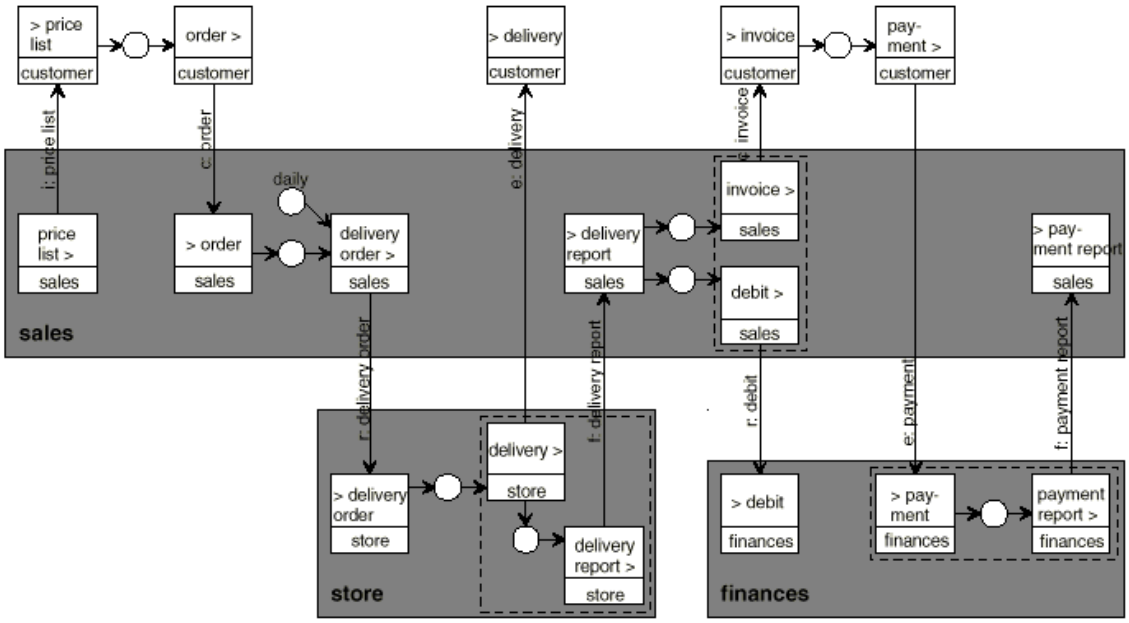


Abbildung 98: SOM-Vorgangs-Ereignis-Schema

Hier wurde das Objekt Leistungserbringer in die Objekte Lager (*store*) und Buchhaltung (*finances*) zerlegt. Außerdem führt das Objekt Verkauf (*sales*) nun zwei Aufgaben parallel aus:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Verkauf	versendet Preisliste:Action	Preisliste :Interaction Coord(request) Content(Preisliste)	:Kunde	erhält Preisliste :Operation
2.	:Kunde	bestellt:Action	Bestellung :Interaction Coord(commitment) Content(Bestellung)	:Verkauf	erhält Bestellung :Operation
	:Leistungs-erbinger		:Aggregation	:Lager	
	:Leistungs-erbringer		:Aggregation	:Buchhaltung	
5.	:Verkauf	verschickt Lieferorder :Action periodic (Int= 1 day)	Lieferorder :Interaction Coord(control) Content(Lieferorder)	:Lager	erhält Lieferorder :Operation
6.	:Lager	liefert :Action	Lieferung :Interaction Coord(performance) Content(Lieferung	:Kunde	erhält Lieferung :Operation
7.	:Lager	verschickt Lieferbericht :Action	Lieferbericht :Interaction Coord(feedback) Content(Lieferbericht)	:Verkauf	erhält Lieferbericht :Operation

8.	:Verkauf	versendet Rechnung :Action	Rechnung :Interaction Coord(commitment) Content(Rechnung)	:Kunde	erhält Rechnung:Operation
9.	:Verkauf	verschickt Rechnungskopie :Action	Rechnungskopie :Interaction Coord(control) Content(Rechnungskopie)	:Buchhaltung	belastet Kundenkonto:Operation
		Rechnung :Interaction	:Eventflow TimingConstraint(Rechnungskopie. sendTime() – Rechnung.sendTime = 0)		Rechnungskopie :Interaction
11.	:Kunde	bezahlt Rechnung :Action	Bezahlung :Interaction Coord(performance) Content(Geldbetrag)	:Buchhaltung	erhält Bezahlung :Operation
12.	:Buchhaltung	vermeldet Zahlungseingang :Action	Zahlungseingang :Interaction Coord(feedback) Content(Zahlungsbeleg)	:Verkauf	erhält Bericht :Operation

Skript 18: Skript zu SOM-Vorgangs-Ereignis-Schema mit Parallelität

Auch dieses Skript zeigt wieder, wie gleichzeitig dynamische und statische Aspekte abgebildet werden können. Die Transaktionstypen der Methode SOM wurden wieder durch das Prädikat `Coord()` übernommen und als Art der Koordination zwischen den beteiligten Objekten verwendet. Durch die beiden Aggregationsbeziehungen wird festgehalten, dass der Leistungserbringer aus dem Lager und der Buchhaltung besteht. Im VES ist diese Beziehung nicht sichtbar. Die Zeile zwischen (9) und (11) setzt die Zeilen (8) und (9) in eine zeitliche Beziehung: Zwischen den Interaktionen (8) und (9) soll die zeitliche Beziehung gelten, dass sie parallel gestartet werden. Die Tatsache, dass die Interaktion Lieferorder einmal täglich stattfindet, wird in der Interaktion 5 durch das Prädikat `periodic(Int = 1 day)` abgebildet.[1]

Auf der Basis von Skript 18 soll noch einmal die Erzeugung des Semantischen Netzes im Konzeptuellen Schema demonstriert werden. Für die im Skript verwandten Objekte Verkauf, Kunde, Lager, Buchhaltung und Leistungserbringer werden im Konzeptuellen Schema die entsprechenden Knoten des Metatyps *CLASS* erzeugt:

[1] Weitere Erläuterungen folgen im Kapitel über die Anwendung des Skriptmodells.

Verkauf:Class

Kunde:Class

Leistungserbringer:Class

Buchhaltung:Class

Lager:Class

Abbildung 99: Erzeugte Klassen und Objekte im Semantischen Netz

Die Einträge in der Tabellenspalte Initiator Responsibility und Participant Responsibility werden zerlegt, wobei der Doppelpunkt als Trennsymbol verwendet wird. Die hinter dem Doppelpunkt stehenden Bezeichner (`Action` und `Operation`) werden verwendet, um die entsprechenden Knotentypen im Konzeptuellen Schema zu erzeugen. Der vor dem Doppelpunkt zu findende Bezeichner wird als Name des Knotens verwendet. Diese Intensionsknoten werden durch einen Beziehungsknoten des Metatyps *INTENSION* mit dem zugehörigen Klassenknoten verbunden:

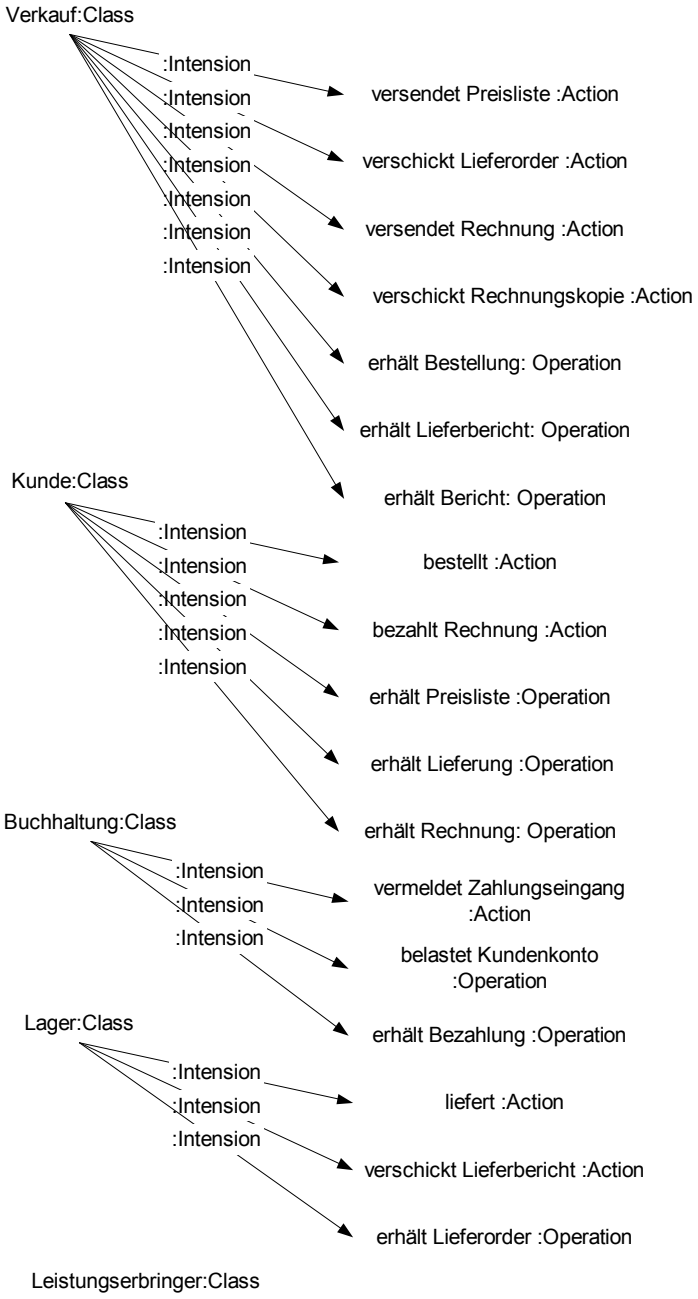


Abbildung 100: Erzeugte Intension und Extension der Klassen im Semantischen Netz

Die Einträge in der Tabellenspalte Connection wird in ähnlicher Weise zerlegt. Der hinter dem Doppelpunkt stehende Bezeichner bestimmt den Typ des Knotens im Konzeptuellen Schema. Zunächst werden die Beziehungsknoten des Typs *INTERACTION* erzeugt. Der vor dem Doppelpunkt stehende Bezeichner wird als Name des Knotens verwendet. Ausgangspunkt der Kante ist der Knoten aus der jeweiligen Tabellenspalte Initiator Responsibility. Endpunkt der Kante ist der jeweilige Eintrag aus der Tabellenspalte Participant Responsibility. Die Transakti-

onstypen der Methode SOM werden als Prädikat `Coord()` der Objekte vom Typ *INTERACTION* abgespeichert, welches im Metamodell des Konzeptuellen Schemas für die Klasse *INTERACTION* definiert wurde. Da diese Prädikate keine eigenständigen Knotentypen darstellen, werden sie in den Abbildungen nicht dargestellt.

Für die Aggregationsbeziehungen zwischen den Zeilen 2 und 5 werden im Konzeptuellen Schema Beziehungsknoten des Typs *AGGREGATION* erzeugt. Die Aggregationsbeziehung wird zwischen den Klassen geknüpft, die in den Tabellenspalten Initiator und Participant zu finden sind, wobei die Klasse in der Tabellenspalte Initiator Ausgangspunkt der Beziehung ist. Die entsprechende Zeilennummer des Skripts wird aus Gründen der Übersichtlichkeit in die Grafik übernommen. Sie ist aber nicht Bestandteil des Konzeptuellen Schemas.

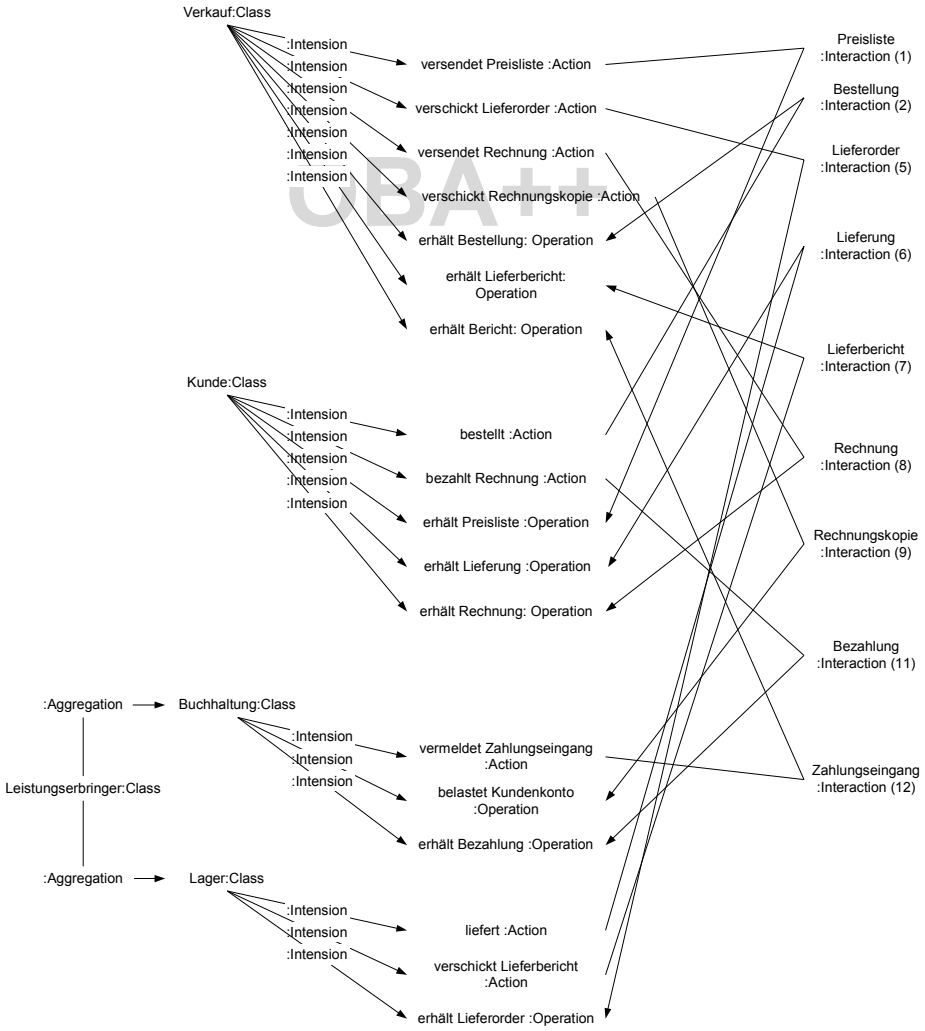


Abbildung 101: Erzeugte Aggregationsbeziehungen im Semantischen Netz

Durch die Zeilenummern werden die Zeilen des Skripts in Reihenfolgebeziehungen gesetzt. Diese Zeilennummern werden im Konzeptuellen Schema durch Abhängigkeitsbeziehungen des Typs *EVENTFLOW* abgebildet. Diese Beziehungsknoten verbinden die entsprechenden Beziehungsknoten des Typs *INTERACTION*. Für die zeitliche Abhängigkeitsbeziehung zwischen Zeile 9 und 11 wird ein weiterer Beziehungsknoten des Typs *EVENTFLOW* angelegt, der die Interaktionen `Rechnung` und `Rechnungskopie` verbindet. Die folgende Abbildung zeigt das Ergebnis des Abbildungsprozesses:

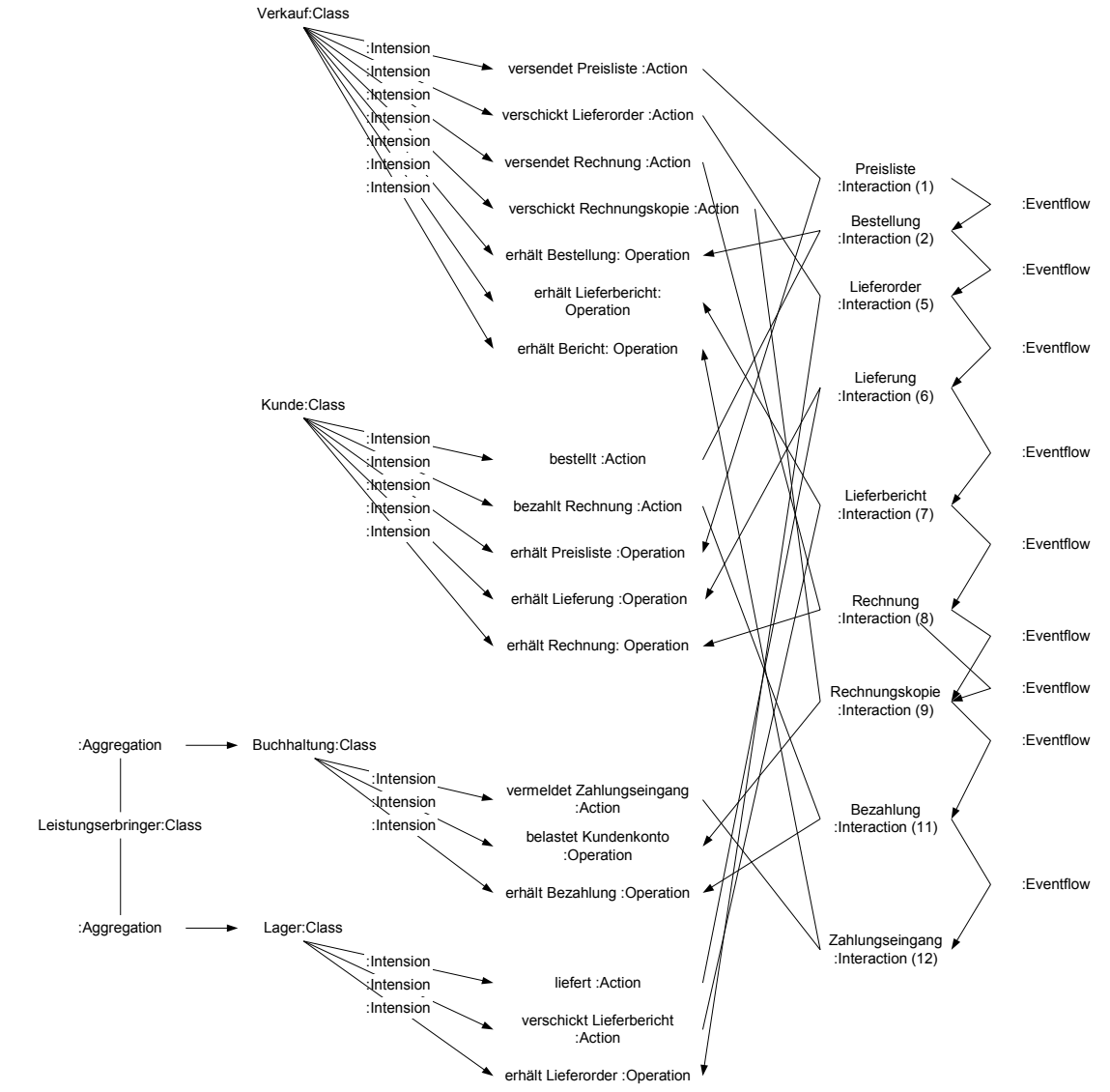


Abbildung 102: Endergebnis des Semantischen Netzes

Auch die Interaktionsdiagramme der UML können durch Skripte ersetzt werden. Dies wird an drei Beispielen aus dem OMG-Standard demonstriert.[1]

5.3.3 UML-Sequenzdiagramm

Das folgende UML-Sequenzdiagramm stellt einen Buchungsvorgang dar, an dem ein externer Benutzer, ein Kiosksystem, ein Datenbankserver und eine Kreditkartengesellschaft beteiligt sind. Das Beispiel ist im *Unified Modeling Language Reference Manual* zu finden.[2]

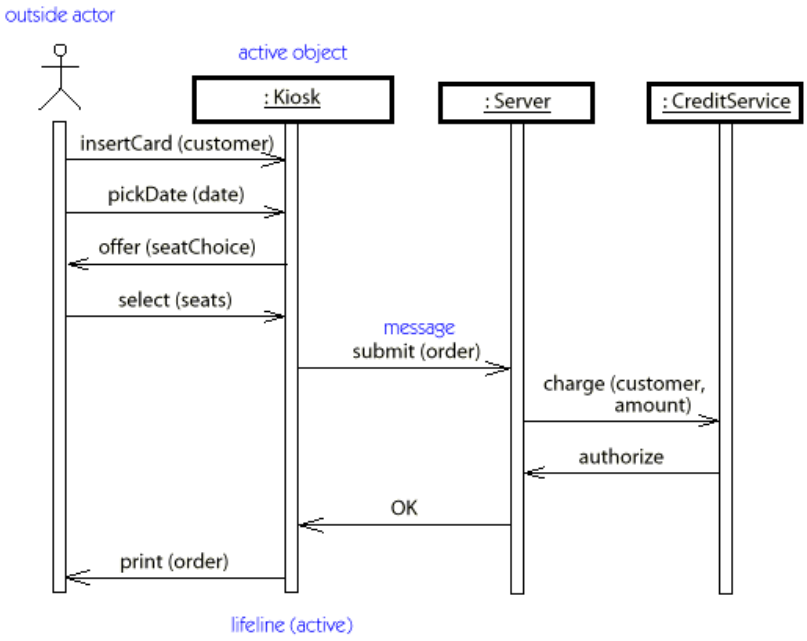


Abbildung 103: Buchungsvorgang als UML-Sequenzdiagramm

In der UML werden die in Sequenzdiagrammen verwendeten Namen der Nachrichten mit den Operationen des Empfängerobjekts gleichgesetzt.[3] Das Konstrukt der Aktion eines Senderobjekts ist in der UML unbekannt. Die vertikale Anordnung der Nachrichten zeigt eine von möglicherweise mehreren zulässigen Reihenfolgen auf.

[1] Vgl. [OMG 1999].

[2] [Rumbaugh 1999: S. 87].

[3] Vgl. [OMG 1999: S. 3-96].

Die vier anonymen Objekte der Klassen Benutzer, Kiosk, Server und Kreditkartengesellschaft können direkt in die Tabellenspalten Initiator und Participant übernommen werden. Die Nachrichten zwischen den Objekten werden als Operation des Participants der Nachricht abgebildet. Die angegebenen formalen Argumente werden durch das Prädikat `In()` abgebildet. Die Aktionen des Initiators erhalten keine Namen. Da die Abfolge der Interaktionen keine feste Reihenfolge darstellt, werden keine Sequenznummern verwendet. Weil in der UML die in den Sequenzdiagrammen verwendeten Namen die Nachrichten identifizieren sollen, werden sie zusätzlich redundant als Name der Interaktion übernommen. Daraus ergibt sich das folgende Skript:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:outside actor	:Action	insertCard :Interaction	:Kiosk	insertCard :Operation In(:customer)
	:outside actor	:Action	pick date :Interaction	:Kiosk	pick date :Operation In(:date)
	:Kiosk	:Action	offer :Interaction	:outside actor	offer :Operation In(:seatChoice)
	:outside actor	:Action	select :Interaction	:Kiosk	select :Operation In(:seats)
	:Kiosk	:Action	submit :Interaction	:Server	submit :Operation In(:order)
	:Server	:Action	charge :Interaction	:CreditService	charge :Operation In(:customer, :amount)
	:CreditService	:Action	authorize :Interaction	:Server	authorize :Action
	:Server	:Action	OK :Interaction	:Kiosk	OK :Operation
	:Server	:Action	print :Interaction	:outside actor	print :Operation In(:order)

Skript 19: Skript zu einem UML-Sequenzdiagramm[1]

5.3.4 UML-Sequenzdiagramm mit Zeitrestriktionen

In der UML werden Sequenzdiagramme auch verwendet, um zeitliche Abläufe, Abhängigkeiten zwischen Nachrichten und den Zeitverbrauch von Nachrichten abzubilden. Das folgende Beispiel ist ebenfalls im *Unified Modeling Language Reference Manual* zu finden.[1]

[1] Die Darstellung als Skript offenbart eine Schwäche in der UML. Die Nachricht `offer` wird nicht vom Benutzer ausgeführt, sondern vom Kiosksystem. Der Benutzer wählt daraufhin einen entsprechenden Platz aus. Eine ähnliche Situation entsteht durch die letzte Nachricht `print` im Sequenzdiagramm. Nicht der Benutzer druckt den Auftragsbeleg, sondern das Kiosksystem. Die hier notwendige Änderung müsste manuell vorgenommen werden.

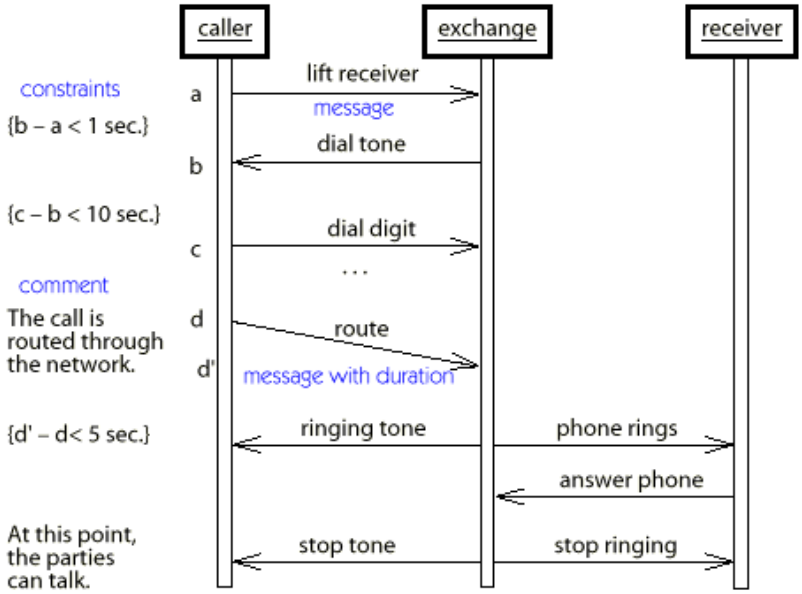


Abbildung 104: UML-Sequenzdiagramm mit Zeitbedingungen

Die am Rand des Sequenzdiagramms in Klammern stehenden Angaben werden in der UML als *timing mark* bezeichnet und dienen zur Angabe von zeitlichen Einschränkungen. Die ersten vier Nachrichten haben zusätzlich symbolische Namen (eine sog. *Ereignismarke*) erhalten (in der Grafik mit a, b, c, d und d‘ angegeben), um die Interaktionen in den zeitlichen Einschränkungen referenzieren zu können. Ihre Übernahme ist hier unnötig, da die Nachrichten über ihren Namen referenziert werden.[2]

[1] [Rumbaugh 1999: S. 424].

[2] Außerdem wird in diesem Diagramm ein asynchroner Kontrollfluss dargestellt. Das bedeutet, dass die Objekte nach dem Versand einer Nachricht nicht warten, bis der Empfänger die Nachricht verarbeitet hat. Dies wird in der Skriptnotation durch ein Prädikat `Concurrency()` abgebildet, auf das im Kapitel über die Anwendung des Skriptmodells eingegangen wird. Es ist hier nicht verwendet worden.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:caller	:Action	lift receiver :Interaction	:exchange	lift receiver :Operation
	:exchange	:Action	dial tone :Interaction	:caller	dial tone :Operation
		lift receiver :Interaction	:Eventflow TimingConstraint (dial tone.sendTime() – lift receiver.receiveTime() < 1 “)	dial tone :Interaction	
	:caller	:Action	dial digit :Interaction	:exchange	dial digit :Operation
		dial tone :Interaction	:Eventflow TimingConstraint(dial digit.sendTime() – dial tone.receiveTime() < 10 “)		dial digit :Interaction
	:caller	:Action	route :Interaction TimingConstraint(route.receiveTime() – route.sendTime() < 5 ”)	:exchange	route :Operation
	:exchange	:Action	ringing tone :Interaction	:caller	ringing tone :Operation
	:exchange	:Action	phone rings :Interaction	:receiver	phone rings :Operation
		phone rings :Interaction	:Eventflow TimingConstraint(phone rings.sendTime() = ringing tone.sendTime() = phone rings.receiveTime() = ringing tone.receiveTime()		ringing tone :Interaction
	:receiver	:Action	answer phone :Interaction	:exchange	answer phone :Operation
	:exchange	:Action	stop tone :Interaction	:caller	stop tone :Operation
	:exchange	:Action	stop ringing :Interaction	:receiver	stop ringing :Operation
		stop ringing :Interaction	:Eventflow TimingConstraint(stop ringing.sendTime() = stop tone.sendTime() = stop ringing.receiveTime() = stop tone.receiveTime()		stop tone :Interaction

Skript 20: Skript zu einem UML-Sequenzdiagramm mit Zeitbedingungen

Aus dem UML-Sequenzdiagramm wird nicht ersichtlich, durch welche Aktivität beim Sender der Nachricht die Nachricht übertragen wird. Aus diesem Grund haben die Initiatoren Aktionen ohne Namen erhalten. Die zeitlichen Restriktionen im UML-Sequenzdiagramm stellen im Skript Abhängigkeitsbeziehungen zwischen Interaktionen dar. Durch zwei der vier Eventflow-Beziehungen wird der Umstand abgebildet, dass jeweils zwei Interaktionen sowohl gleichzeitig beginnen und enden.

5.3.5 UML-Kollaborationsdiagramm

Bei Kollaborationsdiagrammen wird die Reihenfolge des Nachrichtenaustauschs durch Sequenznummern abgebildet. Auch sie lassen sich in Skripte übertragen. Das folgende

Kollaborationsdiagramm stellt einen Buchungsvorgang dar, an dem ein nicht weiter benannter Benutzer, ein Objekt zur Auftragsannahme, ein Datenbankserver und eine Kreditbüro beteiligt sind. Das Beispiel ist im *Unified Modeling Language Reference Manual* zu finden.[1]

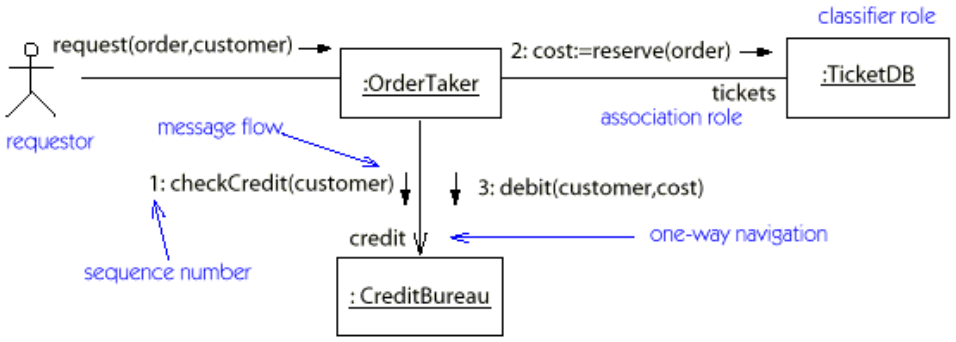


Abbildung 105: Buchungsvorgang als UML-Kollaborationsdiagramm

UML-Kollaborationsdiagramme sind äquivalent zu UML-Sequenzdiagrammen. Die Abbildung als Skript erfolgt also analog. Die in Kollaborationsdiagrammen zusätzlich verwendeten Rollen (hier: tickets, credit) können direkt als Rollennamen im Skript übernommen werden. In der UML erhält die Interaktion des Akteurs mit dem System keine Sequenznummer. Aus diesem Grund sind die Sequenznummern im Skript um 1 zu inkrementieren. Die formalen Argumente der Operationen des UML-Kollaborationsdiagramms sind durch das Prädikat `In()` abgebildet, die Rückgabewerte durch das Prädikat `Out()`. Das zugehörige Skript hat folgende Form:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Actor	:Action	request :Interaction	:OrderTaker	request :Operation In(:order, :customer)
2.	:OrderTaker	:Action	checkCredit :Interaction	/credit :CreditBureau	checkCredit :Operation In(:customer)
3.	:OrderTaker	:Action	reserve :Interaction	/tickets :TicketDB	reserve :Operation In(:order) Out(:cost)
4.	:OrderTaker	:Action	debit :Interaction	/credit :CreditBereau	debit :Operation In(:customer, :cost)

Skript 21: Skript zu einem UML-Kollaborationsdiagramm

[1] [Rumbaugh 1999: S. 89].

5.3.6 Skripte der OBA

Auch die Skripte der OBA von RUBIN und GOLDBERG können in der hier erweiterten Form abgebildet werden. Ihr informeller Charakter macht allerdings eine Reihe von Umformungen notwendig, um dem hier höheren Grad der Formalisierung der OBA++ zu genügen. Dies wird an folgendem Beispiel demonstriert:

Initiator	Action	Participant	Service
User	select D1	Spreadsheet	selects a cell
User	type NEW	D1	Cell sets content to text
User	set text style bold	D1	Cell sets style to bold
User	select A2	Spreadsheet	selects a cell
User	type NAME	A2	Cell sets content to text
	(repeated select and type from example)	*B2, C2, D2, A3:A10*	
User	select Row 2	Spreadsheet	selects a row
User	set text style bold	Row 2	Row sets style to bold
User	extend row height to 34 pixels	Row 2	Row resizes height
User	select A12	Spreadsheet	selects a cell
User	type TOTALS	A12	Cell sets content to text
	(repeat select and type)	*A3, A14*	
User	selects A12:A14	Spreadsheet	selects vertical set of cells
User	set text style bold	A12:A14	Vertical cell collection sets style of its cells to bold
User	select B3	Spreadsheet	selects a cell
User	type 55000	B3	Cell sets content to number
User	choose $xx,xxx	B3	Cell sets format to currency
User	select B3:B10	Spreadsheet	selects vertical set of cells
User	fill down	B3:B10	Vertical cell collection copies first cell into rest of its cells
User	select B4	Spreadsheet	selects a cell
User	replace 55 by 60	B4	Cell sets content to text
	(repeat # changes)	*B5:B10*	

Skript 22: Originalskript in der Notation der OBA[1]

In den Skripten der OBA werden nur Interaktionsbeziehungen abgebildet. Deshalb kann in der Tabellenspalte Connection automatisch der Beziehungstyp Interaction verwendet werden. Entsprechend kann der Inhalt der Tabellenspalte Action in der Tabellenspalte Initiator Responsi-

[1] Das Skript stammt aus den Schulungsunterlagen der Fa. PARCPLACE. [ParcPlace 1992]. (Textauszeichnungen stammen vom Verfasser.) Die kursiv gesetzten Zeilen stellen Kommentare dar, die sich wiederholende Interaktionen zusammenfassen. Sie werden nicht übernommen.

bility als Action und der Inhalt der Tabellenspalte Service in der Tabellenspalte Participant Responsibility als Operation abgebildet werden. Da in der OBA nicht zwischen Objekten, Klassen und Rollen differenziert wird, müssen die Tabellenspalten Initiator und Participant abhängig von ihrem Inhalt abgebildet werden. Wird eine bestimmte OBA-Rolle – wie die Zelle B3 oder die Zeile 3 – angesprochen, wird dies als benanntes Objekt übernommen. Die Zuordnung zu einer Klasse ergibt sich aus dem Zusammenhang. Die OBA-Rolle `Spreadsheet` wird als anonymes Objekt übernommen.

In der OBA wird der Inhalt der Interaktion, also beispielsweise die Eingabe des Benutzers, als Teil der Aktion formuliert. Dieser Inhalt wird nun als Prädikat `Content()` der Interaktion abgebildet. Zu diesen Inhalten werden entsprechende Prädikate In() der Operationen des Participants formuliert. Im Originalskript finden sich zwei Interaktionen, die den eher informellen Charakter der OBA verdeutlichen, weil hier in der Tabellenspalte Participant mehrere Rollen angegeben werden.[1] Geht man davon aus, dass damit eine Art 1-zu-N-Interaktion ausgedrückt werden soll, stellt das im hier verwendeten Modellierungsansatz eine Multicast-Interaktion dar, die durch das Prädikat `Prop()` abgebildet wird.[2] Das Prädikat erhält als Term die Ausbreitungsart und die Menge der Empfängerobjekte. Hierauf wird noch im nächsten Kapitel eingegangen.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:User	select cell :Action	:Interaction Content(D1:cell)	:Spreadsheet	select a cell :Operation In(:cell)
	:User	type text :Action	:Interaction Content(NEW)	D1 :Cell	set content to text :Operation In(:text)
	:User	set text style :Action	:Interaction Content(bold)	D1 :Cell	set style :Operation In(:style)
	:User	select cell :Action	:Interaction Content(A2 :cell)	:Spreadsheet	select a cell :Operation In(:cell)
	:User	type text :Action	:Interaction Content(NAME)	A2 :Cell	set content to text :Operation In(:text)
	:User	select Row :Action	:Interaction Content(Row 2 :Row)	:Spreadsheet	select a row :Operation In(:row)
	:User	set text style :Action	:Interaction Content(bold)	Row 2 :Row	set style :Operation In(:style)
	:User	extend row height :Action	:Interaction Content(34 pixels)	Row 2 :Row	resize height :Operation In(:height)
	:User	select cell :Action	:Interaction Content(A12 :cell)	:Spreadsheet	select a cell :Operation In(:cell)
	:User	type text :Action	:Interaction Content (TOTALS)	A12 :Cell	set content to text :Operation In(:text)
	:User	selects set of cells	:Interaction	:Spreadsheet	select vertical set of cells

[1] Sie sind im Originalskript unterstrichen ausgezeichnet.

[2] Diese Multicast-Interaktion würde prozedural durch eine Iteration abgebildet werden. Die entsprechenden Prädikate wurden wieder unterstrichen.

		:Action	Content(A12 :cell, A14 :cell)		:Operation In(begin :cell, end :cell)
	:User	set text style :Action	:Interaction Content(bold) Prop(Multicast, A12 - A14)	:Cell	set style :Operation In(:style)
	:User	select cell :Action	:Interaction Content(B3 :cell)	:Spreadsheet	select a cell :Operation In(:cell)
	:User	type numbers :Action	:Interaction Content(55000)	B3 :Cell	set content to number :Operation In(:number)
	:User	choose format :Action	:Interaction Content($xx,xxx)	B3 :Cell	set format to currency :Operation In(:format)
	:User	select set of cells :Action	:Interaction Content(B3 :cell, B10 :cell)	:Spreadsheet	select vertical set of cells :Operation In(begin :cell, end :cell)
	:User	fill down :Action	:Interaction Content(B3 :cell) Prop(Multicast, B3 - B10)	:Cell	copy cell :Operation In(:cell)
	:User	select cell :Action	:Interaction Content(B4 : cell)	:Spreadsheet	select a cell :Operation In(:cell)
	:User	replace content :Action	:Interaction Content(60)	B4 :Cell	set content to text :Operation In(:text)

Skript 23: Umformuliertes Skript

5.4 Zusammenfassung

Während das Metamodell des Konzeptuellen Schemas Inhalt des letzten Kapitels war, ist der Gegenstand dieses Kapitels das Metamodell der äußeren Repräsentation und die Abbildung auf das Konzeptuelle Schema. Die äußere Repräsentation des Modellierungsansatzes sind Skripte. Skripte sind Tabellen mit sechs Tabellenspalten. Darüber hinaus wird festgelegt, wie der Inhalt der Skripte auf das Konzeptuelle Schema abzubilden ist. Das Konzeptuelle Schema dient auch zur Synthese mehrerer Skripte. Die Spezifikation der Tabellenspalten der Tabelle besteht aus einem oder mehreren Prädikaten. Das Metamodell beinhaltet auch Beziehungen zwischen Skripten, um komplexe Modelle aus Skripten zusammensetzen zu können.

Auch andere Diagramme können auf das Konzeptuelle Schema abgebildet werden. Dies wurde am Beispiel von UML-Klassendiagrammen, SOM-Vorgangs-Ereignis-Schemata, UML-Sequenz- und -Kollaborationsdiagrammen und OBA-Skripten demonstriert. Das nächste Kapitel erläutert die in diesem Kapitel definierten Prädikate und demonstriert die Verwendung der Skripte im Rahmen der objektorientierten Modellierung.

6 Die Anwendung des Skriptmodells

Nachdem im letzten Kapitel der Aufbau der Skripte durch ein Metamodell festgelegt wurde, behandelt dieses Kapitel die objektorientierte Modellierung mit Skripten. Skripte werden primär verwendet, um Prozesse zu modellieren. Dabei wird die schon eingeführte tabellarische Darstellung mit sechs Tabellenspalten und beliebig vielen Zeilen verwendet. Für den Inhalt der Skripte gilt folgende Faustformel:[1]

- In der zweiten und fünften Tabellenspalte (*Initiator* und *Participant*) wird angegeben, wer oder was an einer Beziehung beteiligt ist.
- In der dritten und sechsten Tabellenspalte (*Initiator Repsonsibility* und *Participant Responsibility*) wird angegeben, welche Aktivitäten die Beteiligten ausführen.
- In der vierten Tabellenspalte (*Connection*) wird angegeben, in welcher Art von Beziehung die Beteiligten stehen.
- In der ersten Tabellenspalte können in einer speziellen Notation Angaben zur Reihenfolge von Ereignissen gemacht werden.

6.1 Initiator und Participant

6.1.1 Objekte, Rollen und Klassen

Der Aufbau der Tabellenspalten Initiator und Participant entspricht den *Collaboration Roles* der UML.[2] In der folgenden Tabelle steht `C` für Klasse, `O` für Objekt und `R` für Rolle. Folgende Kombinationen sind möglich:

[1] Die Einzelheiten sind im Metamodell des Skriptmodells festgelegt.

[2] [OMG 1999: S. 3-110], [OMG 2001: S. 3-131].

Alternative	Syntax	Erklärung
1.	:C	unbenanntes Objekt der Klasse C
2.	/R :C	unbenanntes Objekt der Klasse C, welches die Rolle R übernimmt
3.	O :C	Objekt O der Klasse C
4.	O /R :C	Objekt der Klasse C, welches die Rolle R übernimmt
5.	C	eine Klasse C

Tabelle 8: Mögliche Kombinationen für die Tabellenspalten Initiator und Participant

Das folgende Skript stellt dar, dass ein unbenanntes Objekt – im Folgenden auch als *anonymes* Objekt bezeichnet – der Klasse `Verkäufer` und ein unbenanntes Objekt der Klasse `Kunde` in einer Beziehung zueinander stehen. Der Verkäufer ist der Ausgangspunkt der Beziehung. Er ist in der Tabellenspalte Initiator eingetragen. Der Kunde ist der Endpunkt der Beziehung. Er ist in der Tabellenspalte Participant eingetragen. Welcher Art die Beziehung zwischen Initiator und Participant ist, wird durch die Tabellenspalte Connection festgelegt, wird hier aber zunächst noch nicht beachtet. Ebenfalls weggelassen wurden die Aktivitäten der beiden Objekte, die – je nach Art der Beziehung – in den Tabellenspalten Initiator Responsibility und Participant Responsibility eingetragen werden.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Verkäufer	...	...	:Person	...

Skript 24: Verwendung von Klassennamen

Dabei ist es möglich, dass eine Beziehung zwischen Objekten der gleichen Klasse existiert. Der Beziehungstyp ist wieder weggelassen worden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Verkäufer	...	...	:Verkäufer	...

Skript 25: Klassennamen können identisch sein

Wenn referentielle und taxonomische Beziehungen (Metaklasse *RELATIONSHIP* im Konzeptuellen Schema) verwendet werden, kann der Klassenname ohne das Präfix „:“ verwendet werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Verkäufer	...	...	Kunde	...

Skript 26: Verwendung von referentiellen und taxonomischen Beziehungen

Die Verwendung von Rollen ist beispielsweise sinnvoll, wenn Objekte einer Klasse in einer bestimmten Rolle verwendet werden. Dies ist insbesondere hilfreich, wenn die Interaktion vom Lieferantenobjekt ausgeht und der Kunde als Reaktion eine Leistung abnimmt, also in der Tabellenspalte Participant steht. Die erste Zeile des folgenden Skripts stellt den „Normalfall" des objektorientierten Ansatzes dar, in der der Initiator Kunde eines Lieferanten ist, der in der Spalte Participant zu finden ist. Durch Rollen wird verdeutlicht, dass in der zweiten Spalte der Initiator ein Lieferant ist:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	erfragt Leistung :Action	:Interaction	/Lieferant :Unternehmen	liefert :Operation
/Lieferant :Unternehmen	liefert Leistung :Action	:Interaction	/Kunde :Person	empfängt Leistung :Operation

Skript 27: Verwendung von Rollen

Durch die Verwendung benannter Objekte wird ausgedrückt, dass nicht irgendein (unbenanntes bzw. anonymes) Objekt an der Beziehung teilnimmt, sondern ein ganz bestimmtes. Benannte Objekte werden insbesondere dann verwendet, wenn Objekte einer Klasse eindeutig unterschieden werden sollen:[1]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	...	Entnahme Beleg :Interaction	Belege 1998 :Ordner	...
:SB	...	Hinzufügen Beleg :Interaction	Belege 1999 :Ordner	...

Skript 28: Verwendung benannter Objekte

Auch benannte Objekte können mit Rollenbezeichnern versehen werden, um ihre Rolle in der Beziehung zu verdeutlichen.

[1] In diesem Kapitel wird AL als Abkürzung für Abteilungsleiter, GL als Abkürzung für Gruppenleiter, SB als Abkürzung für Sachbearbeiter verwendet.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Fa. Sch. Ladenbau /Lieferant :Unternehmen	...	...	Fa. Mobelhaus B. /Kunde :Unternehmen	...

Skript 29: Objekte, Rollen und Klassen

6.1.2 Stereotype

Stereotype stellen einen Erweiterungsmechanismus dar, um neue problemspezifische Modellelemente definieren zu können, ohne das Metamodell ändern zu müssen.[1] Sie bieten die Möglichkeit, Modellelemente zu kategorisieren[2] oder Partitionen zu bilden.[3] Ihre Verwendung geht auf Rebecca WIRFS-BROCK zurück, welche die drei Stereotype *entity*, *interface* und *control* zur Typisierung von Objekten während der Anforderungsanalyse verwendet.[4] Auch die von FERSTL und SINZ verwendeten Objekttypen – Umweltobjekte, konzeptuelle Objekttypen (KOT) und Vorgangsobjekttypen (VOT) – stellen im Sinne der UML Stereotype dar.[5]

---

[1] Ordnet man Stereotype in die 4-Ebenen-Architektur der UML ein, handelt es sich um benutzerdefinierte Erweiterungen des Metamodells auf der M1-Ebene der Modelle. Vgl. [Henderson-Sellers 2000: S. 33, 24]. Vgl. außerdem [Booch 1999: S. 29], [Frank 1997d: S. 48, 61], [Atkinson 2000: S. 34].

[2] Vgl. [Hitz 1999: S. 70].

[3] Vgl. [Henderson-Sellers 1998: Appendix E; Stichworte „Stereotypes“, „Partitions“, „Classification and partitions“, S. 2], [OMG 1999: S. 2-63 ff.]. Aus diesem Grund vergleicht BOOCH die Definition eines Stereotyps mit der Definition einer neuen Klasse in einem Metamodell. Vgl. [Booch 1999: S. 80].

[4] Vgl. [Jacobson 2000: S. 111]. Das Stereotyp *entity* wird für Datenobjekte der Persistenzschicht, *interface* für Elemente einer Schnittstelle (z. B. der Mensch-Maschine-Schnittstelle) und *control* für die Applikationslogik verwendet. Vgl. [Jacobson 1994: S. 132], [Jacobson 1999].

[5] Vgl. [Ferstl 1998]. Das Gleiche gilt auch für die von MALISCHEWSKI und AMBERG vorgenommenen Erweiterungen von SOM durch *Schnittstellenobjekttypen*, *Persistente Objekttypen* und *Technische Objekttypen*. Vgl. [Malischewski 1997], [Amberg 1993]. Problematisch bei dem dort entwickelten Verfahren ist, dass auf diese Weise Funktionen, Benutzerschnittstellen und Daten in getrennte Modelle (Objektschemata) zerlegt werden, was dem Kapselungsprinzip widerspricht. Ähnliche Probleme können auch bei den von WIRFS-BROCK eingeführten und von JACOBSON verwendeten Stereotypen entstehen, wenn sie zur Trennung von Funktionalität und Daten „missbraucht“ werden: Es entstehen Modelle mit Datenobjekten ohne Funktionalität und Funktionsobjekte ohne internes Gedächtnis (Zustand). Ohne hier auf die Details weiter eingehen zu können, soll hier der Hinweis genügen, dass dies nicht der Zweck der Stereotypen ist.

Stereotype werden hier durch das Prädikat `Stpe(String)` dokumentiert. Das Prädikat ist im Konzeptuellen Schema über den Metatyp *MODELELEMENT* für alle Modellelemente definiert, wird hier aber nur für den Initiator und den Participant verwendet.[1]

Stereotype sind vom Anwendungsbereich abhängig, da sie problemspezifisch definiert werden. Aus diesem Grund wird hier keine abschließende Definition aller möglichen Stereotypen erfolgen, sondern nur solcher, die für die GPM verwendet werden.[2] Auch in der UML sind eine Reihe von Stereotypen zur GPM vordefiniert.[3] Diese werden hier jedoch nicht verwendet, da ihnen eine entsprechende betriebswirtschaftliche Fundierung fehlt. Es erscheint sinnvoller, sich an den Begriffen der Betriebswirtschaftslehre und insbesondere an denen der Organisationslehre zu orientieren:[4]

- *Organisationen* (`Organization`) bestehen aus *Organisationseinheiten* (`Org-Unit`). Organisationseinheiten entstehen, wenn Aufgaben Personen zugeordnet werden.[5] *Stellen* (`Position`), *Gruppen* (`Group`) und *Abteilungen* (`Department`) sind Formen von Organisationseinheiten. Dabei bestehen Gruppen aus Stellen und Abteilungen aus Organisationseinheiten.[6]
- Als *Aktionsträger* (`Action Bearer`) bezeichnet man alle Objekte einer Organisation, die Aktivitäten ausführen.[7] Man unterscheidet *Sachmittel* [1] (`Resource`) und *Aufgabenträger* [2]

[1] Die Verwendung in den anderen Spalten der Skripte ist späteren Erweiterungen vorbehalten.

[2] Weitere Stereotype für die Modellierung von Softwaresystemen finden sich in [Henderson-Sellers 1998: Appendix E; Stichwort „Stereotypes"] oder in der Dokumentation der UML [Rumbaugh 1999: S. 499 ff.], [Diaz 1999]. JOOS setzt sich in [Joos 1998] kritisch mit dem Konstrukt des Stereotyps in der objektorientierten Modellierung auseinander.

[3] Vgl. [OMG 1999: S. 4-11]. Das Gleiche gilt für die in [Eriksson 2000: S. 76 ff.], [Hubert 2002] und [Marshall 2000] definierten Stereotype.

[4] Stereotype können untereinander in Spezialisierungsbeziehungen stehen (vgl. [Rumbaugh 1999: S. 449, 451], [Frank 1997d: S. 61]). Es wäre also möglich, für die hier definierten Stereotype Vererbungsbeziehungen zu definieren und auf diese Weise ein auf Stereotypen basierendes Metamodell zur Organisationsmodellierung zu entwickeln. Dies ist späteren Erweiterungen vorbehalten.

[5] Vgl. [Schulte-Zurhausen 1999: S. 129]. ÖSTERLE definiert organisatorische Einheit als Zusammenfassung einer oder mehrerer Stellen innerhalb oder außerhalb des Unternehmens. Vgl. [Österle 1995: S. 51]. Hier wird der Begriff Organisationseinheit rekursiv verwendet, damit eine Organisationseinheit aus Organisationseinheiten bestehen kann.

[6] Vgl. [Schulte-Zurhausen 1999], [Kosiol 1962], [Nordsieck 1955], [Nordsieck 1972], [Bleicher 1991], [Bleicher 1993]. Hier können auch die von BLEICHER verwendeten Begriffe *Basissystem* und *Zwischensystem* verwendet werden.

[7] Vgl. [Schulte-Zurhausen 1999: S. 50 f.], [Krüger 1992: Sp. 223]. Man beachte, dass dieser Begriff bei GROCHLA zur Bezeichnung von Menschen und automatisierten Hilfsmitteln verwendet wird. [Grochla

(`Task Bearer`). Die Erfüllung von Aufgaben geschieht durch *Aufgabenträger*. Aufgabenträger sind aktiv handelnd und verantwortlich für die Lösung einer Aufgabe. Aufgaben werden durch Menschen erfüllt,[3] darüber hinaus übernehmen auch Gruppen und Abteilungen Aufgaben.[4] Organisationen und Organisationseinheiten sind ebenfalls Aufgabenträger.

- Menschen verwenden *Sachmittel*, um Verrichtungen an Arbeitsobjekten zu vollziehen und Aufgaben zu erfüllen.[5] Sachmittel führen zwar ebenfalls Aktivitäten aus, sie handeln jedoch nicht aus eigener Initiative und Verantwortung, sondern programmiert.[6] Sie führen entweder relativ selbständig Aktivitäten durch und werden dann als *Arbeitsträger*[7] (`Action Bearer`) bezeichnet oder sie werden von einer Organisationseinheit zur Erfüllung ihrer Aufgaben verwendet und als *Arbeitsmittel*[8] (`Work Instrument`) bezeichnet. Technische Systeme wie Datenbanksysteme, Computer, Fertigungsroboter, Geldautomaten werden als Arbeitsträger abgebildet. Werkzeuge, Einrichtungsgegenstände und andere Hilfsmittel werden dagegen als Arbeitsmittel betrachtet. Mit Ausnahme der reinen Handarbeit stellen immer Menschen in

---

1975b: S. 428]. WILD verwendet diesen Begriff für alle menschlichen Aufgabenträger. Vgl. [Wild 1966: S. 91].

[1] [Nordsieck 1972: S. 5], [Hoffmann 1992b: Sp. 212], [Wild 1966: S. 92], [Kosiol 1962: S. 43], [Krüger 1994: S. 139], [Gallus 1979].

[2] Vgl. [Schmelzer 2002: S. 95], [Schulte-Zurhausen 1999: S. 71], [Hoffmann 1992b: Sp. 212], [Hill 1994: S. 131], [Wild 1966: S. 91], [Kieser 1992: S. 57].

[3] Vgl. [Schulte-Zurhausen 1999: S. 71], [Hoffmann 1992b: Sp. 212], [Hill 1994: S. 131], [Bleicher 1993: S. 117]. Während KOSIOL nur Menschen als verantwortliche Aufgabenträger ansah (vgl. [Kosiol 1962: S. 44]), könnte man diese Eingrenzung heute insofern relativieren, da heutige IT-Systeme selbständig operieren und damit ebenfalls im gewissen Sinne als Aufgabenträger anzusehen sind. So ist es schon bei GROCHLA [Grochla 1975b: S. 428] zu finden, der selbsttätige, sich selbst steuernde und selbst überwachende technische Aggregate als *maschinelle Aktionsträger* bezeichnet, die er von *personellen Aktionsträgern* unterscheidet.

[4] Vgl. [Wild 1966: S. 91], [Kieser 1992: S. 57].

[5] Unter Sachmittel versteht SCHULTE-ZUHAUSEN alle materiellen Hilfsmittel zur Prozessabwicklung. Vgl. [Schulte-Zurhausen 1999: S. 50, 67]. Vgl. außerdem [Nordsieck 1972: S. 5], [Hoffmann 1992b: Sp. 212], [Wild 1966: S. 92], [Kosiol 1962: S. 43], [Krüger 1994: S. 139]. In der klassischen Terminologie der Produktionstheorie wird der Begriff *Arbeitsmittel* und *Betriebsmittel* verwendet. Vgl. [Gutenberg 1973: S. 3 ff.].

[6] Vgl. [Bleicher 1993: S. 117]. Mit Programm meint er kein Softwareprogramm, sondern ein vorgegebenes Schema von Aktivitäten.

[7] [Kosiol 1962: S. 44, 88, 185]. Im objektorientierten WAM-Ansatz von ZÜLLIGHOVEN wird dies als *Automat* bezeichnet. Vgl. [Züllighoven 1998: S. 88 ff.].

[8] [Kosiol 1962: S. 43, 44, 88]. Vgl. auch [Wild 1966: S. 92]. Im objektorientierten WAM-Ansatz von ZÜLLIGHOVEN wird dies als *Werkzeug* bezeichnet. Vgl. [Züllighoven 1998: S. 83 ff.].

Zusammenarbeit mit Sachmitteln die Aktionsträger dar. Eine solche Kombinationen von Sachmitteln und Aufgabenträgern werden in Anlehnung an SCHULTE-ZURHAUSEN als *Aktionseinheit* (`Action Unit`) bezeichnet.[1]

- Als *Geschäftsobjekt* (`Business Object`; auch mit `BO` abgekürzt) werden die Gegenstände bezeichnet, an denen sich das Handeln in Prozessen vollzieht.[2] Bei der GPM sind die Geschäftsobjekte die betriebswirtschaftlichen *Bearbeitungs-*[3], *Aktions-*[4] bzw. *Arbeitsobjekte*. Es handelt sich um die in einem Prozess erstellten oder bearbeiteten, diskreten unterscheidbaren Entitäten, womit hier in erster Linie immaterielle Realobjekte (z. B. Informationen) in unterschiedlichen Granularitäten gemeint sind.[5] Aber auch die Arbeitsobjekte materieller Prozesse und Nominalobjekte werden als Geschäftsobjekte bezeichnet. Soll ein Geschäftsobjekt eindeutig als Informationsobjekt gekennzeichnet werden, kann hierfür das in der UML vordefinierte Stereotyp `Entity` verwendet werden.[6]
- *Methoden*, *Regeln*, *Verfahren* und *Kenntnisse* (`Knowledge`) sind *Informationen*[7], die von Aufgabenträgern zur Lösung von Aufgaben verwendet werden.[1] Dies können Gesetze,

[1] Vgl. [Schulte-Zurhausen 1999: S. 50].

[2] Diese Definition lehnt sich an den Begriff des *Aktionsobjekts* von WILD an. Vgl. [Wild 1966: S. 92].

[3] Vgl. [Schulte-Zurhausen 1999: S. 65], [Schmelzer 2002: S. 81].

[4] Der Begriff *Aktionsobjekt* stammt von WILD und bezeichnet die Gegenstände, an denen durch Verrichtungen Zustands- oder Lageveränderungen durchgeführt werden. Vgl. [Wild 1966: S. 92].

[5] *„Ein (Geschäfts-)Objekt ist ein realer oder gedachter Gegenstand der Leistungserstellung. Geschäftspartner, Anlagen, Aufträge, Verträge usw. sind typische Beispiele."* [Österle 1995: S. 87]. Weitere Beispiele sind Bestellungen, Auftragspositionen, Rechnungen, Rechnungspositionen usw. Vgl. [ZimmermannV 1999: S. 96]. MÜLLER-LUSCHNAT bezeichnet diese Objekte als *Bestandsobjekte*. Vgl. [Müller-Luschnat 1993: S. 78]. Bei FERSTL und SINZ werden sie als *Aufgabenobjekt* bezeichnet. Vgl. [Ferstl 1994: S. 7], [Ferstl 1995: S. 211]. Im objektorientierten WAM-Ansatz von ZÜLLIGHOVEN haben sie die Bezeichnung *Material*. Vgl. [Züllighoven 1998: S. 85 ff.]. GUTENBERG differenziert zwischen immateriellen *Dienstleistungen* und materiellen *Sachgütern*. Bei den *Sachgütern* unterscheidet er zwischen *Rohstoffen* und *Fabrikaten*. Vgl. [Gutenberg 1973: S. 1]. Sie würden hier – neben den Werkstoffen (a. a. O. S. 4) – als Geschäftsobjekte abgebildet.

[6] Vgl. [OMG 2001: S. 4-8].

[7] *Informationen* sind in der Betriebswirtschaftslehre aufgabenbezogene Nachrichten bzw. zweckorientiertes Wissen. Vgl. [Wild 1966: S. 97], [Heinen 1971: S. 24], [Schmidt 1985: S. 74], [Heinen 1985: S. 62], [Berthel 1992: Sp. 872], [Schulte-Zurhausen 1999: S. 63], [Krüger 1994: S. 142]. Zu den besonderen Eigenschaften der Unternehmensressource Information siehe [Picot 1988], [Picot 1988b], [Picot 1991: S. 250 ff.], [Picot 1994].

Vorschriften, Verfahrensanweisungen, Anleitungen, Handlungsanweisungen oder Standards sein. Sie stellen eine besondere Art von Arbeitsmitteln dar. In diesem Sinne besitzt das Element Information analytisch eine Doppelrolle, weil es sowohl ein Arbeitsobjekt wie auch ein *nichtmaterielles Sachmittel* sein kann.[2]

- *Ziele* (`Goal`) und *Probleme* (`Problem`) stellen Informationen über angestrebte oder zu verändernde Zustände dar.[3] Sie werden als Objekte modelliert, wenn sie Eigenschaften oder Zustände besitzen, die über Operationen verändert oder erfragt werden.
- *Individuen* (`Individual`) sind natürliche Personen[4], die nicht als Aufgabenträger abgebildet werden sollen.
- Durch *Prozesse* (`Process`) werden durch eine Folge logisch zusammenhängender und inhaltlich miteinander verknüpfter Aktivitäten Leistungen erstellt oder Objekte verändert.[5] Sie werden als Objekt modelliert, wenn sie eigenständige, identifizierbare Entitäten darstellen, deren Eigenschaften, Zustände und Zustandsveränderungen betrachtet werden, die man durch

---

[1] *Kenntnisse, Verfahren* und *Methoden* sind als *geistiges Hilfsmittel* (vgl. [Krüger 1992: Sp. 222], [Schulte-Zurhausen 1999: S. 51]) zu verstehen, die zur Ausführung von Aktivitäten verwendet werden. Unter *Verfahren* versteht SZYPERSKI die Art und Weise, einen Zweck zu erreichen. Vgl. [Szyperski 1961: S. 108].

[2] Vgl. [Krüger 1994: S. 17]. Auch SCHULTE-ZURHAUSEN weist auf die Rolle der Information als Produktionsfaktor hin. Vgl. [Schulte-Zurhausen 1999: S. 67].

[3] Zum Begriff des Ziels vgl. [Nordsieck 1955: S. 28]. Nach FERSTL und SINZ ist ein Ziel der angestrebte Zustand eines Objekts. Vgl. [Ferstl 1994c: S. 5, Fußnote 2]. Erstmalig erwähnt wurden Ziele im Bereich der objektorientierten Modellierung von KUENG, BICHLER und SCHREFL in [Kueng 1995b]. Allerdings wurden sie dort nur verwendet, um die von Objekten auszuführenden Aktivitäten herzuleiten. Die Modellierung von Zielen als Objekte findet sich in [Eriksson 2000: S. 273 ff.]. Für eine weitergehende Darstellung der Begriffe Ziel, Zielsystem, Zielformulierung siehe [Nagel 1992: Sp. 2626 – 2652], [Hill 1994: S. 24, 141 ff.], [Heinen 1985: S. 93 ff.], [Heinen 1971], [Bamberg 1985: S. 24 ff.]. Auf die Problematik der Zielbildung, der Inhalte von Unternehmenszielen, der Gestaltung des Zielsystems und der Beziehungen zwischen Zielen etc. wird hier nicht weiter eingegangen, da diese Fragestellungen für den Fortgang der Arbeit von geringem Interesse sind. Vgl. hierzu [Heinen 1971], [Hamel 1992], [Adam 1996: S. 99 ff.]. Hier mag die vereinfachende Annahme genügen, dass durch Aktivitäten ein Ziel erreicht werden soll, welches sich in Form eines Zustandes formulieren lässt. Es handelt sich damit um operationale Ziele (vgl. [Heinen 1971: S. 115]) in der Art, dass festgestellt werden kann, ob das als Zustand formulierte Ziel erreicht wird.

[4] Juristische Personen werden als Organisationen abgebildet.

[5] [Schulte-Zurhausen 1999: S. 49, 60, 107], [Striening 1988: S. 57]. MÜLLER-LUSCHNAT bezeichnet diese Objekte als *Vorgangsobjekte*. Vgl. [Müller-Luschnat 1993: S. 82].

Operationen verändert.[1] In diesem Fall stellen sie eine *Verkettung von Aktivitäten* dar.[2] Ein Beispiel für solche Prozessobjekte sind die von KUENG und BICHLER verwendeten *Business-Case*-Klassen.[3] Deren Attribute beinhalten den Zustand des Geschäftsprozesses. Das im Bereich der Workflow-Modellierung verwendete Konzept der *Umlaufmappe*[4] wird als Prozess abgebildet, wenn es die Abfolge der notwendigen Aktivitäten beinhaltet oder die Zustände des Vorgangs erfasst werden.

Es ist von der Anwendungssituation abhängig, ob und wie präzise man eine Klasse stereotypisiert. Bei der GPM wird man beispielsweise zunächst davon ausgehen, dass es sich bei den beteiligten betrieblichen Objekten um Aktionsträger handelt. Erst im weiteren Verlauf der Modellierung wird man die Entscheidung treffen, welche Aktivitäten von Aufgabenträgern und welche Aktivitäten von Arbeitsträgern ausgeführt werden. Darauf wird noch im nächsten Kapitel eingegangen.

Das folgende Beispiel demonstriert die Verwendung einiger Stereotype.[5] Eine Abteilung Lohnbuchhaltung hat die Aufgabe, den geldwerten Vorteil der verbilligten Abgabe von Produkten an einen Mitarbeiter zu berechnen.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Lohnbuchhaltung Stpe(Department)	bestimmt Bruttolistenpreis :Action	:Interaction	:Rechnung Stpe(Business Object)	ermittle Bruttolistenpreis :Operation
:Lohnbuchhaltung Stpe(Department)	bestimmt Abgabepreis :Action	:Interaction	:Rechnung Stpe(Business Object)	ermittle Abgabepreis :Operation
:Lohnbuchhaltung Stpe(Department)	bestimmt Rabatt :Action	:Interaction	:EStG Stpe(Knowledge)	ermittle Rabatt :Operation
:Lohnbuchhaltung Stpe(Department)	entnimmt bisher gewährten Rabatt	:Interaction	:Lohnkonto Stpe(Business Object)	ermittle Rabattsumme :Operation

[1] In der Modellierungspraxis wird häufig gefordert, zwischen Prozessen Spezialisierungsbeziehungen definieren zu können. Auch um diese Forderung erfüllen zu können, müssen Prozesse als Klassen modelliert werden. Vgl. zum Thema der Spezialisierung von Geschäftsprozessen [Kueng 1995], [Müller-Luschnat 1993].

[2] Diese Formulierung geht auf die Position von HILL et. al. zurück. Danach beinhalten Aufgaben verschiedene Aktivitäten, die Elemente von Prozessen sind. Vgl. [Hill 1994: S. 123]. In ähnlicher Weise definiert WILD das Aktionsgefüge eines Betriebes als einen komplexen Prozess, der aus einer Vielzahl zusammenhängender, aufeinander abgestimmter und auf ein Sachziel ausgerichteter Aktionen besteht. Vgl. [Wild 1966: S. 94].

[3] Vgl. [Kueng 1996b: S. 12].

[4] Vgl. [Weise 1999].

[5] Auf die Verwendung der Stereotype wird noch im Kapitel über die Modellierung von Geschäftsprozessen eingegangen.

		:Action			
	:Lohnbuchhaltung Stpe(Department)	ermittelt Rabattfreibetrag :Action	:Interaction	:EStG Stpe(Knowledge)	nenne Rabattfreibetrag :Operation
	:Lohnbuchhaltung Stpe(Department)	errechnet geldwerten Vorteil :Action	:Interaction	:EStG Stpe(Knowledge)	ermittle geldwerten Vorteil :Operation
	:Lohnbuchhaltung Stpe(Department)	trägt Rabatt ein :Action	:Interaction	:Lohnkonto Stpe(Business Object)	speichere neuen Rabatt :Operation
	:Lohnbuchhaltung Stpe(Department)	trägt Vorteil ein :Action	:Interaction	:Lohnkonto Stpe(Business Object)	speichere neuen geldwerten Vorteil :Operation

Skript 30: Verwendung von Stereotypen[1]

Die Verwendung von Stereotypen ist insbesondere dann sinnvoll, wenn bei der GPM zwischen Realwelt-Objekten und Informationen über diese Realwelt-Objekte in Form von Daten differenziert werden soll. Verwendet man die Stereotype `Individual` und `Business Object`, wird verdeutlicht, wann von Personen und wann von Daten die Rede ist, durch die Personen beschrieben werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Besteller Stpe(Individual)	bestellt Ware :Action	:Interaction	:SB Stpe(Position)	nimmt Bestellung entgegen :Operation
	:SB Stpe(Position)	legt neuen Kunden an :Action	:Interaction	:Kunde Stpe(Business Object)	anlegen :Operation

Skript 31: Kennzeichnung von Objekten durch Stereotype

Die Modellierung von Organisationseinheiten als Klassen bzw. Objekte unterscheidet sich von der objektorientierten Erweiterung der Ereignisgesteuerten Einzelprozesskette (oEPK) der Methode ARIS.[2] Dort werden die Organisationseinheiten nicht als Klassen abgebildet. Auf den in der UML vorgesehenen Sonderfall, Ereignisse als Objekte bzw. Klassen abzubilden und durch den Stereotyp `Signal` zu kennzeichnen,[3] wird in Abschnitt 6.8 eingegangen. In den Tabellenspalten Initiator Responsibility und Participant Responsibility sowie in der Tabellenspalte Connection können bei Bedarf ebenfalls Stereotype verwendet werden, da dieses Prädikat für alle Modellelemente definiert ist.

[1] Der Inhalt des Beispiels stammt aus [Heeg] und wurde entsprechend der hier verwendeten Notation verändert.

[2] Vgl. [ZimmermannV 1999], [Nüttgens 1997], [Nüttgens 1998].

[3] Vgl. [Rumbaugh 1999: S. 427].

6.1.3 Diskurswelt und Umwelt

Auch die Umwelt eines Systems kann konsequenterweise als aus Objekten bestehend betrachtet werden. Schon bei der Modellierung von Softwaresystemen ist es in der Regel notwendig, dessen Umsystem in verschiedene Benutzergruppen und Softwaresysteme zu differenzieren. Trotzdem ist bei allen objektorientierten Modellierungsansätzen festzustellen, dass sie Umsysteme als monolithische Blöcke betrachten. So wird zum Beispiel in der Methode SOM nur zwischen den Objekten der betrieblichen Diskurswelt (*Diskursweltobjekte*) und den Objekten der Umwelt (*Umweltobjekte*) unterschieden.[1] In der UML werden alle Objekte der Umwelt als Akteure bezeichnet. Dabei wird nicht zwischen menschlichen Benutzern, externen Computersystemen oder anderen Fremdsystemen unterschieden.

Diese Problematik verstärkt sich bei der GPM, die im Rahmen externer Prozessverkettung[2] durchaus die institutionellen Grenzen eines Unternehmens überschreiten können, wenn beispielsweise die Informationssysteme eines Zulieferers mit den eigenen verbunden werden, um den Ablauf von Lieferprozessen zu beschleunigen. In diesem Fall ist es nicht mehr ausreichend, einfach vom Lieferantenobjekt zu sprechen, wenn eine differenzierte Betrachtung, mit welchem Objekt des Lieferantensystems interagiert wird, notwendig wird. Aus den unter Initiator und Participant verwendeten Bezeichnungen geht aber nicht unbedingt hervor, welche Objekte und Klassen zum betrachteten System und welche zu einem anderen System gehören. Zur Illustration wird folgendes Skript verwendet:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Auslieferungslager	fragt Bestandsdaten ab :Action	:Interaction	:Lagerdaten	Bestand ermitteln :Operation

Skript 32: Systemübergreifende Interaktion (1)

Aus diesem Skript wird der Umstand nicht klar, dass das Objekt der Klasse `Auslieferungslager` zum Lieferanten, dass Objekt der Klasse `Lagerdaten` aber zum Fertigungsunternehmen gehören soll. Eine Möglichkeit besteht nun darin, die Klassen durch eine Aggregationsbeziehung mit einer Klasse `Lieferant` bzw. `Hersteller` zu verbinden. Auf diese Weise erhält man:

[1] Vgl. [Ferstl 1998: S. 4], [Wassermann 2001].

[2] Vgl. [Krüger 1994: S. 127].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Lieferant		:Aggregation	:Auslieferungslager	
	:Auslieferungslager	fragt Bestandsdaten ab :Action	:Interaction	:Lagerdaten	Bestand ermitteln :Operation
	:Hersteller		:Aggregation	:Lagerdaten	

Skript 33: Systemübergreifende Interaktion (2)

So eingängig diese Lösung im ersten Moment scheinen mag, ist sie doch nicht praktikabel. Abgesehen vom zusätzlichen Modellierungsaufwand wird diese Konzeption spätestens dann problematisch, wenn auch für den Hersteller ein Auslieferungslager modelliert werden muss. In diesem Fall wäre die Klasse `Auslieferungslager` sowohl mit der Klasse `Hersteller` wie auch mit der Klasse `Lieferant` über eine Aggregationsbeziehung verbunden. Auch die Lösung, eine Differenzierung durch die Vergabe von Objektbezeichnern herbeizuführen, ist nur unwesentlich besser, da diese Objektbezeichner dann Metainformationen über ihre Zugehörigkeit zu einem System beinhalten. Im hier entwickelten Ansatz wird eine differenziertere Betrachtung ermöglicht, als dies beispielsweise in der UML durch Akteure der Fall ist. Durch das Prädikat `System(`*`System Identifier`*`)` wird festgelegt, zu welchem System eine Klasse bzw. ein Objekt gehört. Der Standardwert – der nicht angegeben werden muss – für dieses Prädikat ist `internal`, wodurch festgelegt wird, dass das Objekt zur betrieblichen Diskurswelt gehört. Durch den Term `external` wird dagegen festgelegt, dass sich das (Umwelt-)Objekt jenseits der festgelegten Systemgrenze befindet, wo immer diese auch angenommen wird. Dies kann durch die Verwendung eines Stereotyps weiter differenziert werden. Auf diese Weise kann Skript 32 in folgender Weise präzisiert werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Auslieferungslager System(external) Stpe(Department)	fragt Bestandsdaten ab :Action	:Interaction	:Lagerdaten System(internal) Stpe(Business Object)	Bestand ermitteln :Operation

Skript 34: Interaktion mit externem Initiator

Zusammengefasst bedeutet das, dass durch das Prädikat System() Klassen unterschiedlichen Domänen (Problembereichen) zugeordnet werden können. Auf diese Weise wird die gezogene Systemgrenze und damit „innen“ und „außen“ deutlich.

6.1.4 Anzahl der Exemplare einer Klasse

Objekte stellen die Extension einer Klasse dar. Häufig tritt der Fall auf, dass durch eine Klasse ein einzelnes oder eine genau definierte Anzahl von Objekten beschrieben wird. Dies ist dann der Fall, wenn dieses Objekt Eigenschaften hat bzw. Verhalten zeigt, welches es von allen anderen Objekten unterscheidet. Ein typisches Beispiel wären bei der objektorientierten Modellierung einer Organisation die Abteilungsleiter eines Unternehmens. Abteilungsleiter besitzen Aufgaben, die vom spezifischen Aufgabenbereich der Abteilung abhängen, die sie führen. Während der Abteilungsleiter einer Abteilung Projektabwicklung eine Tourenplanung seiner Techniker durchführt, wird der Leiter der Entwicklungsabteilung den Aufwand für das nächste Projekt abschätzen.

In einer solchen Situation wird man nicht nur eine allgemeine Klasse `Abteilungsleiter` definieren, sondern für jeden Abteilungsleiter eine weitere spezifische Klasse, die jeweils nur ein Exemplar besitzt – die Person des Abteilungsleiters. Solche Klassen, deren Extension ein einzelnes Objekt umfasst, wurden von EMBLEY als *Singleton* bezeichnet.[1] Diese Restriktion wird durch das Prädikat `Card(`*`Cardinality Expression`*`)` abgebildet, welches im Metamodell des Konzeptuellen Schemas für den Typ *CLASS* definiert ist. Auf diese Weise wird durch `Card()` der Umfang der Extension einer Klasse eingeschränkt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Geschäftsführer Card(1)	...	...	:Leiter Buchhaltung Card(1)	...

Skript 35: Beziehung zwischen Objekten von Singleton-Klassen

Durch das Skript 35 wird festgelegt, dass die beteiligten Objekte die einzigen Exemplare der jeweiligen Klasse `Geschäftsführer` und `Leiter Buchhaltung` darstellen.

6.2 Prädikate der Interaktionsbeziehung

Die Modellierung von Interaktionen ist in der UML angesichts ihrer Bedeutung im objektorientierten Ansatz erstaunlich schwach ausgebildet. Im Wesentlichen beschränkt man sich darauf, die im Rahmen der Interaktion übertragene Nachricht mit der aufgerufenen Operation des

[1] Vgl. [Embley 1992], [Rumbaugh 1999: S. 430].

Empfängerobjekts gleichzusetzen.[1] Außerdem kann man die Art der Synchronisation (siehe Abschnitt 6.2.6) und eine so genannte Überwachungsbedingung (siehe Abschnitt 6.2.5) formulieren. Alle übrigen hier entwickelten Möglichkeiten stellen Erweiterungen gegenüber der UML dar.

Entsprechend dem in dieser Arbeit formulierten *Primat der Interaktionsbeziehung* dienen Skripte primär zur Abbildung der zwischen Objekten stattfindenden Interaktionen.[2] In der Interaktionsbeziehung interagiert der in der Tabellenspalte Initiator aufgeführte Beteiligte mit dem oder den in der Tabellenspalte Participant aufgeführten Beteiligten. Die Interaktion geht vom Initiator aus. Weitere Elemente der Interaktionsbeziehung sind die Aktion des Initiators, die die Ursache der Interaktion darstellt, und die Operation des Participants, die als Reaktion der Interaktion stattfindet. Interaktionen können einen Namen erhalten, wodurch sie innerhalb eines Skripts referenziert werden können. Der Modellierer ist in der Vergabe der Namen frei. In den folgenden Abschnitten werden die umfangreichen Möglichkeiten vorgestellt, Interaktionen abzubilden.

6.2.1 Ströme

Bei der Modellierung von Softwaresystemen wird implizit davon ausgegangen, dass Objekte Informationen austauschen. Dies ist nahe liegend, weil in einem Softwaresystem Informationen in Form von Daten fließen. Aus diesem Grund ist in den Modellierungsansätzen zur Softwareentwicklung keine weitere Differenzierung über die Art des Stromes notwendig. Bei der Modellierung von betrieblichen Prozessen ist dies nicht ausreichend, da – selbst bei den primär informationsverarbeitenden Geschäftsprozessen – nicht nur Informationen ausgetauscht werden.

HEINEN unterscheidet Informations-, Güter- und Geldströme.[3] FERSTL und SINZ unterscheiden sehr allgemein den Austausch von Informationen, Materie und Energie[4] und bei den Flüssen in betrieblichen Abläufen zwischen Güterflüssen (inkl. Energie), Zahlungsflüssen und Dienstleistungsflüssen.[5] Eine andere Einteilung wird von SCHEER vorgenommen, der in ARIS zwischen

---

[1] Eine Ausnahme stellt PAGE-JONES [Page-Jones 2000] dar, der zwischen der aufgerufenen Operation und der übertragenen Nachricht unterscheidet.

[2] Vgl. Abschnitt 1.4.

[3] [Heinen 1985: S. 62]. In gleicher Weise auch in [Kosiol 1966: S. 115, 167] und [Zelewski 1994: S. 75] zu finden. In vereinfachten Modellen werden nur Geld- und Güterströme berücksichtigt, wie z. B. in [Wöhe 1986: S. 11], [Kappler 1991b: S. 902]. Es soll angemerkt werden, dass nach WILD die Betrachtung des Informationsstroms erst verspätet zum Betrachtungsgegenstand der Betriebswirtschaftslehre wurde. Vgl. [Wild 1970: S. S. 50].

[4] Vgl. [Ferstl 1998: S. 17]. Ähnlich auch in [Schulte-Zurhausen 1999: S. 37].

[5] Vgl. [Ferstl 1998: S. 41].

Leistungsfluss und Informationsfluss differenziert. Bei den Leistungen werden Sach- und Dienstleistungen unterschieden.[1] Den Finanzmittelfluss subsumiert SCHEER unter den Leistungsfluss.[2] Informationen werden von SCHEER dann als Informationsfluss abgebildet, wenn sie Mittelcharakter besitzen und nicht Gegenstand bzw. Teil einer Leistung sind.

Dies erscheint problematisch, wenn nicht klar ist, ob die Informationen in die erstellte Leistung einfließen oder verwendet werden, um eine bestimmte Aktivität ausführen zu können, also Mittelcharakter besitzen. Um dieses Zuordnungsproblem zu vermeiden, wird in der OBA++ nicht zwischen Leistungen und Informationen unterschieden. Stattdessen wird der Fluss von Nominalgütern, Realgütern und Informationen unterschieden.[3] Sollte der Fall auftreten, dass Energieflüsse abgebildet werden sollen, werden sie unter die Realgüter subsumiert. Um Dienstleistungsbeziehungen ohne identifizierbare Flüsse[4] kennzeichnen zu können, wird ein Dienstleitungsfluss eingeführt. Zusätzlich wird ein Arbeitsleistungsfluss eingeführt, um die Verrichtung an einem Arbeits- bzw. Geschäftsobjekt durch ein anderes Objekt abzubilden. Der Inhalt des Stromes ist in diesem Fall die verrichtete Arbeit.

In der Skriptnotation wird das Prädikat `Flow(`*`Flow Type`*`)` zur Abbildung der unterschiedlichen Ströme verwendet. Da bei der Modellierung von Informationssystemen und Geschäftsprozessen der Informationsfluss (`Information`) die maßgebende Rolle spielt, braucht er nicht speziell gekennzeichnet zu werden, sondern stellt den Vorgabewert dar. Die übrigen Terme sind Materie (`Material`), Geld (`Money`), Dienstleistung (`Service`) und Arbeitsleistung (`Performance`). Das folgende Skript demonstriert die Verwendung an einem vereinfacht dargestellten Geschäftsprozess:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Person	bestellt Ware :Action	:Interaction	:Auftragserfassung	nimmt Bestellung entgegen :Operation
	:Fertigungs-abteilung	erstellt Artikel :Action	:Interaction Flow(Performance)	Art.Nr. 45672 :Artikel	wird gefertigt :Operation
	:Versand	versendet Ware :Action	:Interaction Flow(Material)	:Person	erhält Ware :Operation
	:Person	bezahlt Ware :Action	:Interaction Flow(Money)	:Buchhaltung	gleicht offenen Posten aus :Operation

Skript 36: Abbildung unterschiedlicher Ströme durch das Prädikat Flow

[1] Vgl. [Scheer 1998: S. 13 – 16], [Gutenberg 1973: S. 1].

[2] a. a. O., S. 23.

[3] Dies orientiert sich an der Gliederung von [Zelewski 1994: S. 75 ff.]. Ein alternativer Standpunkt wäre der, den Geldfluss als Information über Geld zu interpretieren und unter den Informationsfluss zu subsumieren. Diese doppelte Zuordnungsmöglichkeit rechtfertigt nach Meinung des Verfassers die in der OBA++ vorgenommene Definition eines separaten Geldstroms.

[4] Man denke zum Beispiel an die Dienstleistung „*Beherbergung*“ eines Hotels.

Auch eine weitergehende Differenzierung kann – auch wenn sie hier nicht erfolgt – sinnvoll sein. So ließen sich bei den Informationsströmen zum Beispiel Unterrichtungs-, Anordnungs- und Mitteilungsströme unterscheiden.[1]

6.2.2 Inhalt der Interaktion

In der objektorientierten Modellierung unterscheidet man *unidirektionale* und *bidirektionale* Interaktion zwischen Objekten. Bei der unidirektionalen Interaktion sendet ein Objekt A einem Objekt B eine Nachricht. Diese Nachricht kann einen Inhalt (Informationen, Materie usw.) enthalten.[2] Bei der bidirektionalen Interaktion sendet ein Objekt A einem Objekt B eine Nachricht und wartet auf eine Antwort.[3] Diese Antwort von Objekt B an Objekt A wird implizit als Bestandteil der Nachricht von A zu B betrachtet und nicht als separate Interaktion. Das Prädikat `Content(`*`String`*`)` wird verwendet, um die im Rahmen der Interaktion vom Initiator zum Participant übertragenen Inhalte abzubilden. Der oder die Inhalte stellen Objekte dar, die vom Initiator zu Participant übertragen werden.[4] Im folgenden Beispiel werden unbenannte Objekte der Klassen `Abflugort`, `Zielort`, `Abflugdatum` und `Rückreisedatum` übertragen:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Reisender	nennt Reisedaten :Action	:Interaction Content(:Abflugort, :Zielort, :Abflugdatum, :Rückreisedatum)	:Reisebüro	reserviert Flug :Operation

Skript 37: Interaktion mit Inhalt

Zusätzlich benötigt man eine Möglichkeit, die im Rahmen bidirektionaler Interaktion vom Participant an den Initiator übertragenen Inhalte abzubilden. Eine solche Rückgabe wird durch das Prädikat `Answer(`*`String`*`)` dargestellt. Auf diese Weise lässt sich durch die Kombination von `Content()` und `Answer()` eine Anfrage mit direkter Antwort abbilden:

---

[1] Vgl. [Kosiol 1962: S. 148, 149], [Hill 1994: S. 137 f.].

[2] Sie kann aber auch inhaltsleer sein. Darauf wird noch eingegangen.

[3] Vgl. [Embley 1992: S. 177].

[4] Eine ähnliche Sichtweise, die Argumente von Nachrichten konsequenterweise als Objekte anzusehen, findet sich in [Page-Jones 1994].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:GL	fragt nach Status eines Auftrags :Action	:Interaction Content(:Autrag) Answer(:Status)	:SB	gibt Status eines Auftrags zurück :Operation

Skript 38: Interaktion mit Rückgabewert (1)

Die Kombination der Prädikate `Content()` und `Answer()` bedeutet hier, dass im Rahmen einer Interaktion vom Initiator zum Participant ein Auftragsobjekt übertragen wird und in der gleichen Interaktion in umgekehrter Richtung ein Statusobjekt zurück übertragen wird. Die Objekte sind Exemplare der Klassen `Auftrag` und `Status`.[1]

Für die Prädikate `Content()` und `Answer()` können sowohl unbenannte wie auch benannte Objekte verwendet werden. Dabei wird die gleiche Notation wie in den Tabellenspalten Initiator und Participant verwendet: [2]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:GL	fragt nach Status eines Auftrags :Action	:Interaction Content(01289 :Auftragsnummer, 9818: Kundenr.) Answer(:Auftragsstatus)	:SB	gibt Status eines Auftrags zurück :Operation

Skript 39: Interaktion mit Rückgabewert (2)

Wie in Skript 39 zu sehen ist, ist die Spezifikation des Inhalts einer Interaktion nicht auf unbenannte Objekte beschränkt. Das Prädikat `Content(01289 :Auftragsnummer, 9818 :Kundenummer)` bedeutet, dass ein Auftragsstatus zum Auftrag mit der Auftragsnummer 01289 und mit der Kundennummer 9818 übertragen wird. Auf diese Weise wird eine Interaktion mit aktuellen Werten definiert. Kann für den Inhalt der Nachricht kein Typ festgestellt werden, wird die Notation `Objektbezeichner :Klassenbezeichner` nicht verwendet. Stattdessen wird eine Textkette verwendet:

[1] Diese werden in der Methode ROOM als *data classes* bezeichnet. Vgl. [Selic 1994: S. 80].

[2] Man beachte, dass keine Argumente für die Operation `gibt Status eines Auftrags zurück` festgelegt werden. Argumente für Operationen werden in Abschnitt 6.6.1 eingeführt.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Anwender	gibt Daten ein :Action	:Interaction Content(Adressdaten)	:OnlineBestell-system	liest Adressdaten ein :Operation

Skript 40: Interaktion mit nicht-typisiertem Inhalt

Die Übertragung eines Inhalts ist im objektorientierten Ansatz nicht mit der Übertragung der Nachricht zu verwechseln. Auch Nachrichten, die keine Daten transportieren, stellen im objektorientierten Ansatz Nachrichten dar, wodurch sich das objektorientierte Kommunikationsmodell von dem der Nachrichtentechnik unterscheidet. Insbesondere Interaktionen, die die Änderung des Betriebsmodus eines Objekts oder eines ganzen Systems bewirken,[1] finden häufig ohne Datenübertragung statt. So stellt die Interaktion, durch die der Betriebsmodus einer Maschine geändert wird, in der objektorientierten Modellierung eine Nachrichtenübertragung dar, auch wenn keine Daten übertragen werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Bediener	einschalten der Maschine :Action	:Interaction	:Maschine	Betriebsmodus ändern :Operation

Skript 41: Interaktion ohne Datenübertragung

Diese widersprüchliche Situation ist zu erklären, wenn man die Aufforderung an den Participant, eine Operation auszuführen, als Inhalt der Nachricht auffasst. So ist es auch im objektorientierten Ansatz zu verstehen. Die Aufforderung „`Ändere Betriebsmodus`“ stellt den Inhalt der Nachricht dar. Dies wird deutlicher, wenn man die Interaktion in Skript 41 in folgender Weise umformuliert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Bediener	einschalten der Maschine :Action	:Interaction Content(Ändere Betriebsmodus)	:Maschine	schalten :Operation

Skript 42: Abbildung einer Steuernachricht

Die zweite Lösung ist flexibler, da die Operation `schalten` der Klasse `Maschine` mit unterschiedlichen Inhalten aufgerufen werden kann (`Ändere Betriebsmodus`, `Einschalten`, `Ausschalten` etc.). Dem steht als Nachteil gegenüber, dass ggf. überprüft werden muss, ob der übertragene Inhalt zulässig ist. Die erste Lösung ist weniger flexibel, aber robuster, da ein Aufruf

[1] Dies wird als sog. *Steuersignal* bezeichnet. Vgl. [Hatley 1993: S. 200].

mit fehlerhaften Daten nicht möglich ist. Die Prädikate `Flow()`, `Form()` und `Content()` dokumentieren jeweils unterschiedliche Aspekte von dem, was im Rahmen einer Interaktion vom Initiator zum Participant übertragen wird.

Eine andere Möglichkeit, Interaktionen zu verwenden, ist die, kommunizierende Datenträger zu modellieren.[1] In diesem Fall stellen die Datenträger keine Objekte dar, die über Interaktionen ausgetauscht werden, sondern interagieren direkt miteinander. Dies kann beispielsweise dafür verwendet werden, dass im Rahmen von Geschäftsprozessen eine Abfolge von Belegen erzeugt und verarbeitet wird. Ein typisches Beispiel ist die Abfolge von Kundenauftrag, Auftragsbestätigung, Lagerschein, Versandpapieren und Rechnung.[2] Wenn abgebildet werden soll, dass aus einem Beleg ein Folgebeleg entsteht, wird dies durch eine Interaktionsbeziehung abgebildet. Die leeren Zeilen im Skript sollen andeuten, dass durch weitere Interaktionen die Belege durch Aktionsträger erfasst, geprüft, freigegeben, gespeichert und versendet werden.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Kundenauftrag	erzeugt :Action	:Interaction	:Auftragsbestätigung	wird erzeugt :Operation
...	...	...	...	...
:Auftragsbestätigung	erzeugt :Action	:Interaction	:Lagerschein	wird erzeugt :Operation
...	...	...	...	...
:Lagerschein	erzeugt :Action	:Interaction	:Versandpapier	wird erzeugt :Operation
...	...	...	...	...
:Versandpapier	erzeugt :Action	:Interaction	:Rechnung	wird erzeugt :Operation

Skript 43: Abfolgen von Belegen

Bei Interaktionen wird der symbolische Bezeichner *self* (vgl. Abschnitt 6.2.12) verwendet, wenn der Initiator eine Referenz auf sich selbst an den Participant übergibt. Diese Mimik ist insbesondere dann notwendig, wenn ein Objektfluss modelliert wird, bei dem sich der Initiator zum Participant bewegt. Das Skript 44 demonstriert die Verwendung:

[1] Der Begriff der *kommunizierenden Datenträger* geht auf NORDSIECK zurück. Vgl. [Nordsieck 1972: Sp. 153].

[2] Vgl. [HeilmannM 1997: S. 55].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Patient :Person	begibt sich ins Krankenhaus :Action	:Interaction Content(self :Person)	:Krankenhaus	nimmt Patient auf :Operation

Skript 44: Initiator als Inhalt der Interaktion

6.2.3 Form der Interaktion

In der objektorientierten Modellierung wird üblicherweise nicht zwischen den unterschiedlichen Formen unterschieden, in denen eine Nachricht auftreten kann. Nach KRISTEN können Nachrichten die Form von Telefonanrufen, verbaler Kommunikation, Notizen, Signalen, Formularen, Dokumenten, Belegen, Briefen, Berichten etc. haben.[1] Das Prädikat `Form(`*`Form Identifier`*`)` bildet für Interaktionen die physische Repräsentation bzw. das *Medium* der Interaktion ab.

Das folgende Skript demonstriert die Verwendung des Prädikats `Form()` in diesem Zusammenhang. Ein Objekt der Klasse `Antragsteller` sendet einen Antrag an einen Sachbearbeiter `SB` in Form eines Formulars. Der bearbeitende Sachbearbeiter erfasst den Antrag, wodurch er in der folgenden Interaktion die Form von Daten besitzt.[2]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Antragsteller	sendet Antrag :Action	:Interaction Flow(Information) Form(Formular)	:SB	erfasst Antrag :Operation
	:SB	überträgt Antrag :Action	:Interaction Flow(Information) Form(Email)	...	

Skript 45: Form der Interaktion

Das Beispiel macht auch deutlich, warum zwischen der Form der Interaktion und der Art des Stroms unterschieden werden muss. Unabhängig von der physischen Repräsentation handelt es sich in beiden Fällen um einen Informationsfluss.

[1] Vgl. [Kristen 1995].

[2] Für das Prädikat `Form()` werden keine vordefinierten Terme verwendet.

6.2.4 Koordinationsprotokolle

Als Koordination wird die Abstimmung von Objekten auf ein Ziel bezeichnet.[1] Durch Koordinationsprotokolle wird beschrieben, wie Objekte ihre Aktivitäten durch Interaktionen abstimmen. Dabei werden nach FERSTL und SINZ hierarchische und nicht-hierarchische Koordinationsformen unterschieden.[2] Sie unterscheiden die Interaktionstypen *Anbahnung*, *Vereinbarung* und *Durchführung*, um die Phasen nicht-hierarchischer Koordination autonom handelnder Objekte (als *Verhandlung* bezeichnet) zu beschreiben.[3]

- *Anbahnung*: Im Rahmen der Anbahnung treten Initiator und Participant in Kontakt und tauschen Informationen über die gewünschten oder geforderten Leistungen, Konditionen usw. aus.
- *Vereinbarung*: Die Vereinbarung dient dem Abschluss einer Übereinkunft über den Leistungsaustausch. Durch die Interaktion werden irgendwie geartete Vereinbarungen zwischen Initiator und Participant ausgetauscht. Nach Beendigung dieser Phase liegt eine Verpflichtung bezüglich des Leistungsaustauschs vor.
- *Durchführung*: Die Durchführung dient dem Leistungsaustausch zwischen Initiator und Participant und der Kontrolle der Leistung durch den Participant inklusive seiner Gegenleistung. Nach Beendigung dieser Phase ist der Leistungsaustausch abgeschlossen.[4]

---

[1] Vgl. [Schulte-Zurhausen 1999: S. 203 ff.]. Auf die unterschiedlichen Bedeutungen des Begriffs *Koordination* wird hier nicht eingegangen. In [Malone 1990] findet sich eine Zusammenstellung verschiedener Definitionen.

[2] Die Beschränkung auf *Markt* (= nicht-hierarchische Koordination) und *Hierarchie* (= hierarchische Koordination) als alternative Koordinationsprotokolle sozioökonomischer Beziehungen ist mit Sicherheit stark vereinfachend. Tatsächlich lassen sich eine Vielzahl unterschiedlicher Koordinationsformen bzw. -muster identifizieren, die Abstufungen zwischen diesen beiden Extremen darstellen (vgl. [Picot 1982: S. 274]). Im Bereich der Transaktionskostentheorie existiert aber keine Typologie möglicher Transaktionsmuster (vgl. [Picot 1982: S. 273]), und es war hier auch nicht die Aufgabe, eine solche zu entwickeln. Aus diesem Grund bleibt es hier bei den zwei Koordinationsprotokollen, die von FERSTL und SINZ verwendet werden. Die Definition weiterer Protokolle ist durch Einführung zusätzlicher Terme möglich. In [Picot 1990] skizzieren PICOT und DIETL Erweiterungen der Markt-Hierarchie-Dichotomie.

[3] Die Interaktion über Nachrichten wird von FERSTL und SINZ als *Transaktion* bezeichnet. Dabei werden die Aktivitäten der beteiligten Objekte entweder vollständig oder überhaupt nicht ausgeführt. Vgl. [Ferstl 1998: S. 185]. Sie verbindet immer zwei Objekte bzw. deren Aufgaben durch einen Kommunikationskanal. Vgl. [Ferstl 1994], [Malischewski 1997: S. 20 f.], [Ferstl 1998: S. 61, 185 f.], [Ferstl 1995b: S. 448], [Ferstl 1995: S. 217].

[4] Vgl. [Popp 1994: S. 30], [Ferstl 1998: S. 186].

Eine Alternative besteht darin, die im Bereich der Workflow-Modellierung verwendete Strukturierung auf der Basis von sog. *Sprechakten* bzw. *Action-Workflows* einzusetzen.[1] Im Gegensatz zu den von FERSTL und SINZ verwendeten drei Phasen werden bei der Verwendung des Sprechakt-Ansatzes Geschäftsprozesse in vier Phasen zerlegt, indem die Phase der Durchführung in eine Leistungs- und eine Akzeptanzphase zerlegt wird. Diese vier Phasen werden als Aktivitäten angesehen, die iterativ durchlaufen werden können.[2]

- *Anbahnung*: Mit der Phase der Anbahnung oder Vorbereitung beginnt ein Zyklus. Dies kann eine Anfrage eines Kunden oder ein Angebot eines Anbieters sein.[1]
- *Vereinbarung*: Im Rahmen der Vereinbarungsphase einigen sich Initiator und Participant auf die Bedingungen, unter denen eine Leistung zu erbringen ist. Dies umfasst die Eigenschaften der zu erbringenden Leistung, die Gegenleistung und alle übrigen Bedingungen. Einigung ist dann erreicht, wenn Initiator und Participant die Bedingungen des anderen akzeptieren. Die Vereinbarungsphase besteht unter Umständen aus mehreren Interaktionen, in denen Initiator und Participant Angebote und Gegenangebote machen, die angenommen oder abgelehnt werden.
- *Leistung*: In der Leistungsphase werden die vereinbarten Leistungen produziert und geliefert. Dabei sind die in der Vereinbarungsphase abgesprochenen Bedingungen einzuhalten.
- *Akzeptanz*: In der Akzeptanzphase wird die gelieferte Leistung geprüft, ggf. angepasst und akzeptiert oder abgelehnt. Auch die vereinbarte Gegenleistung findet hier statt. Kontrollen der Leistung und Anpassungen gehören zur Akzeptanzphase.

In der Methode SOM ist eine Transaktion zwischen zwei Objekten mit der Durchführungsphase abgeschlossen. Wenn man die Abnahme oder Ablehnung einer Leistung durch den Leistungsempfänger explizit abbilden will, ist die Zerlegung in vier Phasen deutlicher. Ungewöhnlich ist nur, dass nun die Gegenleistung des Leistungsempfängers nicht zur Leistungsphase, sondern zur Akzeptanzphase gehört. Erklärt zum Beispiel der Käufer eines Gutes durch seine Gegenleistung (Bezahlung) die Akzeptanz der Ware, wird dies als Akzeptanz abgebildet.

---

[1] Vgl. [Austin 1994] und [Winograd 1986]. Zur Analyse von Geschäftsprozessen auf der Basis von Sprechakten existierten eine Vielzahl von Veröffentlichungen und auch kommerzielle Werkzeuge. Der interessierte Leser sei auf folgende Veröffentlichungen verwiesen: [Scherr 1993], [Denning 1992], [Medina-Mora 1992], [Elgass 1993], [Schäl 1996], [Winograd 1987]. Im Unterschied zur Methode SOM ist die Sprechakt-Theorie aber kein objektorientierter Modellierungsansatz, sondern stammt aus der Linguistik. Die Adaption des Sprechakt-Ansatzes in einen objektorientierten Ansatz ist meines Wissens hier neu.

[2] Die Bezeichnungen variieren in den unterschiedlichen Veröffentlichungen. Für weitere Details siehe [Schäl 1996: S. 31 ff.].

Diese Koordinationsprotokolle werden durch das Prädikat `Coord(`*`Coordination Identifier`*`)` der Klasse *INTERACTION* abgebildet. Als vordefinierte Terme werden `request`, `commitment`, `performance` und `evaluation` verwendet.[2] Das folgende Beispiel zeigt in vereinfachter Form die Interaktionen zwischen einem Kunden und einem Lieferanten. An den Interaktionen A und B wird dabei deutlich, dass die einzelnen Phasen durchaus aus mehreren Interaktionen bestehen können:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :Person	fragt nach Produkt :Action	A :Interaction Coord(request)	:Verkauf	nimmt Anfrage entgegen :Operation
	:Verkauf	bietet Leistung an :Action	B :Interaction Coord(request)	/Kunde :Person	akzeptiert Angebot :Operation
	:Verkauf	sendet Kaufvertrag :Action	C :Interaction Coord(commitment)	/Kunde :Person	erhält Kaufvertrag :Operation
	:Versand	sendet Ware :Action	D :Interaction Coord(performance)	/Kunde :Person	erhält Ware :Operation
	/Kunde :Person	bezahlt Ware :Action	E :Interaction Coord(evaluation)	:Auftrags-bearbeitung	gleicht offenen Posten aus :Operation

Skript 46: Abbildung der Koordination in Interaktionen

Diese nicht-hierarchische Koordination wird insbesondere verwendet, um die einzelnen Phasen von Geschäftsprozessen zu unterscheiden. Durch die Interaktionstypen *Steuerung* und *Kontrolle* wird dagegen die hierarchische Koordination (als *Regelung* bezeichnet) zwischen zwei Objekten abgebildet, von denen eines ein lenkendes Objekt, das andere ein ausführendes Objekt darstellt.[3]

- *Steuerung* (`control`): Durch Steuerung wird ein Regelstreckenobjekt durch ein Reglerobjekt gesteuert. Das bedeutet hier, dass der Initiator dem Participant eine Anweisung erteilt, die dieser auszuführen hat.
- *Kontrolle* (`feedback`): Durch Kontrolle erfolgt eine Kontrollmeldung vom Regelstrecken-objekt an das Reglerobjekt. Das Regelstreckenobjekt informiert das Reglerobjekt über ein bestimmtes Ergebnis oder einen bestimmten Zustand, welches durch sein Verhalten herbei-

[1] Dies wird als *Push*- oder *Pull-Prinzip* bezeichnet.

[2] Vgl. [Schäl 1996: S. 36 f.].

[3] Vgl. [Ferstl 1995], [Ferstl 1998]. Eine andere mögliche Alternative bzw. Erweiterung besteht darin, die Interaktionen des Informationsprozesses in die vier Phasen *Planung*, *Entscheidung*, *Befehl* und *Kontrolle* zu zerlegen. Diese Einteilung würde sich an dem Phasenmodell orientieren, welches in der Organisationslehre zur Strukturierung des Informationsübertragungsprozesses verwendet wird. Vgl. [Wild 1966: S. 93, 100 f.].

geführt wurde. In der Skriptnotation erfolgt eine Kontrollmeldung vom Initiator an den Participant.

Fehlen die Kontrollinteraktionen, spricht man von einem gesteuerten System, sonst von einem geregelten System.[1] Auf die Verwendung der unterschiedlichen Koordinationsstrukturen wird hier nicht eingegangen, sie sind ausführlich in den Arbeiten von FERSTL und SINZ und in den Veröffentlichungen über Sprechakte dargestellt worden. Ihre Verwendung ist aus zwei Gründen sinnvoll:

1. Bei der Modellierung betrieblicher Anwendungssysteme kann durch die Interaktionstypen Steuerung und Kontrolle unterschieden werden, welches Subsystem die Kontrolle ausübt und welches Subsystem kontrolliert wird. Dies ist beispielsweise bei der Modellierung von Client/Server-Systemen[2] sinnvoll, bei denen ein DBMS von einem Anwendungssystem gesteuert wird.
2. Die Verwendung der Interaktionstypen Anbahnung, Vereinbarung, Leistung und Akzeptanz ist bei der GPM sinnvoll, um die einzelnen Phasen der Interaktion zwischen einem Anbieter und einem Kunden abbilden zu können.

Kombiniert man die Prädikate `Coord()` und `Content()` – also Koordination und Inhalt der Interaktion –, kann man Inhalt und Bedeutung einer Nachricht im Verlauf eines Geschäftsprozesses dokumentieren.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Verkauf	sendet Kaufvertrag :Action	:Interaction Content(:Kaufvertrag, Annahme :Status) Coord(commitment)	/Kunde :Person	erhält Kaufvertrag :Operation

Skript 47: Interaktion im Rahmen eines Geschäftsprozesses

6.2.5 Überwachungsbedingungen

Die in der UML vorgesehene Möglichkeit,[3] Interaktionen in Abhängigkeit von Bedingungen stattfinden zu lassen, wird hier durch das Prädikat `Guard(`*`Constraint Expression`*`)`

[1] Vgl. [Popp 1994: S. 26].

[2] Vgl. [Orfali 1997], [Sims 1994].

[3] Vgl. [Booch 1999: S. 249], [Rumbaugh 1999: S. 338], [OMG 1999: S. 3-117].

abgebildet, um so genannte *Überwachungsbedingungen* in Form logischer Ausdrücke abzubilden.[1] Der Term `Constraint Expression` enthält einen Ausdruck, der sich zu wahr oder falsch evaluieren lassen muss. Solche Überwachungsbedingungen werden zur Modellierung von Fallunterscheidungen und Schleifen verwendet.

6.2.5.1 Fallunterscheidungen als Überwachungsbedingungen

Wenn das Prädikat `Guard()` im Sinne einer Fallunterscheidung verwendet wird,[2] muss die Bedingung erfüllt sein, damit die überwachte Interaktion stattfindet. Im folgenden Beispiel wird bei der Schufa nur dann die Kreditwürdigkeit geprüft, wenn die Kreditsumme höher als 10.000,- GE ist:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	holt Auskunft ein :Action	:Interaction Content(:Kundendaten) Answer(:Kreditwürdigkeit) Guard(Kreditsumme>10.000 GE)	:Schufa	prüft Kreditwürdigkeit :Operation

Skript 48: Beispiel für die Verwendung von Bedingungen bei Interaktionen

Im folgenden Beispiel findet sowohl die Interaktion `Auskunft einholen` wie auch die Interaktion `Kredit genehmigen` statt, wenn die Kreditsumme größer als 40.000 GE ist:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	holt Auskunft ein :Action	Auskunft einholen :Interaction Content(:Kundendaten) Answer(:Kreditwürdigkeit) Guard(Kreditsumme>10.000 GE)	:Schufa	prüft Kreditwürdigkeit :Operation
	:SB	holt Genehmigung ein :Action	Kredit genehmigen :Interaction Content(:Kundendaten) Answer(:Genehmigung) Guard(Kreditsumme>40.000 GE)	:Gruppenleiter	prüft Kredit :Operation

Skript 49: Bedingungen in Interaktionen

[1] Überwachungsbedingungen können für alle Beziehungen des Metatyps *RESPONSIBILITYCOOPERATION* (Interaktionen, Ereignisfluss, den Zugriff auf Eigenschaften und Zustandsübergänge (vgl. Abschnitt 6.3)) verwendet werden.

[2] Vgl. [Hitz 1999: S. 112].

Das Prädikat kann auch verwendet werden, wenn unter unterschiedlichen Bedingungen eine Aktivität des Initiators unterschiedliche Interaktionen zur Folge haben kann:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Alarmanlage	löst Alarm aus :Action	:Interaction Guard(Uhrzeit<18:00 Uhr)	:Pförtner	prüft Alarm :Operation
	:Alarmanlage	löst Alarm aus :Action	:Interaction Guard(Uhrzeit ≥ 18:00 Uhr)	:Wachdienst	prüft Alarm :Operation

Skript 50: Eine Aktivität kann unterschiedliche Interaktionen auslösen

In Skript 50 hat das Ereignis des ausgelösten Alarms unterschiedliche Interaktionen zur Folge. Dagegen bildet Skript 49 zwei voneinander unabhängige Umstände ab.

Es mag Verwunderung hervorrufen, die Bedingung, unter der eine Interaktion ausgeführt wird, als Prädikat der Interaktion abzubilden. Viel intuitiver erscheint es, die Bedingung als Prädikat der Aktion abzubilden. Der Grund hierfür liegt in der verwendeten Abbildung auf das Konzeptuelle Schema. Jede Interaktion wird im Konzeptuellen Schema als separater Knoten abgebildet. Jede einzelne Aktion (identifiziert durch ihren Namen) wird dagegen nur exakt einmal pro Klasse im Konzeptuellen Schema als Knoten angelegt. Daraus ergibt sich: Wenn Interaktionen vom aktuellen Wert einer Bedingung abhängig sind, kann dies folglich nur als Prädikat der Interaktion abgebildet werden.

6.2.5.2 *Überwachungsbedingungen als Schleifen*

In der UML werden durch Überwachungsbedingungen auch wiederholt stattfindende Interaktionen abgebildet.[1] Dabei schreibt die UML kein spezielles Format zur Formulierung der Schleifenbedingung vor.[2] Die Interaktion wird so oft wiederholt, wie die angegebene Überwachungsbedingung wahr ist. Um die Überwachungsbedingung als Schleife kenntlich zu machen, wird sie geklammert und ihr ein Stern (‚*') vorangestellt. Im folgenden Beispiel nimmt der Kundenbetreuer dreimal mit einem Kunden Kontakt auf:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Kundenbetreuer	wählt Nummer des Kunden :Action	:Interaction Guard(*[1…3])	/Kunde :Person	nimmt Anruf entgegen :Operation

Skript 51: Interaktion mit Schleife

Im Gegensatz zur UML wird das Prädikat `Guard()` nur verwendet, um mehrfache Interaktionen zwischen einem Initiator und einem Participant abzubilden. Davon unterschieden werden Interaktionen zwischen einem Initiator und mehreren Participants sowie Interaktionen, die wiederholt in einem bestimmten zeitlichen Abstand (Frequenz) stattfinden. 1-zu-N-Interaktionen eines Initiators mit mehreren Objekten werden in Abschnitt 6.2.12 behandelt, periodisch stattfindende Interaktionen werden in Abschnitt 6.8 beschrieben.

6.2.6 Synchronisation

Zwei oder mehrere Aktivitäten werden als parallel oder nebenläufig bezeichnet, wenn sie gleichzeitig oder voneinander unabhängig durchgeführt werden können. Im engeren Sinne versteht man unter parallelen Aktivitäten eine gleichzeitige bzw. zeitlich überlappende Durchführung. Synchronisation bedeutet die Abstimmung von Aktivitäten im Bezug auf ihre Nebenläufigkeit. Die Synchronisation der Aktivitäten kann durch gegenseitige Beobachtung erfolgen oder durch einen speziellen, mit der Synchronisation beauftragten Überwacher. Der erste Fall wird durch entsprechende Modellprimitive abgebildet, der zweite durch Objekte.[3]

---

[1] Vgl. [Page-Jones 2000: S. 143].

[2] Vgl. [Rumbaugh 1999: S. 338].

[3] Vgl. hierzu Abschnitt 6.11.5.

Bei der Abbildung der Synchronisation durch Modellprimitive existieren unterschiedliche Möglichkeiten, wie sich Initiator und Participant während einer Interaktion verhalten:

- Sie müssen konzeptionell beide gleichzeitig an der Interaktion beteiligt sein, damit diese zu Stande kommt. Die Aktion des Initiators und die Operation der Participants finden konzeptionell gleichzeitig statt. Dabei kann von dem Umstand abstrahiert werden, dass beispielsweise im Rahmen einer Interaktion `Gespräch führen` Initiator und Participant abwechselnd reden und zuhören.
- Die Interaktion besteht darin, dass der Initiator seine Nachricht versendet und der Participant diese zeitverzögert erhält. Die Aktion des Initiators und die Operation der Participants finden versetzt und unter Umständen auch mit großem zeitlichen Abstand statt. Die Interaktion über Email ist ein typisches Beispiel asynchroner Interaktion.
- Der Initiator wartet eine gewisse Zeit auf die Reaktion des Participant, bevor er die Interaktion abbricht.

In der objektorientierten Literatur wird bei der Synchronisation von Aktivitäten die *synchrone, asynchrone, sequentielle, zeitabhängige* und *eingeschränkte* Synchronisation unterschieden:[1]

1. *Asynchrone Interaktion* (`asynchronous`) liegt vor, wenn der Initiator seine Nachricht versendet und danach seine Aktivitäten fortsetzt. Ob, wann und wie der Participant eine Operation ausführt, hat keinen Einfluss auf den Initiator. Die Operation des Participants kann mit zeitlichem Abstand stattfinden. Initiator und Participant sind nicht synchronisiert.[2] Durch

---

[1] Dies ist in unterschiedlichem Umfang und mit unterschiedlichen Bezeichnungen zu finden in [Page-Jones 2000: S. 149], [Rumbaugh 1996b], [Rumbaugh 1999. S. 336], [Shlaer 1992: S. 131], [Embley 1992: S. 172 ff.], [de Champeaux 1993: S. 86 ff.], [Firesmith 1995: S. 246 ff.], [Firesmith 1997: S. 134], [Oestereich 1998: S. 304 f.], [OMG 1999: S. 3-98 ff.], [Hitz 1999: S. 111], [Selic 1994: S. 25, 114 ff., 292 ff.], [Buhr 1996: S. 171 ff.], [Schienmann 1997: S. 246], [Reenskaug 1996: S. 58, 65], [Booch 1994: S. 269], [Booch 1999: S. 212 f., 314 f.], [Douglas 1999: S. 50], [Page-Jones 1994b]. Auf die von DE CHAMPEAUX erwähnte *präemptive Interaktion* [de Champeaux 1993: S. 86 ff., 356, 369] wird nicht eingegangen, da diese die Unterbrechung jeder Operation zu jedem Zeitpunkt ermöglicht und dadurch Aussagen über den Zustand, den eine Operation herbeiführt, erschwert. Es sei darauf hingewiesen, dass im Bereich der parallelen Programmierung außerdem zwischen blockierender und nicht-blockierender Kommunikation und bei der Pufferung zwischen asynchroner und synchroner Kommunikation unterschieden wird. Vgl. [Rechenberg 1999: S. 590 f.]. Diese Differenzierung ist nach Kenntnis des Verfassers in allen objektorientierten Modellierungsansätzen unbekannt. Weitere Probleme der Synchronisation sind nicht Gegenstand dieser Arbeit. Der Leser sei hierfür beispielsweise auf [Wettstein 1993: S. 129 ff.], [Milner 1989] und [Lockemann 1993] verwiesen.

[2] Vgl. [Firesmith 1995: S. 33].

diese fehlende Synchronisation ist dem Initiator nicht bekannt, ob und wann der Participant auf das Ereignis reagiert.[1]

2. *Synchrone Interaktion* (`synchronous`) liegt vor, wenn der Initiator den Participant auffordert, Aktivitäten zu zeigen, und wartet, bis der Participant die Interaktion aufnimmt. Initiator und Participant sind während der Interaktion synchronisiert.[2] Der Initiator ist von der weiteren Ausführung von Aktivitäten blockiert, wenn der Participant die Nachricht angenommen hat und mit der Ausführung seiner Operation beginnt. Durch synchrone Interaktion wird bidirektionale Kommunikation ermöglicht. Auch die wechselseitige Interaktion, wie sie im Rahmen eines Gesprächs stattfindet, kann als synchrone Interaktion angesehen werden. Durch diese Synchronisation „weiß" der Initiator, dass der Participant seine Operation ausführt.

3. *Sequentielle Interaktion* (`sequential`) ist ein besonderer Begriff aus der UML. Nur ein Objekt ist aktiv, während alle übrigen passiv sind. Das aktive Objekt besitzt die Kontrolle im Kontrollstrang (engl. *thread*). Der Initiator übergibt die Kontrolle über den Kontrollstrang an den Participant und ist inaktiv, bis er sie vom Participant zurück erhält. Sequentielle Interaktion wird in der UML als besondere Form der synchronen Interaktion angesehen.

4. Die *zeitabhängige Interaktion* (`timeout`) ist nach DE CHAMPEAUX eine Form der synchronen Interaktion.[3] Wenn der Participant die Nachricht nicht innerhalb einer bestimmten Zeitdauer durch die Ausführung seiner Operation verarbeitet, bricht der Initiator die Interaktion ab und setzt seine Aktivitäten fort.

5. In ähnlicher Weise wird auch die *eingeschränkte Interaktion* (`balking`) eingeordnet. Wenn der Participant die Nachricht nicht sofort durch Ausführung seiner Operation annimmt, bricht der Initiator die Interaktion ab. Beide Fälle werden durch den Wert `timeout` bestimmt. Zur Abbildung der zugehörigen Frist wird in Abschnitt 6.2.7 das Prädikat `Suspend()` definiert.

6. Im Falle einer *nicht spezifizierten Synchronisation* (`unspecified`) wird keine Aussage gemacht, wie sich der Initiator verhält, wenn er mit dem Participant interagiert. Dies ist der Standardwert.

Das Prädikat `Concurrency()` (*Synchronisation* oder *Nebenläufigkeit*) wird verwendet, um diese Form der Interobjektparallelität der Aktivitäten von Initiator und Participant abzubilden. Da es sich

[1] Vgl. [de Champeaux 1993: S. 87], [Page-Jones 1994b].

[2] Vgl. [Firesmith 1995: S. 33].

[3] [de Champeaux 1993: S. 89].

um die Nebenläufigkeit der Interaktion handelt, wird sie als Prädikat in der Tabellenspalte Connection durch das Prädikat `Concurrency(Concurrency Kind)` abgebildet.[1]

Synchrone Interaktionen werden beispielsweise verwendet, wenn die Aktion des Initiators und die Operation des Participants zeitgleich beginnen bzw. wenn die Zeitdifferenz zwischen den beiden Aktivitäten zu vernachlässigen ist:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Sprecher :Person	spricht :Action	:Interaktion Concurrency(synchronous)	/Hörer :Person	hört zu :Operation

Skript 52: Beispiel für synchrone Interaktion

Bei der Modellierung von Softwaresystemen ist im Fall synchroner Interaktion Initiator und Participant an die Interaktion „gebunden".[2] Das bedeutet, dass der Initiator so lange von der weiteren Ausführung von Aktivitäten blockiert ist, bis der Participant seine Operation beendet hat. Allerdings wird hier für synchrone Interaktion nicht verlangt, dass die Aktivitäten von Initiator und Participant zeitgleich beginnen. Synchrone Interaktion wird auch dann verwendet, wenn die Übertragung der Nachricht Zeit benötigt.

In allen objektorientierten Ansätzen wird davon ausgegangen, dass durch die Beendigung der Operation des Participants eine Nachricht an den Initiator übertragen wird, durch die dessen Blockierung aufgehoben wird. Diese Benachrichtigung wird nicht explizit abgebildet, sondern geht implizit aus dem synchronen Charakter der Interaktion hervor.[3]

Asynchrone Interaktionen werden beispielsweise verwendet, wenn der Initiator nicht auf die Reaktion des Participants wartet. Nachdem der Initiator seine Aktion ausgeführt hat, kann er weitere Aktivitäten ausführen:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	versendet Angebot :Action	:Interaction Content(:Angebot) Concurrency(asynchronous)	/Kunde :Person	nimmt Angebot an :Operation

Skript 53: Skript mit asynchroner Interaktion

[1] *ACTION* und *OPERATION* erben es von der Klasse *ACTIVITY*.

[2] Asynchrone Interaktion lässt sich durch synchrone Interaktion mit Kommunikationspuffern modellieren. Vgl. [de Champeaux 1993: S. 87], [Saake 1993: S. 92].

[3] Auf diesen Umstand geht DE CHAMPEAUX ein, wenn er schreibt, dass synchrone Interaktion auch durch Benachrichtigungen (*Acknowledgments*) und Rückrufe (*Callbacks*) realisiert werden kann.

Sequentielle Interaktion ist der Normalfall in Softwaresystemen ohne Nebenläufigkeit.[1] Es existiert nur ein Kontrollstrang, über den jeweils ein Objekt die Kontrolle besitzt. Der Initiator übergibt durch die sequentielle Interaktion die Kontrolle an den Participant. Er erhält sie implizit zurück, wenn die Operation des Participants und alle geschachtelten Interaktionen beendet sind, oder explizit durch eine Interaktion. Bei der Beschreibung der Interaktion zwischen einem Softwaresystem und seinen Anwendern, außerhalb eines Softwaresystems oder bei Softwaresystemen mit mehreren Kontrollsträngen spielt die sequentielle Interaktion keine Rolle, da von Interaktionen zwischen Objekten ausgegangen wird, die jeweils einen eigenen Kontrollstrang besitzen.

6.2.7 Befristung

Zeitabhängige Interaktion (abgebildet durch `Concurrency(timeout)`) wird verwendet, wenn der Initiator den Interaktionsversuch nach einer bestimmten Zeit abbricht, falls der Participant nicht reagiert. Dadurch kann verhindert werden, dass im Fall synchroner Interaktion ein nicht reagierender Participant den Initiator auf unbestimmte Zeit blockiert und der weitere Verlauf eines Prozesses behindert wird. Die UML bietet für dieses Problem keine Sprachmittel an.

Das hierfür verwendete Prädikat `Suspend(`*`Time Expression, Point Of Reference`*`)` drückt aus, dass die entsprechende Interaktion abgebrochen wird bzw. nicht stattfindet, wenn nicht innerhalb der durch `Time Expression` formulierten Frist die Operation des Participants getriggert wird. Die Frist beginnt mit dem durch den *Bezugspunkt* (`Point Of Reference`) angegebenen Zeitpunkt. Seine Angabe ist optional. Ihr Vorgabewert ist der Zeitpunkt, in dem beim Participant die Nachricht eintrifft, d. h. `Interaction.receiveTime()`.[2]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	ruft Kunden an :Action	:Interaction Concurrency(timeout) Suspend(15 s)	/Kunde :Person	wird angerufen :Operation

Skript 54: Zeitlich begrenzte Interaktion (1)

Auf diese Weise wird auch ausgedrückt, dass nach einem bestimmten Zeitraum die Reaktion des Participants nicht mehr möglich ist. Wenn im folgenden Beispiel die Person nicht innerhalb von einer Sekunde die Interaktion durchgeführt hat (d. h. das Glas auffängt), findet sie nicht mehr statt,

[1] Vgl. [Kredel 1999], [Lea 1997].

[2] Vgl. Abschnitt 6.2.14, S. 271.

da das Glas zu Boden gefallen ist. Auf diese Weise lassen sich Ereignisse definieren, die nach Ablauf einer Frist ungültig werden, d. h. auf die der Participant nicht mehr reagieren kann.[1]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Glas	fällt vom Tisch :Action	:Interaction Concurrency(timeout) Suspend(1 s, fällt vom Tisch.startTime())	:Person	fängt Glas auf :Operation

Skript 55: Zeitlich begrenzte Interaktion (2)

Die Möglichkeit, den Bezugspunkt durch das zweite Argument des Prädikats `Suspend()` frei zu wählen, ermöglicht eine präzise und realistische Abbildung der Situation. In Skript 55 ist es unerheblich, wann der Participant bemerkt, dass das Glas vom Tisch fällt. Einzig die Tatsache, wann der Fall begonnen hat, ist erheblich für den Umstand, ob die Person überhaupt reagieren kann. Um dies abzubilden, muss aber die Möglichkeit bestehen, den Beginn der Aktion als Bezugspunkt der Befristung zu referenzieren.

Auch eingeschränkte Interaktion wird durch das Prädikat `Suspend()` abgebildet. Das folgende Skript zeigt eine Interaktion, die abgebrochen wird, wenn der Participant beim Eintreffen der Nachricht nicht sofort reagiert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Ein Objekt :X	benachrichtigt :Action	:Interaction suspend(0 s, receiveTime())	Ein anderes Objekt :Y	reagiert :Operation

Skript 56: Eingeschränkte Interaktion

6.2.8 Benachrichtigung

Die synchrone Interaktion kann dazu führen, dass der Initiator von der weiteren Ausführung von Aktivitäten blockiert ist, weil der Participant die angeforderte Operation nicht ausführt. Dafür ist er auf der anderen Seite sicher, dass nach Abschluss der Interaktion die Operation tatsächlich ausgeführt wurde. Durch asynchrone Interaktion wird die Gefahr der Blockierung des Initiators verhindert. Dafür entsteht aber potentiell das Problem, dass der Participant die Operation überhaupt nicht ausführt und der Initiator dies nicht erfährt. Diesem Problem wird hier dadurch begegnet, dass

[1] Solche Phänomene abbilden zu können ist insbesondere bei der Entwicklung von Echtzeitsystemen wichtig.

man bei asynchroner Interaktion eine Bestätigung an den Initiator der Interaktion versenden kann, dass der Participant die Ausführung der Operation begonnen hat.[1] Die UML bietet für dieses Problem keine Sprachmittel an. Durch eine *Benachrichtigung* (*Acknowledgement*) wird der Initiator darüber in Kenntnis gesetzt, dass die Ausführung der Operation des Participants begonnen hat. Zwei Effekte werden durch Benachrichtigungen erzielt:

- Bei synchroner Interaktion ist der Initiator der Interaktion nur blockiert, bis der Participant eine Benachrichtigung versendet. Danach ist die Synchronisation aufgehoben, und er kann weitere Aktivitäten ausführen.
- Bei asynchroner Interaktion ist der Initiator darüber informiert, dass der Participant die Operation begonnen hat.

Durch die Benachrichtigung wird der Participant einer Interaktion aufgefordert, asynchron eine Nachricht an den Initiator zu versenden. Diese Benachrichtigung beendet nicht die Operation des Participant[2] und sollte nicht mit dem Rückgabewert einer Operation verwechselt werden. Solche Benachrichtigungen werden durch das Prädikat `Reply(`*`Acknowledgement Identifier`*`)` abgebildet. Der `Acknowledgement Identifier` ist entweder `Ack` (für *`Ack`*nowledgment) oder eine andere Textkette, die der Initiator als Benachrichtigung wünscht. Die Vorgabe ist `Ack`. Die Spezifikation von `Reply()` hat zur Folge, dass implizit eine Interaktion vom Participant zum Initiator stattfindet, ohne dass diese explizit abgebildet werden muss. Im Fall von `Reply(Ack)` findet diese Interaktion in Gegenrichtung in dem Moment statt, in dem die Operation des Participant getriggert wird. In allen anderen Fällen ist es dem Participant überlassen, wann welche Benachrichtigung versandt wird. Im folgenden Beispiel wird die Telefonzentrale nicht darüber informiert, wann die Nachricht bei der Abteilung Hotline eingetroffen ist, sondern wenn sie die Beschwerde entgegengenommen hat.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Telefonzentrale	versendet :Action	:Interaction Concurrency(asynchronous) Content(:Beschwerde) Reply(Ack)	:Hotline	nimmt entgegen :Operation

Skript 57: Asynchrone Interaktion mit Benachrichtigung

[1] Vgl. [de Champeaux 1993: S. 88].

[2] [de Champeaux 1993: S. 374 ff.].

6.2.9 Rückruf

Durch einen Rückruf (*Callback*) ruft der Participant eine Operation des Initiators auf, die ihm dieser mitgeteilt hat.[1] Die UML bietet dafür keine Sprachmittel an. Der Initiator einer Operation teilt dem Participant mit, wie er zurückgerufen werden soll. Auch die Spezifikation eines Rückrufs hat zur Folge, dass implizit eine Interaktion vom Participant zum Initiator stattfindet, ohne dass diese explizit abgebildet werden muss. Ein Rückruf wird durch das Prädikat `Callb(`*`Operation Name`*`)` bestimmt, wobei der verwendete Term der Name einer Operation ist, die der Initiator ausführen soll. Die Verwendung soll an einem Beispiel demonstriert werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	will geweckt werden :Action	:Interaction Content(:Uhrzeit) Concurrency(asynchronous) Callb(anrufen :Operation)	:Auftragsdienst	erhält Weckauftrag :Operation

Skript 58: Interaktion mit Rückruf

6.2.10 Modus der Interaktion

Ob im Rahmen der Interaktion unidirektional oder bidirektional Informationen ausgetauscht werden, wird durch die Prädikate `Content()` und `Answer()` ersichtlich. Bei der Unterscheidung von unidirektionaler und bidirektionaler Interaktion spielen jedoch eine Reihe zusätzlicher Konnotationen eine Rolle. JACOBSON unterscheidet in der Methode OOSE *Nachrichten* und *Signale*.[2] Nur Nachrichten stellen eine bidirektionale Interaktion dar. Zusätzlich wird auf diese Weise aber auch unterschieden, ob es sich auf softwaretechnischer Ebene um Interprozesskommunikation oder Intraprozesskommunikation handelt.[3] Diese Unterscheidung ist hier aber von geringem Interesse, da man in dem Moment, in dem man die Betrachtungsebene eines einzelnen Softwaresystems verlässt, von separaten Prozessen bzw. Kontrollsträngen[4] und damit von Interprozesskommunikation ausgehen wird. Bei JACOBSON wird zwischen Signal und Nachricht unterschieden, um im Falle von Interprozesskommunikation die fehlende Synchronisation zwischen Prozessen durch ein Signal abzubilden. Dies wird hier schon durch das Prädikat

[1] [de Champeaux 1993: S. 88, 369], [Page-Jones 2000: S. 151].

[2] Vgl. [Jacobson 1994: S. 220 ff.].

[3] Eine Nachricht ist in diesem Fall mit dem Aufruf einer Methode in einer objektorientierten Programmiersprache, ein Signal ist mit einem *Remote Procedure Call* (RPC) zu vergleichen.

[4] Vgl. Abschnitt 6.2.6.

`Concurrency()` erfasst. Dadurch wird auch festgelegt, ob sich der Initiator einer Interaktion passiv verhält, bis er eine Antwort vom Participant erhält. RUBIN und GOLDBERG differenzieren nun vier verschiedene Interaktionstypen, die sie als die *Vertragsformen* zwischen Objekten bezeichnen:[1]

- Die Inkenntnissetzung (*Notification*) wird verwendet, wenn der Initiator den Participant über einen Umstand informiert.
- Die Informationsübertragung (*Provide Information*) wird verwendet, um die Übertragung von Informationen vom Initiator zum Participant anzuzeigen.
- Die Informationsanfrage (*Request Information*) stellt eine Anfrage nach Informationen dar, die vom Initiator ausgeht. Ein zweiter Informationsfluss findet in umgekehrter Richtung vom Participant zum Initiator statt. Dies wird als bidirektionale Interaktion angesehen.
- Die Leistungsanforderung (*Request Service*) stellt schließlich die Aufforderung des Initiators an den Participant dar, eine bestimmte Dienstleistung auszuführen.

Auch wenn der Vertragsbegriff hier anders verwendet wird,[2] stellt diese Unterscheidung eine sinnvolle Differenzierung dar, um unabhängig von technischen Aspekten wie Synchronisation oder uni- vs. bidirektionaler Interaktion zu erfassen, mit welcher Erwartung der Initiator mit dem Participant eine Interaktion aufnimmt. Eventuell bestehende Synchronisationsbedingungen werden hier nicht beachtet, da sie durch das Prädikat `Concurrency()` erfasst werden.

Im Falle der Informationsanfrage erwartet der Initiator der Interaktion, dass der Participant ihm Informationen liefert. Dabei spielt es keine Rolle, ob auch der Initiator an den Participant Informationen überträgt.

Durch eine Leistungsanforderung wird ein Objekt aufgefordert, seinen Zustand oder den anderer Objekte zu verändern. Wenn der Initiator eine Leistung vom Participant anfordert, ist die Übertragung von Informationen möglich, aber nicht zwingend. Der Initiator erhält nicht unbedingt Informationen zurück, geht aber davon aus, dass der Participant in irgendeiner Weise den Zustand des Systems bzw. eines seiner Elemente verändert.[3]

---

[1] Vgl. [Rubin 1992]; ähnlich auch zu finden in [Firesmith 1993]. PAGE-JONES unterscheidet nur drei Möglichkeiten, indem er die *Notification* und *Provide Information* unter dem Begriff *Informative Message* zusammenfasst [Page-Jones 2000: S. 25 ff.]. Behringer unterscheidet nur zwei Formen: *Request* und *Notification*. Vgl. [Behringer 1997: S. 13].

[2] Vgl. Abschnitt 6.6.9.

[3] In welcher Weise diese Systemveränderung stattfinden soll, wird an anderer Stelle über die Semantik der Operation definiert, die der Participant ausführt. PAGE-JONES grenzt diesen Modus vom *Request Information* (dort: *Interrogative Message*) durch den Umstand ab, dass der Initiator davon ausgeht, dass

Bei der Informationsübertragung handelt es sich um eine unidirektionale Interaktion, da der Initiator keine Antwort erwartet. Er geht aber mindestens davon aus, dass der Participant die Informationen entgegennimmt.

Bei der Inkenntnissetzung informiert der Initiator den Participant ohne weitere Erwartung über einen Umstand. Diese Inkenntnissetzung kann auch Informationen beinhalten, die vom Initiator an den Participant übertragen werden. Wenn daraufhin Informationen vom Participant an den Initiator übertragen werden, handelt es sich nicht um eine Inkenntnissetzung, sondern um eine Informationsanfrage.

Diese Fälle werden als *Modus* der Interaktion bezeichnet. Dieser Modus wird durch das Prädikat `Mode(Interaction Mode)` abgebildet, welches für alle von *COOPERATION* erbenden Beziehungen definiert ist. Die Terme `Notification` und `Request Service` werden als Terme für den Interaktionsmodus übernommen. An Stelle der Bezeichnungen *Provide Information* und *Request Information* werden die Terme `Supply` und `Request` verwendet, da es sich nicht unbedingt um Informationsströme handeln muss.[1] Das folgende Skript stellt die verschiedenen Absichten im Zusammenhang von Interaktionen dar:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
<Initiator>	setzt in Kenntnis :Action	:Interaction Mode(Notification)	<Participant>	wird in Kenntnis gesetzt :Operation
<Initiator>	liefert etwas :Action	:Interaction Mode(Supply)	<Participant>	erhält etwas :Operation
<Initiator>	will etwas erhalten :Action	:Interaction Mode(Request)	<Participant>	liefert etwas :Operation
<Initiator>	wünscht Dienstleistung :Action	:Interaction Mode(Request Service)	<Participant>	erbringt Dienstleistung :Operation

Skript 59: Modi für Interaktionsbeziehungen

Skript 60 demonstriert die Anwendung. Der Kunde fragt in der ersten Zeile des Skripts nach Informationen, die der Vertrieb ihm in der zweiten Zeile liefert.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	fordert Katalog an :Action	:Interaction Mode(Request)	:Vertrieb	erhält Anforderung :Operation
:Vertrieb	sendet Katalog :Action	:Interaction Mode(Supply)	/Kunde :Person	erhält Katalog :Operation

---

eine gewisse Zeit vergehen wird, in der der Participant Aktivitäten ausübt, wogegen die Informationsanfrage eine zeitnahe Reaktion des Participants darstellt, da dieser keine Aktivitäten ausübt. Vgl. [Page-Jones 2000: S. 26].

[1] Vgl. Abschnitt 6.2.1.

			Content(:Katalog)		

Skript 60: Interaktion mit Modus (1)

Im folgenden Skript ist diese Abfolge von zwei Interaktionen in einer bidirektionalen Interaktion zusammengefasst:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	fordert Katalog an :Action	:Interaction Mode(Request) Answer(:Katalog)	:Vertrieb	sendet Katalog :Operation

Skript 61: Interaktion mit Modus (2)

Die folgende Tabelle verdeutlicht den Zusammenhang zwischen dem Fluss und dem Modus der Interaktion:

Modus	**Fluss**	
	Initiator → Participant	**Initiator ← Participant**
Inkenntnissetzung (*Notification*)	wahlweise	nein
Übertragung (*Supply*)	ja	nein
Anfrage (*Request*)	wahlweise	ja
Anforderung (*Request Service*)	wahlweise	wahlweise

Tabelle 9: Zusammenhang zwischen dem Modus der Interaktion und dem Fluss

6.2.11 Kontinuität

Interaktion über Nachrichtenaustausch wird in der objektorientierten Modellierung als Übertragung diskreter Pakete verstanden. Sowohl in der UML wie auch im SOM wird davon ausgegangen, dass auch kontinuierliche Flüsse durch eine Folge von diskreten Paketen abgebildet werden.[1] Nur wenige Ansätze erwähnen zumindest die Möglichkeit einer kontinuierlichen Interaktion.[2] Das Prädikat Stetigkeit (`Continuity( Continuity Identifier)`) wird verwendet, um den kontinuierlichen (`continuous`) vom diskreten (`discrete`) Fluss vom Initiator zum Participant unterscheiden zu können. Folgende vordefinierte Terme werden für die Kontinuität verwendet:

[1] Vgl. [Ferstl 1998: S. 60].

[2] Vgl. [Firesmith 1995: S. 246], [Embley 1992: S. 186 f.].

- `continuous`: Jede Form kontinuierlich fließender Nachrichten.
- `discrete`: Jede Form diskret fließender Nachrichten in Form von Paketen. Dies ist der Standardfall.

Kontinuierliche Interaktion tritt üblicherweise im Zusammenhang mit synchroner Interaktion auf. Das Fließen von elektrischem Strom ist ein typisches Beispiel:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Notstromaggregat	produziert Strom :Action	:Interaction Concurrency(synchronous) Continuity(continuous)	:Abnehmer	konsumieren Strom :Operation

Skript 62: Skript mit stetiger Interaktion

6.2.12 Ausbreitung

Die Tatsache, dass ein Objekt mit mehreren anderen Objekten interagiert, erfüllt in der realen Welt niemanden mit Erstaunen:

- Ein Redner hält einen Vortrag und interagiert mit einer Menge von Zuhörern.
- Ein Trainer einer Mannschaft erteilt den Spielern einer Mannschaft Anweisungen.
- Die Vertriebsabteilung schreibt alle Kunden an.
- Ein Sachbearbeiter telefoniert mit unzufriedenen Kunden.

In der objektorientierten Modellierung ist bis heute ungeklärt, ob die Objektinteraktion als eine Punkt-zu-Punkt oder eine 1-zu-N-Beziehung betrachtet werden soll.[1] In der Realität existieren beide Formen, was bei der GPM und der Entwicklung von Softwaresystemen zu berücksichtigen ist. Aus diesem Grund soll die Abbildung beider Formen unterstützt werden. Ob ein Initiator mit einem oder mehreren Participants interagiert, wird im Folgenden als *Ausbreitung* der Interaktion bezeichnet.

Wie viele objektorientierte Modellierungsansätze bietet die UML für 1-zu-N-Interaktionen keine Konstrukte an.[2] Stattdessen wird in Kollaborationsdiagrammen im Sequenzausdruck einer

[1] Vgl. [Cook 1994b].

[2] Vgl. [Booch 1999: S. 282]. PAGE-JONES verdeutlicht die Interaktion mit einer Menge von Objekten graphisch durch ein spezielles Symbol. Vgl. [Page-Jones 1994]. COOK und DANIELS gehen den

Nachricht durch Formulierung einer Schleifenanweisung angedeutet, dass die Interaktion mit mehreren Empfängerobjekten stattfindet.[1] Durch das Symbol „||“ wird die Tatsache abgebildet, dass die Interaktion parallel stattfinden soll. Wenn mehrere Objekte eine Nachricht erhalten, wird dies durch so genannte *Multiobjekte* abgebildet.[2] In Sequenzdiagrammen ist die Modellierung dieser Situation überhaupt nicht vorgesehen. FUSION bietet Konstrukte, mit denen die Interaktion von einem Objekt mit einer Menge von Objekten abgebildet werden kann.[3] In OML wird die 1-zu-N-Interaktion in Ganzes-Teil-Beziehungen graphisch durch unterschiedliche Markierungen an den Interaktionstypen abgebildet.[4] Folgende Situationen werden hier unterschieden:[5]

- Direkte Interaktion: Der Initiator interagiert mit exakt einem Objekt. Dabei handelt es sich unter Umständen um ein spezielles Objekt, welches durch eine Selektionsanweisung eindeutig identifiziert wird. Auf diese Weise wird ausgedrückt, dass ein Sachbearbeiter mit einem Kunden interagiert, der den Namen „Franz Müller“ besitzt. Dies wird auch als *Unicast* bezeichnet.
- Der Initiator interagiert mit einer Menge von Objekten. Dabei handelt es sich unter Umständen um die Teilmenge einer Menge von Objekten, welche durch eine Selektion zu identifizieren sind. Dies wird als *Multicast* bezeichnet.
- Eine weitere Verallgemeinerung stellt der *Rundruf* bzw. *Broadcast* dar. Dabei wird davon ausgegangen, dass die Interaktion mit allen Objekten einer irgendwie gearteten Gesamtheit stattfindet.
- Das Objekt interagiert mit sich selbst.[6] Dieser Fall ist bei der objektorientierten Modellierung nicht unwichtig, da durch Selbstinteraktion[7] Objekte ihre eigenen Operationen aufrufen. Durch

---

umgekehrten Weg und setzen Nachricht und Ereignis gleich. Jedes Objekt kann auf jedes Ereignis reagieren. Vgl. [Cook 1994].

[1] Vgl. [Hitz 1999: S. 120 f., 123].

[2] Vgl. [OMG 1999: S. 3-112].

[3] Vgl. [Coleman 1994: S. 65 ff.].

[4] Vgl. [Firesmith 1997: S. 133 ff.], [Firesmith 1998b].

[5] Vgl. [Page-Jones 2000: S. 157], [de Champeaux 1993: S. 89], [Embley 1992: S. 174 ff.]. Vgl. im Bereich der Telekommunikationssysteme [Lockemann 1993: S. 51], für den Bereich der Betriebssysteme [Tanenbaum 1994: S. 543] und für den Bereich der parallelen Programmierung [Rechenberg 1999: S. 591].

[6] Vgl. [Stoyan 1991: S. 183], [Saake 1993: S. 18], [Page-Jones 1994].

[7] Auch als *Selbst-Delegation* bezeichnet. Das Objekt beauftragt sich selbst mit der Ausführung einer Operation.

den symbolischen Namen *Self*[1] identifiziert sich ein Objekt selbst, durch *Super* wird eine Nachricht an eine übergeordnete Ebene der Vererbungshierarchie gesandt.

Die unterschiedlichen Formen der Verbreitung werden hier durch das Prädikat *Propagation* (engl.: Ausbreitung; abgekürzt `Prop`) festgehalten. Die Spezifikation der Ausbreitung folgt der Form `Prop(`*`Propagation Kind, Selector Expression, Order Kind`*`)`. Das erste Argument (*`Propagation Kind`*) gibt durch die vordefinierten Terme `Unicast`, `Multicast` und `Broadcast` die Art der Ausbreitung an, wobei die Vorgabe `Unicast` ist. Das zweite Argument (*Selector Expression*) gibt für Unicast- und Multicast-Interaktion (z. B. durch ein Selektionskriterium) an, mit welchen bzw. mit wie vielen Objekten interagiert wird. Im Falle der Unicast- und Broadcast-Interaktion ist dieses Selektionskriterium implizit TRUE. Im Falle der Selbst-Interaktion ist es `Self`.[2] Das dritte Argument (*`Order Kind`*) gibt an, ob die Interaktion konzeptionell als `par`(-allel) oder `seq`(-uentiell) angesehen wird. Die Vorgabe ist `par`. Ist die Anzahl der Participants gleich eins, ist dieses Argument ohne Bedeutung. Das folgende Skript bildet den Umstand ab, dass ein Sachbearbeiter allen Kunden mit der Postleitzahl 40000 einen Brief schickt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	schickt Brief :Action	:Interaction Content(:Brief) Prop(Multicast, Person.PLZ = 40000, par)	/Kunde :Person	erhält Brief :Operation

Skript 63: Multicast-Interaktion mit Selektionskriterium

Das folgende Skript bildet dagegen den Umstand ab, dass ein Sachbearbeiter einen bestimmten Kunden anruft:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	anrufen :Action	:Interaction Prop(Unicast, Person.Name = Fritz Müller, seq)	/Kunde :Person	erhält Brief :Operation

Skript 64: Unicast-Interaktion mit Selektionskriterium

Durch `Prop(Unicast, Self, seq)` wird eindeutig festgelegt, dass es sich bei Initiator und Participant um die gleichen Objekte handelt:

[1] Vgl. [Page-Jones 1994b]. In der objektorientierten Programmiersprache C++ wird diese Referenz als *This* bezeichnet.

[2] Für diese Selektion kann *nicht* das Prädikat `Guard()` verwendet werden, da damit bestimmt wird, unter welcher Bedingung die Interaktion überhaupt stattfindet.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Herr Müller /Sender :SB	sendet Nachticht :Action	:Interaction Prop(Unicast, Self)	Herr Müller /Empfänger :SB	empfängt Nachricht :Operation

Skript 65: Selbst-Interaktion

Diese Interaktion stellt ein objektinternes Ereignis dar, wie es von FERSTL und SINZ verwendet wird. Solche objektinternen Ereignisse werden in der Methode SOM verwendet, um die Sequenzen von Aktivitäten zu modellieren, die von einem Objekt allein ausgeführt werden.[1] Nun soll ein Sachbearbeiter aus allen Kundendaten jene selektieren, die mit Zahlungen im Verzug sind:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	sucht säumige Kunden heraus :Action	Suche nach säumigen Kunden :Interaction Mode (Request) Content (Person.Status = säumig) Answer(Personendaten) Prop(Broadcast, True, par)	/Kunde :Person	prüft, ob mit Zahlung im Verzug :Operation

Skript 66: Beispiel der Suche nach säumigen Kunden

Der Sachbearbeiter sendet parallel an alle Kundenobjekte (`Prop(Broadcast, True, par)`) die Nachricht, säumige Kunden herauszusuchen (`Suche nach säumigen Kunden :Interaction`). Es handelt sich um eine Nachfrage nach Information (`Mode(Request)`). Die Personenobjekte sollen einen bestimmten Wert verwenden (`Content(Person.Status=säumig)`) und die Personendaten zurück liefern (`Answer(Personendaten)`).

Im Gegensatz dazu wird im nächsten Skript eine Interaktion nur mit bestimmten Objekten (`Prop(Broadcast, Person.Status = säumig, seq)`) aufgenommen. Darüber hinaus geschieht die Prüfung sequentiell. Das bedeutet, dass der Sachbearbeiter einen Kunden nach dem anderen Kunden überprüft.

[1] Vgl. [Ferstl 1994: S. 21], [Ferstl 1994b: S. 2], [Malischewski 1997: S. 23].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	sucht säumige Kunden heraus :Action	Prüfen säumiger Kunden :Interaction Mode (Request) Prop(Broadcast, Person.Status = säumig, seq)	/Kunde :Person	prüft, ob mit Zahlung im Verzug :Operation

Skript 67: Beispiel für das Mahnen säumiger Kunden

Hier wird eine Interaktion mit einer Menge von Objekten aufgenommen, für die eine Bedingung zutrifft. Im Skript 66 dagegen haben alle Objekte eine Nachricht erhalten. Broadcast-Interaktion wird insbesondere dann verwendet, wenn von einem direkten und gezielten Nachrichtenversand zwischen Initiator und Participant nicht gesprochen werden kann, sondern die Interaktion einen eher konzeptionellen Charakter besitzt. COOK und DANIELS nennen als typisches Beispiel die Interaktion zwischen Sonne und Singvögeln.[1] Die Behauptung, die Sonne sende jedem Singvogel eine Nachricht, halten sie für irreführend. Der Sonnenaufgang stellt ein Ereignis bzw. einen Stimulus dar, auf den potentiell alle Vögel reagieren. Dies wird durch die Kennzeichnung der Interaktion als `Broadcast` verdeutlicht:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Sonne	geht auf :Action	:Interaction Prop(Broadcast, True, par)	:Singvogel	singt :Operation
	:Sonne	geht auf :Action	:Interaction Prop(Broadcast, True, par)	:Hahn	kräht :Operation

Skript 68: Skript mit Broadcast-Interaktion

Die Broadcast-Interaktion unterscheidet sich von den anderen Ausbreitungsarten dadurch, dass der Initiator nicht weiß, welche Objekte auf seine Nachricht überhaupt reagieren.[2] Ein weiteres Beispiel für Broadcast-Interaktion ist die kontinuierliche Übertragung des Radioprogramms vom Sender zum Empfänger:[3]

[1] Vgl. [Cook 1994: S. 6, 137].

[2] Vgl. [Page-Jones 1994b].

[3] Vgl. [Embley 1992: S. 175].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Radiosender	strahlt Programm aus :Action	Übertragung des Radioprogramms :Interaction Content(:Radioprogramm) Prop(Broadcast, True, par) Continuity(continuous)	:Radiogeräte	empfangen Programm :Operation

Skript 69: Skript mit kontinuierlicher Broadcast-Interaktion

Der umgekehrte Fall der N-zu-1-Interaktion wird in der objektorientierten Modellierung üblicherweise nicht betrachtet. Diese Situation wird durch mehrere Interaktionen abgebildet werden, für die zusätzlich festgehalten wird, dass sie parallel stattfinden.[1]

Mit der Delegation an `Super` wird der Umstand ausgedrückt, dass eine Operation auf der Operation einer vererbenden Klasse beruht.[2] Als Beispiel soll davon ausgegangen werden, dass alle Mitarbeiter eine Operation `erhalte Weihnachtsgeld` besitzen. Die Objekte der Klasse `Außertarifliche Mitarbeiter` verfügen über eine Operation `erhalte Gratifikation`. Außerdem ist die Klasse `Außertarifliche Mitarbeiter` durch eine Vererbungsbeziehung mit der Klasse `Mitarbeiter` verbunden, d. h., `Außertarifliche Mitarbeiter` sind eine besondere Form (Spezialisierung) von `Mitarbeitern`. Das folgende Skript drückt aus, dass die Operation `erhalte Gratifikation` der Klasse `Außertariflicher Mitarbeiter` auf der Operation `erhalte Weihnachtsgeld` der Klasse `Mitarbeiter` beruht:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	:Außertariflicher Mitarbeiter	erhält Gratifikation :Operation
	:Außertariflicher Mitarbeiter	hat Gratifikation erhalten :Action	:Interaction Prop(Unicast, Super, seq)	:Mitarbeiter	erhält Weihnachtsgeld :Operation
	Außertariflicher Mitarbeiter		Inheritance	Mitarbeiter	

Skript 70: Interaktion mit Super

Auf diese Weise wird eine Pseudointeraktion „aufwärts“ entlang der Vererbungsbeziehung abgebildet.[3] Pseudointeraktion deshalb, weil an der Interaktion nur ein Objekt beteiligt ist. Diese

[1] Vgl. Abschnitt 6.2.14.

[2] Vgl. [Budd 1997: S. 210], [Firesmith 1995: S. 436], [Meyer 1997: S. 1128].

[3] Diese besondere Form der Interaktion ist in den verbreiteten Modellierungsansätzen unbekannt. Nur in [Baklund 1995] wird auf diese Möglichkeit hingewiesen. Sie ist notwendig, um die Situation abbilden zu können, dass nicht nur die Empfängerklasse die Operation ausführt, sondern mindestens teilweise

besondere Semantik der Interaktionsbeziehung beruht auf dem Umstand, dass außertarifliche Mitarbeiter eine besondere Form von Mitarbeitern *sind*. Durch die Semantik der Vererbungsbeziehung ist im OA festgelegt, dass ein Objekt der Klasse `Außertariflicher Mitarbeiter` gleichzeitig ein Objekt der Klasse `Mitarbeiter` *ist*. Die Delegation an `Super` stellt einen Sonderfall dar, der dazu verwendet wird, Operationen in erbenden Klassen aufrufen zu können und die Ausführung an die vererbende Klasse weiterzureichen. Diese Form der Delegation an eine übergeordnete Klasse ist in den Interaktionsdiagrammen der UML unbekannt.[1]

6.2.13 Ausnahmen

Das Eintreten von Ausnahmen bewirkt außergewöhnliche Abläufe,[2] die besonders gekennzeichnet werden sollen. Im Falle von Ausnahmen wird als Aktivität des Initiators eine Exception verwendet. Auf diese Ausnahme wird durch eine Operation reagiert. Ausnahme und reagierende Operation werden durch eine Interaktionsbeziehung verbunden. Interaktionen besitzen ein Prädikat `Kind(`*`Interaction Kind`*`)`, welches die Terme `Exception` oder `Normal` besitzen kann.[3] Die Voreinstellung dieses Prädikats ist `Normal`. Die im Falle von Ausnahmen stattfindenden Interaktionen erhalten den Term `Exception`.. Auf diese Weise werden Interaktionen gekennzeichnet, die nicht als normaler Verlauf in einem System angesehen werden. Dies kann beispielsweise der Fall sein, wenn im Rahmen einer Operation etwas Unvorhergesehenes eintritt, auf das reagiert werden muss:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Lokales Netz	bricht zusammen :Exception	:Interaction Kind(Exception)	:Wartungstechniker	wird informiert :Operation

Skript 71: Abbildung einer Ausnahmebedingung

Es ist im Rahmen der objektorientierten Modellierung nahe liegend, die Reaktion auf Ausnahmen als Interaktionen abzubilden, wenn man beim Entwurf eines Systems von der Maxime ausgeht,

---

Verhalten der vererbenden Klasse verwendet wird. Vgl. hierzu auch die ähnliche Position in [Page-Jones 1994].

[1] Vgl. [OMG 1999]. Es ist überhaupt nur eine Veröffentlichung bekannt, in der die Möglichkeit erwähnt wird, diese Delegation in Interaktionsdiagrammen darzustellen. Vgl. [Baklund 1995].

[2] Das sog. *exception handling*. Vgl. [Firesmith 1995: S. 164].

[3] Dies ist nicht mit dem in der UML definierten Stereotyp *Exception* (vgl. [Rumbaugh 1999: S. 269]) zu verwechseln, welches nicht in Interaktionsmodellen verwendet wird.

dass (a) das Eintreten von Fehlern als Normalfall anzusehen ist, (b) zuverlässige Systeme auf Fehler in einer definierten Art und Weise reagieren können müssen und (c) die Beschreibung der Ausnahmebehandlung von der regulärer Abläufe getrennt werden können muss, damit diese nicht zu unübersichtlich werden.

6.2.14 Zeitliche Restriktionen

Die Erfassung von Anforderungen zeitlicher Natur ist bis heute in verbreiteten Entwicklungsmethoden nur rudimentär entwickelt.[1] Dies gilt selbst für Methoden, die die Entwicklung von Echtzeitsystemen unterstützen.[2] Auch die UML beinhaltet nur ein rudimentäres Gerüst zur Formulierung von zeitlichen Aussagen über das Systemverhalten. Sie werden als *zeitliche Einschränkung* (*timing constraint*)[3] bezeichnet, was eine andere Formulierung für einen Kommentar darstellt, der Anforderungen zeitlicher Natur an ein System beinhaltet. Diese Einschränkungen werden in der Nähe des Modellelements positioniert, für das sie gelten sollen. Inhalt einer solchen Einschränkung ist ein Zeitausdruck (*time expression*). Der Zeitausdruck kann eine absolute oder relative Zeitangabe beinhalten.[4] Restriktionen zeitlicher Natur sollen mit Hilfe der Bezeichner *sending time* und *receiving time* spezifiziert werden. Dafür kann eine Nachricht einen Namen erhalten,[5] was insofern bemerkenswert ist, weil in der UML sonst nur Operationen Namen besitzen und die Nachricht mit dem Namen der Operation gleichgesetzt wird, die sie triggert.[6] Welchen Stellenwert man Zeitaussagen in der UML beimisst, kann man daran abschätzen, dass diese Bezeichner im Metamodell der UML keine Erwähnung finden und ihre Anwendung auch nicht demonstriert wird. Bei RUMBAUGH und BOOCH weichen sowohl die Bezeichnungen wie auch die Art ihrer Verwendung voneinander ab.[7]

---

[1] Diese Aussage ist nach HOOGEBOOM auf die gesamte Disziplin Informatik übertragbar. Vgl. [Hoogeboom 1991]. Man beachte die von FEZER dargestellten Probleme der graphischen Darstellung zeitlicher Abhängigkeiten im Kölner Integrationsmodell (KIM). Vgl. [Grochla 1974: S. 107 ff.]. Ohne diese Problematik hier weiter erörtern zu können, ist offensichtlich, dass sich Zeit nur schwierig mit einem statischen Medium wie einer Zeichnung oder Grafik abbilden lässt. Eine weitere Untersuchung dieses Phänomens ist späteren Veröffentlichungen vorbehalten.

[2] Vgl. [Hatley 1993: S. 195], [Balzert 1996: S. 428], [Raasch 1993: S. 218], [Selic 1994], [Ward 1991]. Eine Ausnahme stellt die umfangreiche Monographie von DOUGLAS dar. Vgl. [Douglas 1999].

[3] Vgl. [Booch 1999: S. 322, 324]. Ähnliche Konzepte finden sich auch in der Methode OSA [Embley 1992: S. 188] und bei DE CHAMPEAUX [de Champeaux 1993: S. 171].

[4] Vgl. [Rumbaugh 1999: S. 475], [Booch 1999: S. 467], [OMG 1999: S. 2-75, 2-78].

[5] Vgl. [Douglas 1999: S. 228].

[6] Vgl. [OMG 1999: S. 3-100].

[7] Vgl. [Booch 1999: S. 323], [Rumbaugh 1999: S. 476].

Deshalb wird auf der Grundlage der UML ein eigener Rahmen für die Formulierung zeitlicher Restriktionen entwickelt. Zeitliche Einschränkungen werden im Metamodell für die Metaklassen *ACTIVITY* und *RESPONSIBILITYCOOPERATION* durch das Prädikat `TimingConstraint (Time Expression)` dokumentiert.[1] Durch den Zeitausdruck werden Aussagen über zeitliche Restriktionen einer Interaktion und der beteiligten Aktivitäten formuliert. Dabei wird zwischen einem Zeitpunkt und einer Zeitdauer unterschieden:

- Ein *Zeitpunkt* ist eine ausdehnungslose Zeitangabe. Zwei Zeitpunkte sind entweder identisch oder verschieden, niemals jedoch überlappend.
- Eine *Zeitdauer* bezeichnet die Größe eines Zeitintervalls. Zeitintervalle sind Paare von absoluten Zeitpunkten (Anfangszeitpunkt und Endzeitpunkt), die in einer Reihenfolgebeziehung zueinander stehen. Die Zeitdauer ist die Differenz zwischen Endzeitpunkt und Anfangszeitpunkt.[2]

Zeitdauern werden in der UML durch die Länge des Zeitintervalls bestimmt. Diskrete Zeitstrukturen werden nicht betrachtet. In dem Zeitausdruck werden die Aktion, die Operation und die Interaktion durch die Angabe des jeweiligen Namens referenziert. Dabei werden die folgenden Prädikate verwendet:[3]

- Der Zeitpunkt, durch den die Interaktion beginnt, wird ermittelt durch `<Interaktionsname>.sendTime()`.
- Der Zeitpunkt, durch den der Participant von der Interaktion erfährt, wird ermittelt durch `<Interaktionsname>.receiveTime()`.
- Der Startzeitpunkt einer Aktivität wird ermittelt durch `<Aktivitätenname>.startTime()`. Im Falle der Operation entspricht dies dem Zeitpunkt, zu dem eine bestimmte Operation eines Objekts getriggert wird.
- Der Abschlusszeitpunkt einer Aktivität[4] wird ermittelt durch `<Aktivitätenname>.stopTime()`.

---

[1] Weitere Elemente dieses Rahmens finden sich in den Abschnitten 6.6.11 und 6.8.

[2] In kontinuierlichen Zeitstrukturen besteht ein Zeitintervall aus einem kontinuierlichen Intervall, in diskreten Zeitstrukturen aus einer Menge aufeinander folgender Punkte zwischen absoluten Zeitpunkten.

[3] Vgl. [Booch 1999: S. 323 f.], [Rumbaugh 1999: S. 476]. Ausdrücklich erwähnt BOOCH die Möglichkeit, diese Funktionen sowohl für Nachrichten wie auch für Operationen zu verwenden. Auf diese Weise sollen auch neue Funktionen definiert werden können, um beispielsweise die Komplexität der Ausführungszeit einer Operation angeben zu können.

[4] BOOCH verwendet `startTime()` und `stopTime()` auch für den Start- und Abschlusszeitpunkt von Interaktionen. Vgl. [Booch 1999: S. 323 f.].

– Die Zeitdauer einer Interaktion oder Aktivität wird ermittelt durch
`<Aktivitätenname>.executionTime()`.

Dabei werden die Datentypen t_p für Zeitpunkte und t_d für Zeiträume verwendet. Die genannten Prädikate sind für die Metaklassen *ACTIVITY* und *INTERACTION* definiert und werden durch einen Punktoperator mit dem entsprechenden Modellelement verbunden. Man beachte, dass sie keinen Wert setzen, sondern einen Wert ermitteln. Abgesehen von `executionTime()` besitzen sie als Ergebnis (Rückgabewert) einen Zeitpunkt. Das Prädikat `executionTime()` hat als Ergebnis einen Zeitraum (t_d):

`<Interaktionsname> .sendTime()`	→ t_p
`<Interaktionsname> .receiveTime()`	→ t_p
`<Aktivitätenname> .startTime()`	→ t_p
`<Aktivitätenname> .stopTime()`	→ t_p
`<Interaktionsname> .executionTime()`	→ t_d
`<Aktivitätenname> .executionTime()`	→ t_d

Zwei Zeitpunkte oder Zeiträume lassen sich durch logische Vergleichsoperatoren vergleichen. Das Ergebnis der Vergleichsoperation ist ein logischer Wert:

t_d `<Vergleichsoperator>`	t_d	→ `Bool`
t_p `<Vergleichsoperator>`	t_p	→ `Bool`

Die Addition und Subtraktion von Zeitpunkten und Zeiträumen ergibt einen Zeitraum:

t_p `<Subtraktionsoperator>`	t_p	→ t_d
t_p `<Additionsoperator>`	t_p	→ t_d
t_d `<Subtraktionsoperator>`	t_d	→ t_d
t_d `<Additionsoperator>`	t_d	→ t_d

Darüber hinaus können Zeitpunkte und Zeiträume skalar mit einem Faktor multipliziert werden. Das Ergebnis der Rechenoperation ist ebenfalls ein Zeitpunkt bzw. ein Zeitraum.

t_p `<Multiplikationsoperator>`	`<Faktor>`	→ t_p
t_d `<Multiplikationsoperator>`	`<Faktor>`	→ t_d

Weitere Operationen können bei Bedarf definiert werden.[1] Die Prädikate beziehen sich auf ein bestimmtes Element in der Zeile des Skripts, welches durch den Punkt-Operator referenziert wird.

[1] Eine Möglichkeit der Weiterentwicklung wäre die Verwendung der intervallbasierten temporalen Logik, die 1983 von ALLEN entwickelt wurde. Vgl. [Allen 1983]. Die dort entwickelten Operatoren für Zeitintervalle

So bedeutet `arbeiten.startTime()` den Zeitpunkt eines Arbeitsbeginns. Die folgende Interaktion drückt aus, dass der Versand (präziser: die Übermittlung des Angebots) maximal drei Tage dauert. Für die eindeutige Referenzierung werden die verwendeten Beziehungs- und Intensionsnamen verwendet.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Verkauf	sendet Angebot :Action	Angebotsversand :Interaction TimingConstraint(Angebotsversand.receiveTime() – Angebotsversand.sendTime() ≤ 3 d)	/Kunde :Person	annehmen Angebot :Operation

Skript 72: Zeitverbrauch der Interaktion

Als Konvention wird festgelegt, dass die Beziehungs- und Interaktionsnamen weggelassen werden können, wenn sich das referenzierte Modellelement in der aktuellen Zeile des Skripts befindet. Die zeitliche Restriktion der Interaktion `Angebotsversand` aus Skript 72 entspricht damit folgender Kurzschreibweise: `TimingConstraint(receiveTime()– sendTime()≤ 3 d)`

Weitere Beispiele für zeitliche Restriktionen sind:

- `TimingConstraint(<Operationsname>.startTime()=<Aktionsname>.startTime())` bedeutet, dass Aktion und Operation gleichzeitig beginnen sollen.
- `TimingConstraint(<Operationsname>.startTime()=<Aktionsname>.startTime()+ 5 h)` bedeutet, dass die Operation des Participants 5 Stunden nach dem Beginn der Aktion des Initiators beginnen soll.
- `TimingConstraint((<Operationsname>.startTime()=<Aktionsname>.startTime()) AND (<Operationsname>.endTime()=<Aktionsname>.endTime()))` bedeutet, dass die Aktivitäten gleichzeitig beginnen und gleichzeitig enden sollen.
- `TimingConstraint((<Aktionsname>.executionTime()+<Interaktionsname>.executionTime()< 0,5 h)` bedeutet, dass die Aktivität des Initiators und die Übertragungszeit nicht mehr als 30 Minuten dauern sollen.
- `TimingConstraint((<Interaktionsname>. executionTime() =30 s) AND ((<Operationsname>. receiveTime() - <Aktionsname>. sendTime()) = 45 s))` bedeutet, dass die Übertragungsdauer der Interaktion 45 Sekunden beträgt und die Interaktion 30 Sekunden dauert.[1]

(*before*, *equal*, *meets*, *overlaps*, *during*, *starts*, *finishes*) würden die hier verwendeten Vergleichs- und Rechenoperatoren für Zeitdauern erweitern.

[1] Man denke an eine Werbesendung von 30 s Dauer.

- `TimingConstraint((Brief schreiben.executionTime() = 5 min * Brief.Seitenzahl)` bedeutet, dass das Schreiben des Briefs 5 Minuten pro Seite multipliziert mit der Anzahl der Seiten benötigt.[1]

Die folgende Interaktion bildet den Umstand ab, dass der Versand eines Angebots via Faxgerät genau so lange dauert wie sein Empfang. Darüber hinaus beginnt der Empfang 30 Sekunden, nachdem der Versand begonnen hat.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Sender :Faxgerät	sendet :Action	Übermittlung Fax :Interaction Mode(Supply) Content(:Angebot) TimingConstraint((sendet.startTime() + 30 s = empfängt.startTime()) AND (empfängt. executionTime() = sendet.executionTime()))	/Empfänger :Faxgerät	empfängt :Operation

Skript 73: Zeitverzögerte Aktivitäten in einer Interaktion

Der Aspekt räumlicher Verteilung spielt bei der Architektur heutiger Softwaresystemen eine immer größere Rolle. Durch zeitliche Restriktionen werden auch die für einen Transport über große räumliche Entfernungen benötigten Zeiten abgebildet. Bei der GPM ist zu berücksichtigen, dass Geschäftsprozesse unter Umständen an unterschiedlichen Orten, die räumlich getrennt sind, ausgeführt werden. Die UML bietet für die Modellierung dieses Umstands in den Interaktionsdiagrammen keine entsprechenden Konstrukte an. Die Verteilungsdiagramme sind nur dazu gedacht, die Verteilung der Softwarekomponenten auf unterschiedliche Hardwarekomponenten zu beschreiben.[2]

Räumliche Verteilung modelliert man, indem man das Transportmedium[3], welches für die Übertragung einer Nachricht zuständig ist, explizit als Objekt darstellt. Dies soll durch das folgende Beispiel verdeutlicht werden:

[1] Dabei soll ggf. über die nachlässige Behandlung von Maßeinheiten hinweggesehen werden. An dieser Stelle soll die Aussagefähigkeit des Ausdrucks Vorrang vor mathematischer Präzision erhalten.

[2] Vgl. [Booch 1999].

[3] In der Nachrichtentheorie würde man dies als (Übertragungs-)*Kanal* bezeichnen. Vgl. [Steinbuch 1974: S: 92], [Rechenberg 1999: S. 192, 198].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Sender	versendet Brief :Action	:Interaction TimingConstraint(executionTime() = 2 d) Flow(Material) Form(Brief)	:Empfänger	erhält Brief :Operation

Skript 74: Modellierung eines Transportmediums (1)

Im vorangegangenen Skript wird zwar die Übertragung des Briefs abgebildet, nicht jedoch der Umstand, dass ein Dritter für die Beförderung zuständig ist. Der Verursacher für die Zeitverzögerung wird nicht sichtbar. Wenn er berücksichtigt werden soll, spaltet man – anschaulich gesprochen – die Interaktion auf und fügt das für die Übertragung zuständige Objekt ein.[1] Den Aktivitäten dieses Objekts ordnet man die zum Transport benötigten Zeiten zu:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Sender	versendet Brief :Action	:Interaction Flow(Material) Form(Brief)	:Post	befördert Brief :Operation TimingConstraint(executionTime() =2 Tage)
	:Post	liefert Brief aus :Action	:Interaction Flow(Material) Form(Brief)	:Empfänger	erhält Brief :Operation

Skript 75: Modellierung eines Transportmediums (2)

Ein anderes typisches Anwendungsgebiet für zeitliche Restriktionen sind Antwortzeiten von betrieblichen Anwendungssystemen. Bei einer Call-Center-Lösung beispielsweise sind zu hohe Antwortzeiten zu vermeiden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Anrufer :Person	ruft an :Action	:Interaction	:Mitarbeiter	nimmt Anruf entgegen :Operation
/Anrufer :Person	gibt Kundennummer an :Action	:Interaction	:Mitarbeiter	nimmt Kundennummer entgegen :Operation
:Mitarbeiter	gibt Kundennummer ein :Action	:Interaction	:Call-Center-Lösung	sucht im Kundenbestand :Operation TimingConstraint(executionTime()≤ 5 s)

Skript 76: Modellierung von einzuhaltenden Antwortzeiten

[1] Auf solche Modelloperationen kann hier leider nicht eingegangen werden. Ein ähnliches Prinzip wurde bei der Entwicklung des Konzeptuellen Schemas verwendet.

Das hier verwendete Rahmenwerk kann durch die Definition weiterer Operatoren und Funktionen erweitert werden.[1] Beispielsweise könnte man durch eine Funktion `jitter()` die maximale Schwankung der Ausführungsdauer `executionTime()` einer Interaktion oder Aktivität dokumentieren, wie dies im Bereich der Echtzeitsysteme geschieht.[2]

6.3 Zustandsübergang

Im Gegensatz zu anderen Modellierungsansätzen können mit Skripten auch die Projektionen *System- und Komponentenstrukturen* sowie *Kontrollaspekte und Steuerung* abgebildet werden. Dadurch wird eine simultane Modellierung möglich.

Im objektorientierten Ansatz werden die Veränderungen von Objekten durch Zustandsveränderungen dargestellt. Bei Zustandsübergängen ist der in Initiator und Participant verwendete Objekt- und Klassenname identisch, da die Zustandsübergänge der Objekte einer Klasse abgebildet werden.

Zustandsübergangsdiagramme werden in der Literatur in verschiedensten Variationen eingesetzt. Häufig wird der Name des Zustandsübergangs dazu verwendet, um das auslösende Ereignis des Zustandsübergangs anzugeben. Dass dieses Ereignis eine Aktivität des betreffenden Objekts auslöst, die den Zustandsübergang herbeiführt, bleibt dabei unberücksichtigt. Einige Autoren berücksichtigen dies in der Weise, dass sie den Namen der jeweiligen Operation als Bezeichnung des Zustandsübergangs verwenden,[3] was aber nur eine Konvention darstellt. Davon abweichend wird hier ein anderes Verfahren verwendet, um diesen Zusammenhang zu präzisieren. Wenn der äußere Anstoß, der die Ursache des Zustandsübergangs darstellt, bekannt ist, wird er durch eine Interaktion dargestellt, deren Aktivität den Zustandsübergang des Participants herbeiführt.[1] Im folgenden Beispiel nimmt der Sachbearbeiter einen Antrag entgegen und verändert dadurch seinen Zustand von „unbeschäftigt“ auf „beschäftigt“.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	stellt Antrag :Action	:Interaction Mode(Notification)	:SB	nimmt Antrag entgegen :Operation
:SB	unbeschäftigt :State	Arbeitsbeginn :StateChange	:SB	beschäftigt :State

Skript 77: Zustandsübergang als Folge einer Interaktion

[1] Vgl. [Findler 1971], [Bruce 1972].

[2] Vgl. [Rechenberg 1999: S. 711].

[3] Zum Beispiel [Oestereich 1998: S. 311].

Der Name der Aktivität kann dabei als Prädikat des Zustandsübergangs verwendet werden, um den Zusammenhang zwischen Interaktion und Zustandsübergang zu verdeutlichen. Er wird durch das Prädikat `Act(String)` ausgedrückt, welches für den Beziehungsknotentyp *STATECHANGE* definiert ist. Dies basiert auf der im OA üblichen Sichtweise, Zustandsübergänge mit Aktivitäten zu verbinden, die im Zuge des Zustandsübergangs auszuführen sind.[2] In der Regel stellen diese Aktivitäten Operationen dar, wie das folgende Beispiel zeigt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Konto	nicht ausgeglichen :State	:Statechange Mode(nonspontaneous) Act(Konto ausgleichen :Operation)	:Konto	ausgeglichen :State

Skript 78: Zustandsübergang mit ausführender Aktion

Als weiteres Anwendungsbeispiel wird die Modellierung des Vier-Augen-Prinzips demonstriert. Das Vier-Augen-Prinzip soll sicherstellen, dass im Verlauf der Bearbeitung eines Vorgangs mindestens zwei Personen den Vorgang geprüft haben. Durch zwei Prüfungen verändert sich der Zustand des Vorgangs von „*ungeprüft*“ zu „*Erste Prüfung erfolgt*“ und schließlich zu „*freigegeben*“. Auch hier werden wieder die Möglichkeiten simultaner Modellierung deutlich:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Erstprüfer :SB	prüft Vorgang :Action	:Interaction	:Vorgang	Erstprüfung :Operation
	:Vorgang	ungeprüft :State	Erste Prüfung :StateChange Act (Erstprüfung :Operation)	:Vorgang	Erste Prüfung erfolgt :State
	/Zweitprüfer :SB	prüft Vorgang zur Kontrolle :Action	:Interaction	:Vorgang	Zweitprüfung :Operation
	:Vorgang	Erste Prüfung erfolgt :State	Zweite Prüfung :StateChange Act (Zweitprüfung :Operation)	:Vorgang	freigegeben :State

Skript 79: Modellierung des so genannten Vier-Augen-Prinzips durch Zustände

---

[1] Die auf diese Weise in einen Zusammenhang gesetzten Zustandsübergänge und Interaktionen bilden zusammen die Dynamik eines Systems ab. Vgl. [Thrampoulidis 1995].

[2] Vgl. [Hitz 1999: S. 130, 139], [Embley 1992: S. 64 ff.], [de Champeaux 1993: S. 67]. Das Prädikat `Act()` entspricht bei Zustandsübergängen der *action expression* in der UML, mit der die im Rahmen einer Transition stattfindenden atomaren Berechnungsschritte dokumentiert werden. Vgl. [Rumbaugh 1999: S. 122 ff., 480 ff.].

Eine sinnvolle Erweiterung dieses Konzepts ist es, Zustandsübergänge sowohl mit Operationen wie auch mit Aktionen zu verbinden, da sich durch Interaktionen sowohl der Zustand des Initiators wie auch der des Participants ändern kann:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Maschine	fällt aus :Action	:Interaction	:Arbeiter	prüft Maschine :Operation
:Arbeiter	Arbeitsvorgang beobachtend :State	:Statechange Act(prüft Maschine :Operation)	:Arbeiter	Maschine prüfend :State
:Maschine	In Betrieb :State	:Statechange Act(fällt aus :Action)	:Maschine	Außer Betrieb :State

Skript 80: Interaktion mit Zustandsübergang in Initiator und Participant

Das Skript stellt einen Zusammenhang zwischen der Aktion des Initiators, der Operation des Participants und den jeweiligen Zuständen dar. In der Notation der UML entspricht dies zwei kommunizierenden Zustandsdiagrammen[1] und stellt eine verbesserte Möglichkeit dar, objektübergreifende Dynamik abzubilden.[2]

Das Prädikat `Guard(`*`Constraint Expression`*`)` wird verwendet, um – wie in anderen objektorientierten Ansätzen – Bedingungen auszudrücken, die an Zustandsübergänge geknüpft sind.[3] Diese Bedingung kann Attribute und Selektoren[4] beinhalten. Sie werden spezifiziert, wenn sich der Zielzustand nicht eindeutig aus dem Ausgangszustand und dem Ereignis bestimmen lässt, d. h., von einem Ausgangszustand gehen zwei Zustandsübergänge aus, die durch das gleiche Ereignis ausgelöst werden können. Damit der Zustandsübergang schaltet, müssen zwei Bedingungen erfüllt sein: Der Initiator muss sich im unter Initiator Responsibility angegebenen Zustand befinden. Außerdem muss die unter `Guard()` angegebene Bedingung zutreffen.[5] Im folgenden Skript wird der Kunde nur dann beliefert, wenn sein Attribut `offene Posten` einen Wert kleiner oder gleich 5 hat:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	gesperrt :State	Lieferprüfung :StateChange Guard(self.Anzahl Offene Posten ≤ 5)	/Kunde :Person	wird beliefert :State

[1] Vgl. [OMG 1999: S. 3-135, 3-137]

[2] Verbessert deshalb, weil kommunizierende Zustandsdiagramme in der UML in der aktuellen Version nicht mehr verwendet werden. Vgl. [OMG 2001]

[3] [de Champeaux 1993: S. 67], [Coleman 1992: S. 13].

[4] Vgl. Abschnitt 6.6.4.

[5] Vgl. [de Champeaux 1993: S. 83, 85].

/Kunde :Person	gesperrt :State	Lieferprüfung :StateChange Guard(self.Offene Posten > 5)	/Kunde :Person	gesperrt :State

Skript 81: Beispiel eines Zustandsübergangs mit Bedingung

Durch die Verwendung von `Guard()` wird die Formulierung von Alternativen bei Zustandsübergängen möglich. Auf diese Weise ist es möglich, dass ein Ereignis unterschiedliche Zustandsübergänge bewirken kann. Auf diese Weise wirkt `Guard()` wie ein Wächter, der dafür sorgt, dass nur gültige Zustandsübergänge schalten. Das folgende Beispiel zeigt ein Skript mit bedingten Zustandsübergängen, das resultierende Semantische Netz und das entsprechende UML-Zustandsdiagramm:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Lenkung	Fahrt geradeaus :State	rechts Einschlagen :StateChange Act(rechtsEinschlagen :Operation)	:Lenkung	Rechts eingeschlagen :State
	:Lenkung	Rechts eingeschlagen :State	links Einschlagen :StateChange Act(linksEinschlagen :Operation)	:Lenkung	Fahrt geradeaus :State
	:Lenkung	Rechts eingeschlagen :State	rechts Einschlagen :StateChange Act(rechtsEinschlagen :Operation) Guard(kein Vollausschlag)	:Lenkung	Rechts eingeschlagen :State
	:Lenkung	Fahrt geradeaus :State	links Einschlagen :StateChange Act(linksEinschlagen :Operation)	:Lenkung	Links eingeschlagen :State
	:Lenkung	Links eingeschlagen :State	rechts Einschlagen :StateChange Act(rechtsEinschlagen :Operation)	:Lenkung	Fahrt geradeaus :State
	:Lenkung	Links eingeschlagen :State	links Einschlagen :StateChange Act(linksEinschlagen :Operation) Guard(kein Vollausschlag)	:Lenkung	Links eingeschlagen :State

Skript 82: Zustandsübergänge mit Überwachungsbedingungen

Der in Skript 82 abgebildete Zusammenhang ist wie folgt zu verstehen: Die Zustände der Lenkung eines Kraftfahrzeugs wird durch die Zustände `links eingeschlagen`, `rechts eingeschlagen` und `Fahrt geradeaus` diskretisiert.

- Fährt man geradeaus, kann man nach links und nach rechts lenken.
- Wenn die Lenkung rechts eingeschlagen ist, kann man nach links lenken.
- Wenn die Lenkung rechts eingeschlagen ist, kann man weiter nach rechts lenken, so weit man nicht den Vollausschlag erreicht hat.
- Wenn die Lenkung links eingeschlagen ist, kann man nach rechts lenken.
- Wenn die Lenkung links eingeschlagen ist, kann man weiter nach links lenken, so weit man nicht den Vollausschlag erreicht hat.

Daraus ergibt sich folgendes Semantisches Netz. Man beachte, dass das Prädikat `Guard()` hier nicht abgebildet ist:

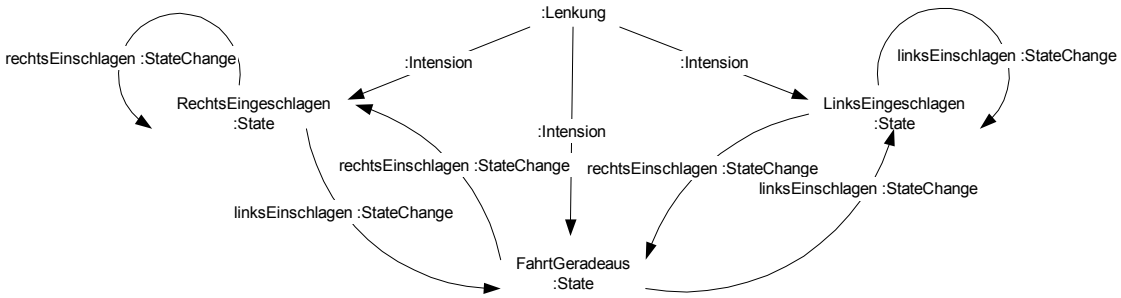


Abbildung 106: Semantisches Netz mit Zustandsübergängen

Aus dem Konzeptuellen Schema lässt sich folgendes UML-Zustandsdiagramm erstellen:

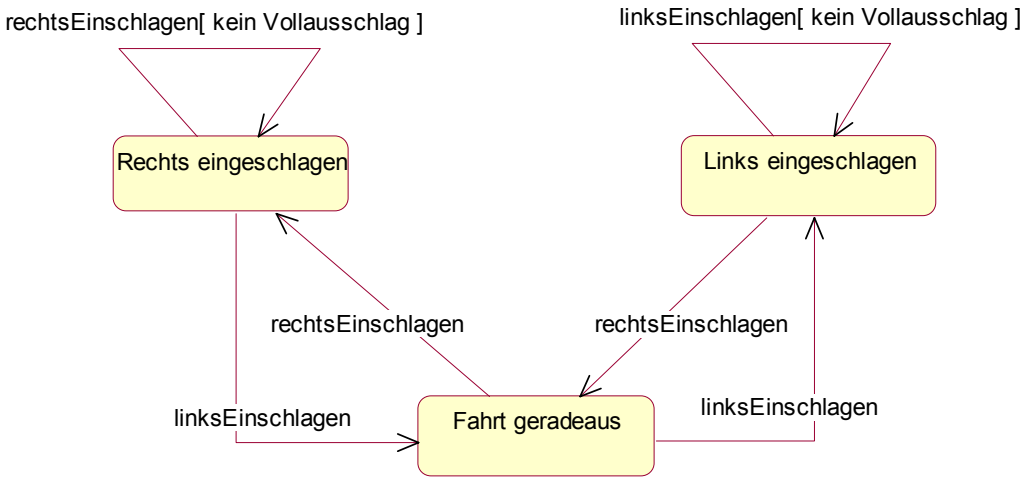


Abbildung 107: UML-Zustandsdiagramm

Da auch Ereignisse Daten beinhalten können,[1] dient das Prädikat `Content(String)` dazu, diese anzugeben, wie folgendes Beispiel zeigt:

[1] Vgl. [Shlaer 1992: S. 43], [de Champeaux 1993: S. 83].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Computer	wartet auf Eingabe :State	Eingabe :StateChange Content(”A”:String)	:Computer	verarbeitet Eingabe :State

Skript 83: Zustandsübergang mit Argument

Das Prädikat `Mode(Statechange Mode)` wird bei Zustandsübergängen verwendet, um folgende besonderen Zustandsübergänge abbilden zu können:[1]

- `Automatic`: Der Zustandsübergang wird ausgelöst, ohne dass eine Interaktion stattgefunden hat.
- `Nonspontaneous`: Der Zustandsübergang benötigt den Eingang einer Nachricht. Dies ist der Standardwert, der nicht explizit angegeben werden muss.
- `Initial`: Der Zustandsübergang hat keinen Vorgängerzustand und ist immer möglich. Für die Initiator Responsibility wird der Zustand `Initial` gesetzt.[2]
- `Final`: Der Zustandsübergang besitzt keinen Folgezustand. Für die Participant Responsibility wird der Zustand `Final` gesetzt.[3]
- `Internal`: Der Start- und Endzustand sind identisch.[4]

Für jede Klasse sind drei Zustände vordefiniert:[5]

- Erzeugt (`Initial`): Der Zustand, den ein Objekt in dem Moment besitzt, in dem es erzeugt wurde.
- Beendet (`Final`): Der Endzustand eines Objekts.
- Immer (`Any`): Dieser Zustand wird verwendet, um auszudrücken, dass der von diesem ausgehende Zustandsübergang immer schalten kann.

Bestimmte Zustandsübergänge können auch durch das Eintreten von Zeitpunkten oder durch das Ablaufen von Zeitfristen schalten. Diese Umstände werden in der UML nicht durch eine Überwachungsbedingung abgebildet, sondern durch die Prädikate `when()` und `after()`:[6]

---

[1] Vgl. [Firesmith 1995: S. 446 f.], [Embley 1992: S. 76 ff.].

[2] Vgl. [Rumbaugh 1999: S. 397, 444], [Booch 1999: S. 289].

[3] a. a. O.

[4] Vgl. [Rumbaugh 1999: S. 436].

[5] Vgl. [Firesmith 1995: S. 446 f.]. Die Differenzierung der UML zwischen Pseudo- und Spezialzuständen wird nicht übernommen. Vgl. [Rumbaugh 1999: S. 397].

[6] Vgl. [Rumbaugh 1999: S. 475].

- when(*Expression*): Eine Bedingung *Expression* trifft zu. Dies kann auch das Eintreten eines absoluten Zeitpunktes, wie when(Date =01.01.2003), sein.
- after(*Period, Point Of Reference*): Ein bestimmter Zeitraum ist seit dem Bezugspunkt vergangen. Die Angabe des Bezugspunktes ist optional.

Die Prädikate when() und after() unterscheiden sich in folgender Weise von Guard(): Guard() wird verwendet, um auf ein Ereignis mit unterschiedlichen Zustandsübergängen reagieren zu können. Notwendig ist also in jedem Fall das Eintreten eines Ereignisses. when() und after() benötigen dagegen kein zusätzliches Ereignis. Der Zustandsübergang tritt automatisch ein, wenn die angegebene Bedingung zutrifft oder der angegebene Zeitraum vergangen ist:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Objekt	Aktiv :State	freigeben :StateChange after(2 s)	:Objekt	Frei :State

Skript 84:Schalten eines Zustandsübergangs nach Ablauf einer Frist

Auch für Zustandsübergänge ist das Prädikat TimingConstraint(*Time Expression*) definiert. Auf diese Weise können zeitliche Einschränkungen ausgedrückt werden, die während eines Zustandsübergangs beachtet werden müssen.[1] Die folgende Einschränkung erweitert das Skript 78 in der Form, dass ein Konto innerhalb von drei Tagen ausgeglichen werden muss:

```
ausgleichen :Statechange
TimingConstraint(ausgeglichen – nicht ausgeglichen ≤ 3 d)
```

Diese Modellierung ist in der UML nicht vorgesehen[2] und ergibt sich aus der Tatsache, dass hier Zustandsübergänge Zeit verbrauchen können.[3] Durch das Prädikat Kind() wird – wie bei Interaktionen – ausgedrückt, ob der Zustandsübergang als normal angesehen wird oder als Reaktion auf einen eingetretenen Fehler stattfindet.[1]

Die Modellierung von Zielen und Problemen als Objekte (vgl. Abschnitt 6.1.2) ermöglicht die Modellierung von Zuständen und Zustandsübergängen auch für solche Objekte. Dies sei an dem Beispiel demonstriert, dass ein Zielobjekt durch eine Aktivität den Zustand erreicht erhält:

---

[1] Dies wurde durch [Embley 1992: S. 84 ff.] und [de Champeaux 1993: S. 170 ff.] inspiriert. Es wird in der UML als *Transition Time* bezeichnet. Sie wird dort jedoch nicht in Zustandsdiagrammen, sondern in Sequenz- und Kollaborationsdiagrammen verwendet. Vgl. [OMG 1999: S. 3-100].

[2] Die UML sieht nur Startzeitpunkte für Zustandsübergänge vor. Vgl. [OMG 1999: S. 3-130].

[3] Vgl. Abschnitt 4.3.5.2.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Kundenbefriedigung Stpe(Goal)	nicht erreicht: :State	:Statechange Act(Ware liefern :Operation)	:Kundenbefriedigung Stpe(Goal)	erreicht :State

Skript 85:Veränderung des Zustands eines Zielobjekts

6.4 Zugriff auf Eigenschaften und Zustände

Würde das Setzen des Zustands eines Objekts oder die Veränderung eines Attributs über eine Interaktionsbeziehung geschehen, hätte dies zur Folge, dass ein Objekt direkt den abstrakten oder konkreten Zustand eines anderen Objekts verändern könnte. Das folgende Skript demonstriert eine solche – hier nicht zugelassene – Veränderung des abstrakten Objektzustands über eine Interaktionsbeziehung:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	trinkt Flasche leer :Action	Ungültige Interaktion :Interaction	:Flasche	leer :State

Skript 86: Unzulässige Veränderung des Objektzustands durch Interaktion

Obwohl in vielen Modellierungsansätzen prinzipiell möglich, sollte eine solche direkte Manipulation des Objektzustands unzulässig sein, da sie dem Geheimnisprinzip widerspricht. Aus diesem Grund wird sie hier ausgeschlossen. Es gilt: Die Kombination von `Interaction` in der Tabellenspalte Connection und `State` oder `Attribute` in der Tabellenspalte Participant Responsibility ist unzulässig.

Die Zustandsveränderung eines Objekts der Klasse Flasche muss über eine Operation der Flasche erfolgen, die wiederum den (internen) Zustand durch eine Aktion verändert. Hierbei handelt es sich weder um eine Interaktion zwischen zwei Objekten noch um die Selbst-Interaktion[2] eines Objekts. Deshalb wird dieser Zugriff auf die Interna eines Objekts durch den Beziehungsknotentyp *INTERNAL* abgebildet: [3]

[1] Vgl. Abschnitt 6.2.13.

[2] Vgl. [Embley 1992: S. 189], [de Champeaux 1993: S. 89].

[3] Zustandsveränderungen dokumentieren Veränderungen des abstrakten Zustands, interne Beziehungen dokumentieren den Teil des konkreten Zustands, der durch Attribute gebildet wird.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	trinkt Flasche leer :Action	:Interaction	:Flasche	leeren :Operation
:Flasche	wird geleert :Action	:Internal	:Flasche	leer :State

Skript 87: Erlaubte Veränderung des Objektzustands über eine interne Beziehung

Soll ausgedrückt werden, dass sich durch die Aktivität `füllen` der Zustand einer Flasche auf `gefüllt` ändert, wird dies in folgender Weise formuliert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Flasche	füllen :Action	:Internal	:Flasche	gefüllt :State

Skript 88: Explizites Setzen eines Zustands

Dabei kann die Operation `leeren` durch Vor- und Nachbedingungen die stattfindende Zustandsveränderung abbilden.[1]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	trinkt Flasche leer :Action	:Interaction	:Flasche	leeren :Operation Pre(not (State = leer)) Post(State = leer)

Skript 89: Durch Vor- und Nachbedingungen ausgedrückte Zustandsveränderung

Für interne Zugriffe gilt zusätzlich die Restriktion, dass der Objekt- und Klassenname von Initiator und Participant identisch ist. Das Prädikat Inhalt `Content(`*`String`*`)` wird hier verwendet, um zu dokumentieren, in welcher Weise das Attribut verändert wird. Hier können Objektbezeichner, Attribute, Variablen, Berechnungsausdrücke oder Werte verwendet werden.

Das Prädikat `Mode(`*`Access Mode`*`)` wird bei internen Beziehungen verwendet, um beispielsweise lesenden (`read`) und schreibenden (`change`) Zugriff unterscheiden zu können. Das folgende Beispiel zeigt, wie sich durch die Aktivität `heiraten` das Attribut `Familienstand` einer Person auf den Wert „`verheiratet`" ändert. Der Zugriff auf das Attribut geschieht als Folge der vorher stattfindenden Interaktion:[2]

[1] Darauf wird noch in Abschnitt 6.6.5 eingegangen.

[2] Man beachte, dass die Tabellenspalte Participant Responsibility zwar den Namen des Attributs, nicht jedoch den aktuellen Wert des Attributs beinhaltet. Hier kommt das gleiche Prinzip zur Anwendung, welches

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Standesbeamter	traut :Action	:Interaction	:Person	heiraten :Operation
	:Person	heiraten :Action	:Internal Mode(change) Content(”verheiratet”)	:Person	Familienstand :Attribute

Skript 90: Beispiel einer internen Beziehung

Das Prädikat `Content( „verheiratet“)` gibt den Inhalt des Schreibvorgangs auf Exemplarebene an. Die Verwendung der Punktnotation ist hier nicht notwendig, da durch den Inhalt der Tabellenspalten Participant und Participant Responsibility Klasse und Attribut festgelegt sind. Unter Anwendung der Punktnotation ergibt sich: `Content (Person .Familienstand = „verheiratet“)`. Weitere Beispiele für zulässige Inhaltsprädikate sind:

- `Content(2654*4,3+101)`
- `Content(Person.Kundennummer)`
- `Content(Kassenbestand – Betrag)`

6.5 Referentielle und taxonomische Beziehungen

Die im Skriptmodell verwendeten taxonomischen und referentiellen Beziehungen unterscheiden sich von den bisher verwendeten Beziehungstypen dadurch, dass die Tabellenspalten Initiator Responsibility und Participant Responsibility unbesetzt bleiben. Außerdem können in den Tabellenspalten Initiator und Participant Klassennamen ohne vorangestelltes „:“ verwendet werden, wenn Beziehungen zwischen Klassen dokumentiert werden.[1]

6.5.1 Assoziationsbeziehung

Bei den referentiellen Beziehungen wird in der UML zwischen Beziehungen auf Typebene – die als *Assoziation* bezeichnet werden[2] – und Beziehungen auf Exemplarebene – die als *Link* bezeichnet werden[3] – unterschieden. Erstere verbinden Klassen, letztere verbinden Objekte.

---

schon bei Interaktionsbeziehungen und Operationen verwendet wird. Aktuelle Werte und Argumente werden in der Tabellenspalte Connection abgebildet.

[1] In diesem Fall muss der Rollenname in der Form `Klassenname /Rollenname` geschrieben werden.

[2] Vgl. [OMG 1999: S. 2-15, 2-19], Rumbaugh 1999: S. 152].

[3] Vgl. [Rumbaugh 1999: S. 153, 325 ff.]. Diese Unterscheidung ist rein terminologisch. In der graphischen Darstellung unterscheidet sich der Link von der Assoziation nur durch die fehlenden Kardinalitätsangaben, die beim Link keinen Sinn ergeben. Nur durch den Umstand, dass die verwendeten Modellelemente

Das folgende Skript stellt ein Beispiel einer Assoziationsbeziehung zwischen den Klassen `Mitarbeiter` und `Pkw` dar. Der in der Tabellenspalte Initiator verwendete Begriff wird als Urbild der Assoziation angesehen, d. h., die Assoziation wird von links nach rechts gelesen.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Person		Eigentümer von :Association	Pkw	

Skript 91: Beispiel einer Assoziation auf Typebene

Eine referentielle Beziehung zwischen Objekten ist daran zu erkennen, dass im Skript in den Tabellenspalten Initiator und Participant eine Zeichenkette der Form `Objektbezeichner :Klassenbezeichner` verwendet wird.[1] Im Gegensatz zur UML werden keine besonderen Bezeichnungen für referentielle Beziehungen auf Objektebene verwendet:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Fred Bast :Person		Eigentümer von :Association	W-LI 1966 :Pkw	

Skript 92: Beispiel einer Assoziation auf Objektebene

Der Grund für diese abweichende Verwendung liegt in einer Schwäche des Sprachentwurfs der UML. Die Instanz einer Assoziation heißt dort *Link*. Die Instanz einer Aggregationsbeziehung besitzt dagegen keinen eigenen Namen, sondern wird ebenfalls als *Link* bezeichnet.[2] Die hier verwendete Konzeption, Beziehungen auf Typ- und Exemplarebene immer gleich zu benennen und dadurch zu unterscheiden, dass im einen Fall Klassen, im anderen Fall Objekte beteiligt sind, erscheint günstiger.

In der objektorientierten Modellierung werden Kardinalitäten verwendet, mit denen angegeben wird, wie viele Elemente der Urbildmenge mit wie vielen Elementen der Bildmenge verbunden sein können.[3] Da diese Kardinalitäten von Beziehung zu Beziehung abweichen können, sind sie ein Prädikat der Beziehung und entsprechen nicht der mit `Card()` festgehaltenen Kardinalität der

---

Objekte sind, ist in der UML eine eindeutige Unterscheidung möglich. Insofern ist hier nur ein – in der praktischen Anwendung redundanter – Begriff entfernt worden.

[1] Dies entspricht der in der UML verwendeten Notation für Objekte. Vgl. [Rumbaugh 1999: S. 361].

[2] Vgl. [Rumbaugh 1999: S. 326]. Die Ursache hierfür besteht darin, dass die Aggregation in der UML eine Assoziation mit zusätzlicher Semantik darstellt.

[3] Vgl. [Vetter 1995: S. 97 ff.].

Extension einer Klasse.[1] Man benötigt zwei Prädikate, um bei referentiellen Beziehungen auf Klassenebene die Anzahl der Objekte bestimmen zu können, die auf der Initiator- und Participant-Seite an der Beziehung teilnehmen können. Diese Kardinalitäten werden durch die Prädikate `DomCard(`*`Cardinality Expression`*`)` für den Initiator und `RngCard(`*`Cardinality Expression`*`)` für den Participant ausgedrückt.[2] Das folgende Beispiel drückt die Beziehung aus, dass Mitarbeiter keinen oder einen Taschenrechner besitzen. Außerdem sind Taschenrechner immer Eigentum eines Mitarbeiters:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Person /Eigentümer		Eigentümer von :Association DomCard(1) RngCard(0..1)	Taschenrechner /Eigentum	

Skript 93: Beispiel einer statischen Beziehung mit Kardinalitäten

Wie man sieht, wird in diesem Fall der Rollenname hinter dem Klassennamen aufgeführt. Sonst wäre der Klassenname nicht zu identifizieren, da Leerzeichen keine Trennsymbole darstellen. Daraus entsteht folgendes Semantische Netz:

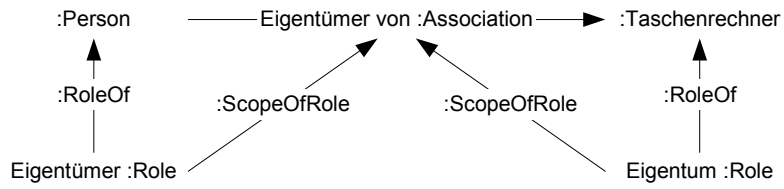


Abbildung 108: Semantisches Netz für eine statische Beziehung

Wie im Konzeptuellen Schema erwähnt, sind referentielle Beziehungen unidirektional bzw. gerichtet. Abweichend von der UML werden keine bidirektionalen bzw. ungerichteten referentiellen Beziehungen verwendet, sondern durch zwei gerichtete referentielle Beziehungen

[1] Vgl. Abschnitt 6.1.4.

[2] Die Angabe der Kardinalitäten entspricht der von CHEN entwickelten Art und Weise, die auch in der UML verwendet wird. Vgl. [Castellani 2000]. In der Methode Object Oriented Analysis (OOA) von COAD und YOURDDON [Coad 1991] erfolgt die Angabe genau umgekehrt. Die Präfixe `Dom` und `Rng` sind mnemonische Abkürzungen für *Domain* (Urbild bzw. Definitionsbereich) und *Range* (Bild bzw. Wertebereich).

ersetzt. Dies entspricht der Konzeption in der OML[1], in der bidirektionale Beziehungen als verkürzte Schreibweise für zwei unidirektionale Beziehungen angesehen werden. Dadurch wird die eindeutige Interpretation der Beziehung sichergestellt, die der Leserichtung von Initiator zum Participant entspricht.

6.5.2 Mereologische Beziehungen

Bei mereologischen Beziehungen findet sich in der Initiator-Tabellenspalte der die Ganzheit ausdrückende Begriff. Die Unterscheidung von Beziehungen zwischen Klassen und Objekten wird analog zum in Abschnitt 6.5.1 verwendeten Verfahren abgebildet. Wenn Mitarbeiter als Mitglieder einer Abteilung angesehen werden, wird dies als Skript wie folgt ausgedrückt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Abteilung		:Membership	Mitarbeiter	

Skript 94: Mereologische Beziehung auf Klassenebene

Wenn dagegen ausgedrückt werden soll, dass eine bestimmte `Person` Mitglied einer `Abteilung` ist, wird ein Objektbezeichner verwendet:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Nr. 6 :Abteilung		:Membership	Marcel Mass :Mitarbeiter	

Skript 95: Mereologische Beziehung auf Objektebene

Auch eine *shared aggregation*, also eine Aggregation, bei der ein Teil Bestandteil von mehreren Ganzen ist, ist durch ein Skript abzubilden. Skript 96 stellt eine *shared aggregation* auf Exemplarebene dar, Skript 97 auf Klassenebene:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Nr. 6 :Abteilung		:Membership	Marcel Mass :Mitarbeiter	
	Nr. 3 :Abteilung		:Membership	Marcel Mass :Mitarbeiter	

Skript 96: Eine sogenannte shared aggregation *auf Objektebene*

[1] Vgl. [Firesmith 1997].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Abteilung		:Membership DomCard(1..n) RngCard(n)	Mitarbeiter	

Skript 97: Eine shared aggregation *auf Klassenebene*

Die im Konzeptuellen Schema erwähnten unterschiedlichen Bedeutungen bzw. Arten, mit denen die Aggregationsbeziehung und die Mitgliedschaft verwendet wird, werden durch das Prädikat `SubType(`*`Relationship Type`*`)` abgebildet. Auf diese Weise lässt sich die Art der mereologischen Struktur genauer bestimmen. Der `Relationship Type` ist als Prädikat und nicht als Beziehungstyp angelegt, da die praktische Erfahrung zeigt, dass eine genaue Bestimmung schwierig und bis zu einer endgültigen Entscheidung häufig verändert wird. Für *`Relationship Type`* werden die Terme

- *Komposition* (`Component-Integral Object Composition`),
- *Material-Objekt-Komposition* (`Material-Object Composition`),
- *Feature-Aktivität* (`Feature-Activity`),
- *Ort-Bereich* (`Place-Area`) für die Aggregationsbeziehung

und

- *Portion-Objekt-Komposition* (`Portion-Object Composition`),
- *Kollektion* (`Member-Bunch Composition`),
- *Partnerschaft* (`Member-Partnership`) für die Mitgliedschaft verwendet.[1]

6.5.3 Containerbeziehung

Bei der Containerbeziehung beinhaltet die Initiator-Tabellenspalte den Behälter. Wenn eine Person in der Rolle eines Fahrgastes als Inhalt eines Busses angesehen wird, wird dies als Skript wie folgt ausgedrückt:

[1] Auf Beispiele wurde verzichtet. Siehe dazu den entsprechenden Abschnitt im Konzeptuellen Schema.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Bus		:Containment DomCard(1) RngCard(0..96)	Person /Fahrgast	

Skript 98 : Beispiel einer Containerbeziehung

Die Unterscheidung von Beziehungen zwischen Klassen und Objekten wird analog zum in Abschnitt 6.5.1 verwendeten Verfahren abgebildet.

6.5.4 Prädikate für referentielle Beziehungen

In der UML ist für die Assoziationsbeziehung keine präzise Semantik definiert.[1] Das Gleiche gilt auch für die Aggregationsbeziehung, die dort eine Spezialisierung der Assoziation darstellt. Die OML bietet an dieser Stelle bessere Primitive, um die Semantik der referentiellen Beziehungen durch Stereotype zu präzisieren.[2] Daraus wurden hier die folgenden Prädikate entwickelt:

- Das Stereotyp *Lebensdauer* drückt in der OML aus, ob die Existenz des Participant von der Existenz des Initiators abhängig ist. Dabei wird nach FIRESMITH nur die Situation des Löschens berücksichtigt. Das Prädikat `Lifespan(`*`Lifespan Identifier`*`)` drückt aus, ob der Participant gelöscht wird, wenn der Initiator gelöscht wird. Wird als Term `dependent` verwendet, bedeutet dies, dass die Lebensdauer des Participant von der Lebensdauer des Initiators abhängig ist. Die anderen Terme sind `independent` und `unknown`..
- Das Stereotyp *Variabilität* drückt in der OML aus, ob ein Participant die Beziehung zum Initiator wechseln kann. Eine variable Beziehung liegt beispielsweise vor, wenn der Taschenrechner aus Skript 93 einen anderen Eigentümer erhalten kann. Dies wird durch das Prädikat `Var(`*`Variability Identifier`*`)` abgebildet. Durch den Term `variable` wird ausgedrückt, dass der Participant die Zugehörigkeit zu einem Initiator wechseln kann. Durch den Term `constant` wird ausgedrückt, dass der Participant die Zugehörigkeit zum Initiator nicht wechseln kann.[3]

---

[1] Vgl. [OMG 1999: S. 2 – 19].

[2] Vgl. [Firesmith 1998b], [Henderson-Sellers 1998: Appendix E, Stichwort „*Aggregation, membership and containment*"].

[3] Das in der OML verwendete Stereotyp *Optionality* wird hier durch das Prädikat `RngCard()` ausgedrückt. Da der Nachrichtenaustausch hier durch den Beziehungstyp `Interaction` abgebildet wird, war die Übernahme des OML-Stereotyps *Usage* nicht notwendig. Die Übernahme der übrigen Stereotype der OML (*Implementation*, *Concurrency*, *Persistence*, *Source*) war nicht sinnvoll, da es sich nach FIRESMITH um

- Darüber hinaus dokumentiert das Prädikat `Mode(Change Mode)`, wenn eine referentielle Beziehung angelegt (`create`) oder beendet (`destroy`) wird.[1] Auf diese Weise werden Veränderungen von referentiellen Beziehungen im Zeitverlauf gekennzeichnet.[2]

6.5.5 Spezialisierungsbeziehungen

Bei Spezialisierungsbeziehungen (Beziehungstyp *SPECIALISATION* im Konzeptuellen Schema) findet sich in der Participant-Tabellenspalte der generelle Begriff. Üblicherweise erhalten Spezialisierungsbeziehungen keinen Namen. Das folgende Skript beinhaltet als Beispiel eine Spezialisierungsbeziehung, durch die ausgedrückt wird, dass es sich bei einem Taschenrechner um eine Spezialisierung eines Betriebsmittels handelt. Vererbungsbeziehungen werden in gleicher Weise dargestellt. Ihr Beziehungstyp ist dann `Inheritance`. Der Diskriminator der Spezialisierungsbeziehung wird durch das Prädikat `Discrim(String)` angegeben. Weitere Prädikate können bei Bedarf definiert werden.[3]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Taschenrechner		:Specialization Discrim(Organisatorische Einordnung)	Betriebsmittel	

Skript 99: Beispiel einer Spezialisierungsbeziehung in Skriptnotation

Spezialisierungsbeziehungen zwischen Objekten werden zwar sehr selten benötigt, sind aber prinzipiell möglich und daran zu erkennen, dass auch hier in den Tabellenspalten Initiator und Participant eine Zeichenkette der Form `Objektbezeichner :Klassenbezeichner` verwendet wird. Wenn mehrfache Spezialisierung bzw. Vererbung abgebildet werden soll, geschieht dies durch zwei entsprechende Beziehungen. Die „Ursache“ der Spezialisierung wird als Diskriminator angesehen.

Differenzierungen handelt, die für die Implementierung eines Objektmodells in einer objektorientierten Programmiersprache von Bedeutung sind. Dies ist hier jedoch nicht das Thema.

[1] Siehe hierzu Abschnitt 6.6.4. Das Prädikat wird verwendet, wenn durch eine Aktivität zwischen Objekten eine referentielle Beziehung geknüpft wird.

[2] Dies wird von RAMACKERS und VERRIJN-STUART als Dynamik zweiter Ordnung bezeichnet, wogegen die Dynamik erster Ordnung die Systemstruktur unverändert lässt. Vgl. [Kruse 1996: S. 20].

[3] Dies könnte z. B. ein Prädikat sein, durch welches abgebildet wird, ob die Vererbungsbeziehung disjunkt oder überlappend ist. Vgl. [OMG 2001: S. 2-46].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Hausboot		:Specialization Discrim(Wohnort)	Haus	
	Hausboot		:Specialization Discrim(Schwimmfähigkeit)	Boot	

Skript 100: Mehrfache Spezialisierung mit Differenzierung durch einen Diskriminator

Skript 100 ist ein Lehrbuchbeispiel gegen Mehrfachvererbung[1] und zeigt den Vorteil der Definition einer allgemeinen Spezialisierungsbeziehung. Umgangssprachlich ist ein Hausboot eine Art Boot und eine Art Haus. Die Klasse `Hausboot` erfüllt aber nicht die Kriterien für die Mehrfachvererbung. Eine Klasse `Haus` wird über ein Attribut `Keller` verfügen, was bei der Klasse Hausboot unsinnig ist. Während die Verwendung des Beziehungstyps `Inheritance` aufgrund der damit assoziierten Semantik unpassend ist, ist die Verwendung des schwächeren Beziehungstyps `Specialization` durchaus akzeptabel.

6.6 Operationen

Die Participant Responsibility beinhaltet für Zeilen, die eine Interaktion darstellen, die Spezifikation einer Operation, die vom in der Tabellenspalte Participant genannten Objekt ausgeführt wird. Durch die Zuordnung einer Operation zu einem Objekt bzw. zu einer Klasse wird im Rahmen der objektorientierten Modellierung die Aufgabenverteilung abgebildet. Bei der GPM übernehmen die Objekte bzw. Klassen die Rolle von *Aktionsträgern*. Dabei kann es sich nach SCHULTE-ZURHAUSEN sowohl um einzelne Personen wie auch um Sachmittel handeln.[2]

Hier ist zu beachten, dass es sich beim Inhalt der Tabellenspalte um die Spezifikation der Schnittstelle der Operation handelt, nicht etwa um die Prozeduren, welche ein Objekt ggf. zur Erfüllung der Operation ausführt.[3] Diese Prozeduren sind Interna des Objekts und werden hier

[1] Der Autor hat es nicht selbst konstruiert. Allerdings ist ihm der Fundort nicht mehr bekannt.

[2] Vgl. [Schulte-Zurhausen 1999: S. 50]. An dieser Stelle wäre eine weitergehende Differenzierung in Aufgaben- und Arbeitsträger möglich, wie dies in der Organisationslehre der Fall ist. Vgl. [Wild 1966], [Kieser 1992], [Hoffmann 1992b], [Kosiol 1962], [Nordsieck 1955], [Nordsieck 1972], [Schulte-Zurhausen 1999], [Krüger 1994]. Auf diese Weise könnten auf der einen Seite Aufgabenträger weiter in Stellen, Gruppen und Abteilungen usw. differenziert werden, auf der anderen Seite könnte zwischen Arbeitsträgern und Arbeitsobjekten unterschieden werden. Dies erfolgt hier durch die in Abschnitt 6.1.2 genannten Stereotype.

[3] Die Realisierung einer Operation geschieht in der UML durch die Metamodell-Klasse `Method` (vgl. [OMG 1999: S. 2-35]). Diese besitzt ein Attribut `body`, welches Pseudocode beinhalten kann. Ausführbare Programmanweisungen sind nicht Inhalt der UML.

nicht betrachtet. Sie sind durch das objektorientierte Geheimnisprinzip verborgen. Nur die zur Realisierung der Operation notwendigen Aktionen werden in den folgenden Zeilen eines Skripts in der Tabellenspalte Initiator Responsibility als Aktionen dokumentiert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	:Interaction	:Objekt X	Leistung A :Operation
	:Objekt X	resultierende Aktion 1 :Action	...	...	...
	:Objekt X	resultierende Aktion 2 :Action	...	...	...
	...	...	...	...	...

Skript 101: Operation und Aktion

Die Angabe der aus der Operation erwachsenden Aktionen ist nicht zwingend erforderlich. Die Spezifikation der Schnittstelle ist ausreichend, um den Verlauf eines Prozesses zu dokumentieren,

- wenn die Details der Realisierung nicht relevant sind,
- wenn die Realisierung vollständig dem Objekt überlassen ist,
- wenn im Rahmen der Realisierung keine Interaktionen mit anderen Objekten stattfinden,
- wenn keine Zustandsveränderungen zu beobachten sind,
- wenn die Details verborgen sind oder verborgen bleiben sollen.

Das Geheimnisprinzip geht davon aus, die Interna der Objekte und somit auch die Interna der Operationen vor ihrer Umgebung zu verbergen. Wenn eine Klasse `Schreibkraft` eine Operation `Bericht erstellen` ausführt, sind die Details, wie diese Operation ausgeführt wird, in der Regel nicht von Interesse. Wenn diese Details untersucht werden sollen, werden die im Rahmen der Erfüllung der Operation ausgeübten Aktionen dokumentiert. Interagiert im Rahmen einer Operation ein Objekt mit anderen Objekten, können diese als separate Interaktionen abgebildet werden. Finden keine Interaktionen statt, können die einzelnen Schritte einer Operation als Zustandsübergänge abgebildet werden.[1]

Die UML verwendet für die Abbildung der Interna einer Operation Kollaborationsdiagramme, Zustandsdiagramme und Aktivitätendiagramme, welche eine besondere Form von sequentialisierten Zustandsdiagrammen derart darstellen, dass der Ablauf als Abfolge von Transitionen

[1] Vgl. [Rumbaugh 1999: S. 135].

angesehen wird. Sie sind aber auch zur Darstellung von Algorithmen verwendbar.[1] Wenn im Rahmen einer Operation Interaktionen mit anderen Objekten stattfinden, wird dies in der UML durch Kollaborationsdiagramme und hier durch Skripte abgebildet.[2]

Wenn die Interna der Operation nicht bekannt sind, kann ihre Semantik nur aus der Schnittstelle erschlossen werden. Dies gilt für alle objektorientierten Modellierungsansätze.[3] Es gilt also, durch Prädikate eine aussagefähige Spezifikation von Operationen zu ermöglichen.

Der Name ist der Name der Operation, der in der Tabellenspalte `Participant Responsibility` angegeben wird. Er wird in der Form `Operationsname :Operation` festgehalten. Für Operationen ist die Angabe eines Namens zwingend. Dieser Name beinhaltet üblicherweise ein Verb (schreiben, erstellen, löschen, erzeugen, versenden, ...), das den verrichtungsorientierten Charakter der Operation wiedergibt. Dies ist in starkem Maße vom Modus der Interaktion abhängig, in deren Rahmen die Operation stattfindet. Der Modellierer hat die Freiheit, ob das Verb im Infinitiv, im Imperativ oder in der dritten Person Singular formuliert wird.[4]

Bei Operationen, die im Rahmen von Informationsanforderungen (Interaktionen mit dem Modus *Request*) ausgeführt werden, besteht der Name meistens aus einem Hilfsverb (`gib` .., `ermittle` ..., ...) und der zurück zu liefernden Information (`Adresse`, `Auftragsstatus`, `Kundenname`, ...). Auch Konstruktionen mit substantivierten Verben sind angebracht (`Ermittlung` ..., `Rückgabe` ...). Der Modus einer Interaktion kann aber durchaus der einer Informationsanforderung sein und bei der Operation des Participants eine Berechnung bewirken (Beispiel: `berechne Tagesumsatz`). Der Grund besteht darin, dass der Modus der Interaktion den Zweck der Interaktion dokumentiert und nicht, welche Operation zur Erfüllung des Zwecks vom Participant ausgeführt wird. Dies ist auch auf die anderen Modi der Interaktionen übertragbar. Auf eine Informationsübermittlung (Interaktion mit Modus *Supply*) kann beispielsweise der Participant eine Berechnung durchführen, die die übermittelten Daten als Eingabe verwendet.

---

[1] Die Dokumentation der Interna von Operationen mit Aktivitätendiagrammen stellt eine sinnvolle Erweiterungsmöglichkeit des hier vorgestellten Modellierungsansatzes dar. Da das Thema der Arbeit die objektorientierte Modellierung mit Hilfe von Skripten ist, wird auf diese Möglichkeit nicht weiter eingegangen.

[2] Vgl. Abschnitt 6.11.4.

[3] Zur Spezifikation von Operationen in der UML vgl. [Rumbaugh 1999: S. 369], [Hitz 1999: S. 23], [Booch 1999: S. 129]. Andere Modellierungsansätze verwenden die gleichen oder zumindest ähnliche Konzepte. Vgl. [Frank 1998b: S. 28], [Balzert 1996], [Kappel 1996: S. 43], [Waldén 1995: S. 40], [Booch 1995: S. 249 f.].

[4] Die Anwendung der OBA zeigte, dass der Versuch, dies zu normieren, eher kontraproduktiv ist.

6.6.1 Ein- und Ausgaben

Ein- und Ausgaben von Operationen werden in der UML über Argumente bzw. formale Parameter (*Parameter List*) und Rückgabewerte (*Return Type*) abgebildet.[1] Die Ein- und Ausgabe wird hier durch die beiden Prädikate `In(`*`Identifier List`*`)` und `Out(`*`Identifier List`*`)` dokumentiert, die eine Liste von Argumentnamen und Objekttypen in der Form `Argumentname :Objekttyp` (Beispiel: `Vorgang :Mahnung`) beinhaltet. Diese Liste kann leer sein oder ein oder mehrere Elemente beinhalten. Die Argumentnamen können weggelassen werden. `In()` beinhaltet die der Operation übergebenen Argumente und stellt damit faktisch den Empfänger des Prädikats `Content()` der Interaktion dar. `Out()` stellt die Werte dar, die an den Initiator zurück übermittelt werden, und bildet damit eine Art Sender für das Prädikat `Answer` der Interaktion.[2] Ein Element, welches sowohl unter `In` wie unter `Out()` verwendet wird, stellt einen Wert dar, dessen Veränderung durch die Operation für den Initiator zur Verfügung steht.[3] Aus der Semantik der Interaktionsbeziehung im objektorientierten Ansatz ergibt sich, dass die Ausgabe einer Operation über `Out()` an den Initiator der Operation erfolgt. Darüber hinaus muss es sich um eine Interaktionsform handeln, die bidirektionale Interaktion ermöglicht, also sequentiell oder synchronisiert ist.[4]

Die Interaktion stellt die aktuelle Verwendung einer Operation da, während die Operation eine formale Schnittstelle darstellt, die unabhängig von der aktuellen Verwendung ist.[5] Das gleiche Objekt kann auf unterschiedliche Nachrichten mit der gleichen Operation reagieren:[6]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	gibt ein :Action	Eingabe A :Interaction Content(”2”:Tastendruck)	:Taschenrechner	lies Eingabe :Operation In(y:Tastendruck)
:Person	gibt ein :Action	Eingabe B :Interaction Content(”4”:Tastendruck)	:Taschenrechner	lies Eingabe :Operation In(y:Tastendruck)
:Person	gibt ein :Action	Eingabe C :Interaction Content(”÷”:Tastendruck)	:Taschenrechner	lies Eingabe :Operation In(y:Tastendruck)
:Person	gibt ein :Action	Eingabe D:Interaction	:Taschenrechner	lies Eingabe :Operation

[1] Vgl. [Rumbaugh 1999: S. 371, 381].

[2] Entspricht `Out` oder `Return` in der UML. Vgl. [OMG 1999: S. 2-39].

[3] Vgl. [Saake 1993: S. 97].

[4] Vgl. [Saake 1993: S. 96].

[5] Das unter Participant Responsibility dokumentierte Prädikat In() stellt die formalen Argumente der Operation dar, während das unter Connection aufgeführte Prädikat Content() die aktuellen Argumente der Operation beinhaltet. Vgl. den ähnlichen Aufbau interner Beziehungen in Abschnitt 6.4.

[6] Außerdem können unterschiedliche Objekte auf die gleiche Nachricht durch ihre jeweiligen Operationen reagieren, was als Polymorphismus bezeichnet wird.

		Content(“8”:Tastendruck)		In(y:Tastendruck)
:Person	gibt ein :Action	Eingabe E :Interaction Content(”=”:Tastendruck)	:Taschenrechner	lies Eingabe :Operation In(y:Tastendruck)

Skript 102: Inhalt einer Interaktion und Argumente einer Operation

Dies wird deutlich, wenn man die Abbildung auf das Konzeptuelle Schema betrachtet:

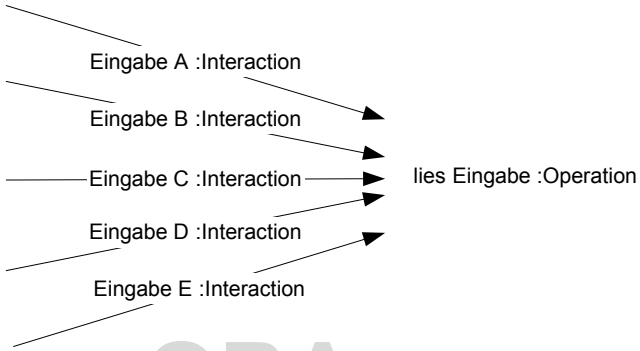


Abbildung 109: Interaktion und Operation im Konzeptuellen Schema

Die Verwendung der Prädikate `Content()` und `Answer()` in Interaktionen und von `In()` und `Out()` in Operationen stellt keinen überflüssigen Luxus dar, sondern ist notwendig, um in Skripten aktuelle und formale Argumente[1] zusammen dokumentieren zu können. Die Tabellenspalte Connection beinhaltet bei Interaktionen die Informationen über die aktuelle Interaktion. Die Tabellenspalten Initiator und Participant Responsibility dagegen beinhalten formale Spezifikationen der beteiligten Konzepte.

6.6.2 Nebenläufigkeit

Die Nebenläufigkeit (Prädikat `Concurrency(`*`Concurrency Kind`*`)`) einer Operation bestimmt die Intraobjekt-Parallelität eines Objekts bei der Ausführung einer Operation. Dadurch wird festgelegt, wie sich das Objekt verhält, wenn es eine Operation X ausführt und eine weitere Aufforderung erhält, diese Operation auszuführen.

[1] Vgl. [Rechenberg 1999: S. 472, 495].

Hier werden für die Terme die in der UML definierten Bezeichner verwendet:[1] *Gepuffert* (`guarded`) bedeutet, dass das Objekt zwar mehrere Aufforderungen entgegennimmt, die Operation auszuführen, diesen Aufforderungen aber durch sequentielle Ausführung der Operation entspricht. Die eintreffenden Aufforderungen werden intern „zwischengespeichert“ und in Reihenfolge ausgeführt.[2] Ein typisches Anwendungsbeispiel für gepufferte Intraobjekt-Parallelität ist der Fall, wenn ein Mitarbeiter einen Arbeitsauftrag erhält, aber erst später bearbeitet, da er im Augenblick des Erhalts einen anderen Vorgang bearbeitet. Aus diesem Grund wird gepufferte Intraobjekt-Parallelität als Standardfall angesehen, der nicht explizit angegeben wird. *Sequentiell* (`sequential`) bedeutet, dass das Objekt nur eine Operation zu einem bestimmten Zeitpunkt ausführen kann und mehrfache Aufforderungen nicht puffert. Kann das Objekt nicht sofort reagieren, wird der Aufforderung nicht entsprochen. Das Ereignis geht „verloren“.[3] Wenn ein Objekt durch asynchrone Interaktion die Aufforderung erhält, eine sequentielle Operation auszuführen, besteht durch die fehlende Synchronisation die Möglichkeit, dass die Operation nicht ausgeführt wird und der Initiator der Operation dies nicht erfährt.

---

[1] Vgl. [Booch 1999: S. 315 f.], [OMG 1999: S. 2-38].

[2] TAYLOR geht davon aus, dass alle Geschäftselemente (Organisationseinheiten, Prozesse und Ressourcen) die Fähigkeit besitzen, angeforderte Operationen zu puffern und nach bestimmten Kriterien auszuführen. Vgl. [Taylor 1995: S. 105 – 124]. Die UML beinhaltet keine Aussagen darüber, in welcher Art und Weise die nächste auszuführende Operation ausgewählt wird. Hierfür ließen sich – ähnlich wie in Simulationssprachen (vgl. [Witte 1994: S. 271 ff.], [Pritsker 1989: S. 47 ff., 68 ff.]) – objektinterne Warteschlangen und ein entsprechendes Prädikat definieren, durch welches festgelegt wird, in welcher Reihenfolge die nächste, in der Warteschlange gepufferte Anforderung einer Operation ausgewählt wird, die zur Ausführung kommen soll. Vgl. zu diesem Punkt auch die Ausführungen von DE CHAMPEAUX [de Champeaux 1993: S. 87, 233] und ALLEN [Allen 1995]. Ein anderes Konzept wird in der Methode ROOM und von PAGE-JONES erwähnt, die Ereignisse Prioritäten zuordnen. Vgl. [Selic 1994: S. 220], [Page-Jones 1994b], [Page-Jones 2000: S. 157]. Beide führen die Idee aber nicht weiter aus, und so wird auch nicht klar, wie der Empfänger der Nachricht mit diesen Prioritäten umgeht. Da es dem objektorientierten Ansatz widersprechen würde, wenn der Initiator Einfluss darauf nimmt, in welcher Reihenfolge der Participant Operationen ausführt, wird diese Idee hier nicht verfolgt. Eine sinnvolle Alternative besteht darin, die Priorität als Bestandteil der übertragenen Informationen durch das Prädikat `Content()` abzubilden, welches vom Participant ausgewertet wird. Auf diese Weise ließe sich beispielsweise durch `Content(High :Priority)` oder `Content(10 :Priority)` eine hohe Priorität und bevorzugte Bearbeitung der Nachricht abbilden. Auf der Seite des Participants müsste definiert werden, in welcher Reihenfolge die gepufferten Operationen ausgeführt werden. Auf diese Weise würde weiterhin – wie es dem objektorientierte Ansatz entspricht – der Participant die Verantwortung haben, angeforderte Operationen in adäquater Frist auszuführen. Vgl. hierzu auch die Modellierung von Prioritätsregeln in Abschnitt 7.4.21.

[3] Vgl. [de Champeaux 1993: S. 87].

Nebenläufig (`concurrent`) bedeutet, dass das Objekt eine Operationen mehrfach parallel ausführen kann. Unbestimmt (`unspecified`) bedeutet, dass die Nebenläufigkeit einer Operation unbekannt ist oder nicht festgelegt werden kann.

Zum Beispiel kann eine Tür in einem bestimmten Moment nur einmal geschlossen werden. Für die Operation `schliessen` der Klasse `Tür` würde folgerichtig `Concurrency(sequential)` gelten. Dagegen kann eine Schreibkraft mehrfach dazu aufgefordert werden, einen Brief zu erstellen. Die entsprechenden Aufforderungen würden nach und nach abgearbeitet werden. Für die Operation `Brief erstellen` der Klasse `Schreibkraft` würde `Concurrency(guarded)` gelten. Schließlich gibt ein Objekt Stromnetz auch dann Strom ab, wenn es mehrfach dazu aufgefordert wird. Für die Operation `Strom abgeben` der Klasse `Stromnetz` würde `Concurrency(concurrent)` gelten.[1]

6.6.3 Sichtbarkeit

Die Sichtbarkeit (*visibility*) einer Operation legt fest, von welchen anderen Objekten eine Operation angefordert werden kann. Die UML unterscheidet – in Anlehnung an gebräuchliche Programmiersprachen – drei Kategorien.[2] Dies wird durch das Prädikat `vis(`*`Visibility Kind`*`)` abgebildet. Öffentlich (`public`) bedeutet, dass die Operation von Objekten anderer Klassen angefordert werden kann. Neben den üblicherweise von außen sichtbaren Operationen werden verborgene Operationen verwendet, die nur vom Objekt selbst aufgerufen werden können.[3] Privat (`private`) bedeutet, dass die Operation nur vom Objekt selbst aufgerufen werden kann. In diesem Fall muss der Klassen- und Objektname von Initiator und Participant übereinstimmen. Geschützt (`protected`) bedeutet, dass die Operation von Objekten erbender Klassen aufgerufen werden kann.

Die Modellierungssprache BON verwendet – in Anlehnung an die Programmiersprache EIFFEL – ein anderes Konzept der Sichtbarkeit. Dort kann für eine Operation jede Klasse angegeben werden, die als Initiator der Operation auftreten kann.[4] Auf diese Weise wird eine wesentlich präzisere Steuerung von Zugriffsmöglichkeiten realisiert. Für die Operation `Anweisung entgegennehmen` einer Klasse `Sachbearbeiter` ist auf diese Weise festzulegen, dass sie nur

---

[1] Siehe den Abschnitt 6.2.11.

[2] Vgl. [Rumbaugh 1999: S. 373, 497]. In der dort verwendeten Terminologie ist der Initiator der Klient einer Klasse.

[3] Dieses Konzept wurde aus C++ übernommen. Dort werden verborgene Operationen als *private member function* bezeichnet. Vgl. [Stroustrup 1992: S. 160], [Stroustrup 2000].

[4] Vgl. [Waldén 1995: S. 39], [Meyer 1997: S. 191], [Meyer 1992b: S. 97].

von Objekten der Klasse `Vorgesetzter` und deren Erben angefordert werden kann.[1] Zwar lässt die UML die Definition weiterer Sichtbarkeiten zu,[2] doch dadurch wird es nicht möglich, einen qualifizierten Exportmechanismus mit den Sichtbarkeitskategorien der UML zu vereinen. Der qualifizierte Exportmechanismus verwendet Klassennamen. Die Sichtbarkeitskategorien der UML verwenden vordefinierte Bezeichner. Würde man beides vermischen, wäre nicht zu unterscheiden, ob eine Operation mit der definierten Sichtbarkeit `public` eine öffentliche Operation ist oder eine, die für eine Klasse `public` exportiert wird. Als Kompromiss bietet sich an, die Sichtbarkeit über zwei getrennte Primitive abzubilden. Das Prädikat `Vis()` legt die Sichtbarkeitskategorie fest. Hier wird als zusätzlicher Term der qualifizierte Export (`qualified`) eingeführt. In diesem Fall beinhaltet das Prädikat `Exp(`*`Identifier List`*`)` eine Liste von Klassennamen, die diese Operation verwenden können. Dies lässt sich zur Modellierung von Befugnissen verwenden. Für die Operation `Mahnstufe aufheben` einer Klasse `Person` könnte spezifiziert werden:

```
Person :Class
Mahnstufe aufheben :Operation
Vis(qualified)
Exp( :Buchhalter, :Geschäftsführer)
```

Durch diesen Mechanismus wird die Überprüfung ermöglicht, ob ein Objekt die Berechtigung besitzt, von einem anderen Objekt eine Operation anzufordern. Folgende Interaktion wäre beispielsweise unzulässig, wenn die Klasse `SB` nicht entweder von einer Klasse `Buchhalter` oder `Geschäftsführer` erbt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	hebt Mahnstufe auf :Action	:Interaction	/Kunde :Person	Mahnstufe aufheben :Operation Vis(qualified) Exp(:Buchhalter, :Geschäftsführer)

Skript 103: Zugriffsberechtigung im Skript

Der qualifizierte Export erlaubt so den Zugriff auf eine Operation nur für alle Objekte der Klassen, die unter `Exp()` angegeben werden.

[1] Die Sichtbarkeit einer Operation einer Klasse muss auch für deren Erben gelten, weil eine erbende Klasse alle Rechte ihrer Vorfahren erbt.

[2] Vgl. [Rumbaugh 1999: S. 498].

6.6.4 Modus der Aktivität

Bei der objektorientierten Modellierung lassen sich – unabhängig von einem bestimmten Problembereich – bestimmte Arten von Aktivitäten differenzieren.[1] Dies sind Aktivitäten, durch die

- Informationen geliefert,
- Objekte verändert,
- Objekte gelöscht oder erzeugt,
- Objekte aus einer Menge von Objekten selektiert,
- Objekte aus einer Menge von Objekten entfernt,
- Objekte in eine Menge von Objekten eingefügt,
- bestimmte Umstände angezeigt werden, oder
- die eine Menge von Objekten nacheinander durchlaufen.

Sowohl beim Anlegen eines neuen Kunden in einem Kundenstamm, beim Einstellen eines neuen Mitarbeiters oder beim Immatrikulieren eines Studenten an einer Hochschule handelt es sich um eine Operation, die ein Kunden-/Mitarbeiter-/Studentenobjekt erzeugt und dieses erzeugte Objekt in eine Menge von Objekten einfügt. Während der Name der Operation eine Bezeichnung darstellt, die aus dem Problembereich stammt (`einstellen`, `erzeugen`, `immatrikulieren`, `erstellen`, `montieren`), wird hier für Aktivitäten das Prädikat `Mode(`*`Activity Mode`*`)` verwendet, um die Art oder Gattung der Aktivität zu bestimmen. Für das Prädikat werden folgende Terme unterschieden:[2]

- Ein *Konstruktor* (`Constructor`) erzeugt Objekte und initialisiert sie gegebenenfalls.
- Ein *Destruktor* (`Destructor`) löscht Objekte.
- Ein *Selektor* (`Selector`) liefert Informationen über Zustand oder Lage eines oder mehrerer Objekte oder selektiert ein oder mehrere Objekte aus einer Menge von Objekten.[3] Selektoren zeichnen sich dadurch aus, dass sie keinen Seiteneffekt besitzen. Auch Aktivitäten, die einen berechneten Wert an ein Objekt liefern, werden als Selektor bezeichnet.

---

[1] Vgl. [Embley 1992: S. 178].

[2] Diese Unterscheidung stellt eine Weiterentwicklung der Klassifizierung von BOOCH dar, die durch die Hinweise von EMBLEY inspiriert wurde. Vgl. [Booch 1995: S. 118], [Embley 1992: S. 178 ff.]. Siehe auch [BalzertH 1999: S. 32 ff.], [Kappel 1996: S. 46], [Berard 1993: S. 139], [Douglas 1999: S. 351].

[3] Diese werden von COLEMAN als *Observer* bezeichnet. Vgl. [Coleman 1992: S. 12].

- Ein *Modifizierer* (`Modifier`) verändert den Zustand oder die Lage eines oder mehrerer existierender Objekte.[1] Auch eine Aktivität, die einen Wert speichert, ist ein Modifizierer.
- Ein *Iterator* (`Iterator`) durchläuft eine Menge von Objekten und führt auf diesen Objekten eine Operation aus. Iteratoren sind von LISKOV und GUTTAG entwickelte Schleifenabstraktionen.[2]
- Ein *Connector* (`Connector`) erzeugt Beziehungen zwischen Objekten.
- Ein *Disconnector* (`Disconnector`) löscht Beziehungen zwischen Objekten.
- Ein *Indikator* (`Indicator`) zeigt einen bestimmten Umstand an. Indikatoren werden für Aktionen verwendet, die in erster Linie die Aufgabe erfüllen, Operationen zu triggern. Auch das Auftreten einer Ausnahmebedingung wird durch einen Indikator angezeigt.

Nur für den Selektor existiert in der UML ein Äquivalent in dem Attribut *Query* für Operationen ohne Seiteneffekte, die aus einer Menge von Objekten jene selektieren, auf die eine vorgegebene Bedingung zutrifft.[3] Andere Bezeichnungen werden manchmal in informeller Weise in Interaktionsdiagrammen in Form von Stereotypen verwendet.[4] Dies widerspricht aber der Idee der Stereotype, diese zu problemspezifischen Erweiterungen des Metamodells zu verwenden.

Konstruktoren besitzen eine Sonderstellung in der objektorientierten Modellierung. Eine Konstruktoroperation erzeugt ein Objekt durch eine Interaktion mit einem noch nicht existierenden Objekt. Wenn ein Sachbearbeiter ein Personenobjekt erzeugt, stellt dies eine Interaktion mit dem noch nicht existierenden Objekt dar:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	legt Kunden an :Action	:Interaction	:Kunde	anlegen :Operation Mode(Constructor)

Skript 104: Anlegen eines Objekts

Hier sind auch andere Lösungen möglich, die mehr den Abläufen in einem objektorientierten Programm oder einem objektorientierten DBMS entsprechen. In diesem Fall würde die Interaktion

---

[1] Auch Operationen, die Beziehungen zwischen einem oder mehreren Objekten verändern, können als Modifizierer angesehen werden. Sie werden hier jedoch separat aufgeführt und als Connector und Disconnector bezeichnet. Streng genommen sind die beiden Modi Spezialformen des Modifizierers.

[2] Vgl. [Liskov 1986: S. 118 ff.]

[3] Vgl. [Rumbaugh 1999: S. 373].

[4] Vgl. [Booch 1999: S. 245].

mit der Klasse `Kunde` stattfinden, die zur Laufzeit ein neues Kundenobjekt erzeugt. Die Operation `anlegen` wäre eine Klassenoperation, d. h. eine Operation, die nicht von den Objekten, sondern von der Klasse selbst ausgeführt wird. Dadurch ist dann aber der Begriff der Klasse als Instanzierungsvorschrift für Objekte ausgehöhlt, da nun auch Klassen selbst Objekte von Metaklassen darstellen, die Verhalten zeigen. Diese Metaklassen können wiederum Exemplare von Meta-Metaklassen sein usw. Dieser – auf den ersten Blick – stringente Ansatz entpuppt sich als unnötig kompliziert und schwer nachvollziehbar.

Iteratoroperationen finden häufig in mereologischen Strukturen Verwendung, um Operationen auf den einzelnen Elementen der Struktur durchzuführen:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Belegschaft		:Membership	:Mitarbeiter	
...	...	:Interaction	:Belegschaft	Gehalt um 10 % erhöhen :Operation Mode(Iterator)
:Belegschaft	durchläuft alle Mitarbeiter :Action Mode(Iterator)	:Interaction	:Mitarbeiter	Gehalt um 10 % erhöhen :Operation Mode(Modifier)

Skript 105: Verwendung eines Iterators

Indikatoren signalisieren bestimmte Situationen, auf die der Participant reagieren soll:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Glas	fällt vom Tisch :Action Mode(Indicator)	:Interaction	:Person	fängt Glas auf :Operation

Skript 106: Verwendung eines Indikators

Die Wahl der Bezeichnung Modus geschieht nicht aus Zufall. Durch dieses Prädikat wird der enge Zusammenhang zwischen dem Modus einer Interaktion und dem Modus der beteiligten Aktivitäten ausgedrückt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	ermittelt Werte :Action Mode(Selector)	:Interaction Mode(Request)	:Kunde Stpe(Business Object)	Umsatzzahlen ermitteln :Operation Mode(Selector)

Skript 107: Zusammenhang zwischen dem Modus von Aktivitäten und der Absicht einer Interaktion

Modifizierer stehen dabei in direktem Zusammenhang mit Zustandsveränderungen oder Attributzugriffen des Participants. Attributzugriffe sind schon in Abschnitt 6.4 dargestellt worden. Zustandsveränderungen können entsprechend im Skript dargestellt werden:[1]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Person	schließt Tür :Action	:Interaction	:Tür	schließen :Operation Mode(Modifier)
	:Tür	offen :State	:StateChange Act(schließen :Operation)	:Tür	geschlossen :State

Skript 108: Modifizierer und Zustandsveränderung des Participants

Werden dagegen im Rahmen von Interaktionen Beziehungen zwischen Objekten erzeugt oder beendet (`Connector` oder `Disconnector`), verändert sich dadurch die Objektkonfiguration.[2] Dies kann in der UML mit den aktuell vorhandenen Mitteln in Interaktionsdiagrammen nicht dargestellt werden.[3] Hierfür wurde das Prädikat `Mode(`*`Change Mode`*`)` eingeführt, welches für den Beziehungsknotentyp *REFERENTIALRELATIONSHIP* definiert ist. Auf diese Weise kann dokumentiert werden, dass eine referentielle Beziehungen angelegt (`create`) oder beendet (`destroy`) wurde:[4]

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Schmidt :AL[5]	nimmt Mitarbeiter in Abteilung auf :Action	:Interaction Content(Müller :Mitarbeiter)	Nr. 6 :Abteilung	Mitarbeiter hinzufügen :Operation Mode(Connector) In(:Mitarbeiter)
Nr. 6 :Abteilung		:Membership Act(Mitarbeiter hinzufügen :Operation) Mode(create)	Müller :Mitarbeiter	

Skript 109: Interaktion und Veränderung der Objektkonfiguration (1)

[1] Vgl. auch Abschnitt 6.3.

[2] Das ist der Teil des konkreten Zustands, der durch die aktuellen Beziehungen eines Objekts gebildet wird.

[3] Dem Verfasser ist nur aus der Methode CATALYSIS die Möglichkeit bekannt, die Effekte von Operationen auf die Objektstruktur durch sog. Schnappschüsse (*snapshots*) zu dokumentieren. Vgl. [D'Souza 1998: S. 2-73, 3-112, 3-118].

[4] Vgl. Abschnitt 6.5.4.

[5] Abteilungsleiter.

Die zusätzliche Membership-Beziehung verdeutlicht, dass durch die Operation des Participants eine Veränderung der Objektkonfiguration stattfindet. Der Mitarbeiter Müller wird zu einem Mitglied der Abteilung Nr. 6. Sollte der Mitarbeiter mit dem Namen Müller entlassen werden, kann der Abbau der Objektbeziehung in der gleichen Weise dokumentiert werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Schmidt :AL	entlässt Mitarbeiter :Action	:Interaction Content(Müller :Mitarbeiter)	Nr. 6 :Abteilung	Mitarbeiter entfernen :Operation Mode(Disconnector) In(:Mitarbeiter)
Nr. 6 :Abteilung		:Membership Act(Mitarbeiter entfernen :Operation) Mode(destroy)	Müller :Mitarbeiter	

Skript 110: Interaktion und Veränderung der Objektkonfiguration (2)

Diese Form simultaner Modellierung der dynamischen und statischen Sicht ist in anderen Modellierungsansätzen nicht möglich.

6.6.5 Vor- und Nachbedingungen

Die Semantik einer Operation wird nicht imperativ durch Angabe einer Verarbeitungsvorschrift festgelegt.[1] Dies würde dem Prinzip des *Information Hiding* widersprechen (vgl. Abschnitt 2.2.4, 2.2.6). Außerdem sollen keine Realisierungsentscheidungen durch Vorgabe eines Algorithmus vorweggenommen werden. Stattdessen wird die Semantik einer Operation deklarativ durch ihre Effekte in Form von Vor- und Nachbedingungen beschrieben. Es wird also nicht beschrieben, wie eine Operation ausgeführt wird, sondern welche Effekte ihre Ausführung hat. Dafür werden die Prädikate `Pre(`*`Expression`*`)` und `Post(`*`Expression`*`)` verwendet. Zwar sind auch in der UML die Stereotype *Precondition* und *Postcondition* definiert, die Modellierungs*sprache* UML erklärt aber nicht, wie man sie sachgerecht verwendet.[2] Die Vor- und Nachbedingungen einer Operation

[1] In der UML soll die Semantik von Operationen durch das Schlüsselwort/Wert-Paar (*tagged value*) `semantics` festgehalten werden. Dessen Wert soll die Semantik der Operation beinhalten (vgl. [OMG 1999: S. 2-39]). Dessen Verwendung wird aber nicht erläutert und bleibt unklar. Die hier entwickelte Konzeption stellt eine Präzisierung dar, da die Semantik von Operationen durch mehrere Prädikate abgebildet wird.

[2] Hier wird die Differenzierung zwischen Sprachdefinition und Methode deutlich. Die UML erklärt ein Vokabular und die Syntax zur objektorientierten Modellierung. Sie erklärt nicht, wie man stilistisch sauber modelliert. Dies ist in etwa mit einer Definition der deutschen Sprache durch den Duden der Recht-

können Aussagen über die Argumente der Operation beinhalten, über den internen Zustand des Lieferantenobjekts vor und nach der Ausführung der Operation, aber auch über den Zustand anderer Objekte im System.[1]

Die Vorbedingung ist ein logischer Ausdruck, der wahr sein muss, damit der Participant der Aufforderung nachkommt, eine vom Initiator angeforderte Operation auszuführen. Sie ist kein Test zur optionalen Ausführung einer Operation,[2] sondern eine Sicherheits- bzw. Anwendbarkeitsbedingung.[3] Auf dieses Weise wird ausgedrückt, dass eine Operation nur ausgeführt werden kann, wenn die in der Vorbedingung aufgeführten Bedingungen erfüllt sind. Die Nachbedingung ist ein logischer Ausdruck, der wahr ist, wenn die Operation abgeschlossen wurde.[4] Die Nachbedingung beschreibt den Effekt der Operation.[5] Für die Operation x einer Klasse K in der Notation der *guarded commands* von DIJKSTRA gilt:[6] `{Pre Op x} Klasse K.Operation x {Post Op x }`. Dies bedeutet, dass vor der Ausführung der Operation x einer Klasse K die Vorbedingung $Pre_{Op\ x}$ gilt und nach der Ausführung der Operation die Nachbedingung $Post_{Op\ x}$. Kann eine Operation mehrere unterschiedliche Ergebnisse haben, besteht die Nachbedingung aus mehreren Teilausdrücken, von denen mindestens einer wahr wird.

---

schreibung und Grammatik vergleichbar: Diese beiden Bände erklären auch nicht, wie man stilistisch sauber schreibt. Wäre die UML eine Methode, würde – nach allgemein verbreiteter Auffassung über den Zweck einer Methode – auch erklärt werden, wie man mit diesen Primitiven sauber modelliert.

[1] [Kappel 1996: S. 44].

[2] Vgl. [Rumbaugh 1999: S. 392].

[3] Vgl. [Saake 1993: S. 49].

[4] Die Begriffe Vor- und Nachbedingung werden sowohl im Singular wie auch im Plural verwendet. Dies ergibt sich aus dem Zusammenhang, dass eine Operation eine Vorbedingung und eine Nachbedingung besitzt, die allerdings wiederum aus mehreren einzelnen Bedingungen bestehen können.

[5] Das Konzept der Vor- und Nachbedingungen stammt aus dem Bereich der formalen Spezifikationssprachen, wo es zur Beschreibung der Semantik von Funktionen verwendet wird. Vgl. [Dawes 1991], [Sheppard 1994], [Andrews 1991], [Spivey 1992], [Wordsworth 1992], [Andrews 1997], [Turner 1994: S. 40]. Es findet mittlerweile zunehmende Verwendung im Bereich der Softwareentwicklung, insbesondere in der Anforderungsanalyse. Vgl. [Partsch 1998: S. 95]. Auch in der objektorientierten Modellierung wird es in einigen Modellierungsansätzen verwendet. Hier ist insbesondere BON zu nennen. Vgl. [Waldén 1995]. In der UML werden Vor- und Nachbedingungen üblicherweise nur in Form von Kommentaren verwendet. Vgl. [Rumbaugh 1999: S. 391 ff.], [Hitz 1999: S. 75]. Hier bietet die zur UML gehörende Spezifikationssprache OCL (vgl. [Warmer 1999]) möglicherweise eine sinnvolle Erweiterung, auch wenn sie nicht die Mächtigkeit formaler Spezifikationssprachen wie VDM oder Z besitzt.

[6] Vgl. [Amstel 1987], [Meyer 1997: S. 334]. Formal stellen Vor- und Nachbedingungen beliebige prädikatenlogische Formeln dar. Vgl. [Lipeck 1989: S. 94].

Die erfolgreiche Ausführung einer Operation kann in diesem Sinne als Transaktion betrachtet werden, durch die ein oder mehrere Objekte von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt werden.[1] Eine Operation, die immer ausgeführt werden kann, erhält – nach MEYER – implizit die Vorbedingung logisch wahr (`Pre(True)`). So könnte als Nachbedingung der Operation `Brief erstellen` einer Klasse `Schreibkraft` dokumentiert werden. Im folgenden Beispiel wird objektorientiert der Umstand modelliert, dass durch eine Operation Kosten verursacht werden, die Kostenstellen und Kostenträgern zugeordnet werden. Die dabei verwendeten Objekte sind typischerweise Kostenarten, Kostenstellen und Kostenträger. Durch die *Kostenarten* wird festgehalten, welche Kosten angefallen sind. Durch die *Kostenträger* wird festgehalten, wofür Kosten angefallen sind. Durch die *Kostenstellen* werden die Kostenarten auf die Betriebsbereiche verteilt, in denen sie angefallen sind:[2]

```
Schreibkraft :Class
Brief erstellen :Operation
In( Adressat: Kundenadresse, Mahnung: Inhalt)
Post( Brief.State = erstellt;
      Betrag X1 der Kostenart KA1 ist auf Kostenstelle KS verbucht;
      Betrag X2 der Kostenart KA2 ist für Kostenträger KT verbucht)[3]
```


Die Nachbedingung einer Operation beschreibt ihre Effekte unabhängig von den auszuführenden Aktivitäten. Auch die Zustandsveränderungen von Objekten werden durch die Nachbedingung dokumentiert.[4] Davon bleibt aber unberührt, die im Rahmen der Operation auszuführenden Aktivitäten und Zustandsveränderungen als Skript zu dokumentieren. Dort könnte dokumentiert werden, dass im Rahmen der Operation `Brief erstellen` ein Brief entsteht und Kosten verursacht werden, die einem Kostenträger zugeordnet werden. Das folgende Beispiel dient der Veranschaulichung:

---

[1] Dadurch wird keine Aussage über das Vorliegen einer Datenbanktransaktion getätigt. Es ist im Einzelfall zu prüfen, ob die Operation atomar, konsistenzerhaltend, isoliert und dauerhaft (ACID-Kriterien; vgl. [Lockemann 1993: S. 415 ff.]) ist.

[2] Vgl. [Wöhe 1986: S. 1157], [Heinen 1991c].

[3] Es wird hier aus zwei Gründen davon Abstand genommen, die Vor- und Nachbedingungen in einer formalen Notation anzugeben: Die Verständlichkeit der Darstellung würde darunter leiden, und die Notation müsste hier erst noch eingeführt werden. Davon abgesehen, spricht nichts dagegen, eine formale Notation (VDM oder OCL) zu verwenden.

[4] Dieser Ansatz folgt der von COOK und DANIELS entwickelten Konzeption, das Verhalten von Operationen zu spezifizieren. Vgl. [Cook 1994]. Auf diese Weise wird vermieden, alle eintretenden Zustandsveränderungen als separate Zeile in einem Skript modellieren zu müssen.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	:Sekretärin	Brief erstellen :Operation
	:Sekretärin	schreibt Brief :Action	A :Interaction	:Brief	wird erzeugt :Operation
	:Sekretärin	schreibt Brief :Action	B :Interaction Content(X1 :Betrag, KA1 :Kostenart)	KS :Kostenstelle	belasten :Operation
	:Brief	Initial :State	erstellen :Statechange Act(wird erzeugt :Operation)	:Brief	erstellt :State
	:Brief	wird erzeugt :Action	D :Interaction Content(X2 :Betrag, KA2 :Kostenart)	KT :Kostenträger	belasten :Operation

Skript 111: Im Rahmen einer Operation stattfindende Aktivitäten als Zeilen im Skript

Die Zeilen stellen die Interaktionen und Zustandsveränderungen dar, die im Rahmen der Operation stattfinden.[1]

Während die Nachbedingung die Effekte der erfolgreichen Ausführung einer Operation dokumentiert, beinhaltet die Vorbedingung die notwendigen Bedingungen für die Herbeiführung der Nachbedingung. Die Nachbedingung braucht nur erfüllt zu werden, wenn die Vorbedingung erfüllt ist. Dies soll am Beispiel eines Barkaufs einer Klasse `Person` verdeutlicht werden:

```
Person :Class
bezahlen :Operation
In( Höhe :Betrag)
Post( Person. Barvermögen neu = Person. Barvermögen alt - Höhe)
```

Dieser Effekt kann nur eintreten, wenn folgende Bedingung gilt:

```
Pre( Person. Barvermögen > = Höhe)
```

Auch die Vorbedingung kann beliebig umfangreich sein. Vorbedingungen sind Ausdrücke mit[2]

- Bezug auf Attributwerte vor dem Aufruf der Operation.
- Bezug auf den Zustand des Objekts.
- Operationen, die keine Seiteneffekte bewirken.

Dabei werden Selektoren[3] verwendet, um den Zustand anderer Objekte zu ermitteln. Werden der Zustand oder die Attributwerte der aktuellen Klasse/des aktuellen Objekts verwendet, ist eine

[1] Dieser Zusammenhang wird noch im weiteren Verlauf in Abschnitt 6.12 präzisiert.

[2] Vgl. [Saake 1993: S. 49].

[3] Vgl. Abschnitt 6.6.4.

vollständige Referenzierung nicht notwendig. Auf diese Weise wird mit Hilfe von Vorbedingungen abgebildet, was passiert sein muss, damit eine Operation anwendbar ist. Dies sei an folgendem Beispiel demonstriert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Schreibkraft	erstellt Brief :Action	anfertigen :Interaction	:Brief	erstellen :Operation Pre(State = nicht erstellt) Post(State = versandfertig)
	:Schreibkraft	verschickt Brief :Action	versenden :Interaction	:Brief	versenden :Operation Pre(State= versandfertig) Post(State=abgeschickt)

Skript 112: Abbildung von Abfolgen durch Vorbedingungen (1)

Ein Brief muss versandfertig sein, damit er verschickt werden kann. Die Anordnung der Zeilen in dem Skript suggeriert die Notwendigkeit einer Abfolge von Interaktionen, die weder beabsichtigt noch notwendig ist. Vor der Interaktion `versenden` muss die Interaktion `anfertigen` überhaupt nicht stattgefunden haben. Die Reihenfolge der Zeilen ist für das Verständnis des Lesers wichtig. Aber welche Reihenfolge in der Ausführung von Interaktionen tatsächlich erlaubt sind, wird durch die Vor- und Nachbedingungen der Operationen festgelegt. Dies wird beim Vergleich mit dem folgenden Skript deutlich, in dem der Brief durch eine andere Interaktion (`erstellen`) versandfertig wird. Auch diese Interaktion führt eine Nachbedingung herbei, der der Vorbedingung der Operation `versenden` genügt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	schreibt Brief :Action	erstellen :Interaction	:Brief	erstellen :Operation Pre(State = nicht erstellt) Post(State = versandfertig)
	:SB	übergibt Brief :Action	übergeben :Interaction Content(:Brief)	:Schreibkraft	erhält Brief :Operation In(:Brief)
	:Schreib-kraft	verschickt Brief :Action	versenden :Interaction	:Brief	versenden :Operation Pre(State= versandfertig) Post(State=abgeschickt)

Skript 113: Abbildung von Abfolgen durch Vorbedingungen (2)

Vor- und Nachbedingungen können starr modellierte Abfolgen ersetzen. Denn wichtig ist eigentlich nur, dass die in der Vorbedingung angegebenen Bedingungen eingehalten werden, d. h., dass vorher bestimmte Ergebnisse herbeigeführt worden sind. Die Verwendung von Vorbedingungen ist ein erheblich präziseres und aussagefähigeres Instrument, mögliche Abfolgen zu spezifizieren, als dies mit Fluss- oder Prozessdiagrammen möglich ist, weil dadurch deutlich wird,

warum eine bestimmte Abfolge eingehalten werden muss. Untersucht man die Vor- und Nachbedingungen von in Prozessen stattfindenden Operationen, wird häufig deutlich, dass bisher eigentlich als notwendig betrachtete Reihenfolgen nur eine von mehreren möglichen Varianten der Ausführung darstellen. Die letzten beiden Skripte sind ein gutes Beispiel dafür. Ein weiterer Vorteil der Verwendung von Vor- und Nachbedingungen besteht darin, dass sie ein Bestandteil der Spezifikation der Operation sind und Eingang in das Konzeptuelle Schema finden. Dadurch sind die Bedingungen für die Ausführung der Operation unabhängig von ihrer Verwendung in einem Skript als Bestandteil des Protokolls einer Klasse dokumentiert. Dies entspricht nach MEYER eher den Prinzipien des Objektansatzes als starr festgelegte Reihenfolgen.[1]

Für eine Operation `Brief erstellen` einer Schreibkraft könnte folgende Nachbedingung formuliert werden:

```
Schreibkraft :Class
Brief erstellen :Operation
In( Adressat: Kundenadresse, Mahnung: Inhalt)
Pre( Schreibkraft hat freie Kapazität)
Post(( Brief.State = erstellt;
       Betrag X1 der Kostenart KA1 ist auf Kostenstelle KS verbucht;
       Betrag X2 der Kostenart KA2 ist für Kostenträger KT verbucht) OR
       (Brief.State = nicht erstellt;
       Betrag X3 der Kostenart KA1 ist auf Kostenstelle KS verbucht;
       Betrag X4 der Kostenart KA2 ist für Kostenträger KT verbucht))
```

Offensichtlich ist die erfolgreiche Ausführung einer Operation ein rein formales Kriterium, welches von der formulierten Nachbedingung abhängig ist. Die Operation ist auch dann erfolgreich, wenn der Brief nicht erstellt wurde – allerdings mit unterschiedlichen Ergebnissen. Im weiteren Verlauf des Prozesses können durch bedingte Interaktionen (abgebildet mit `Guard(Brief ist erstellt)` und `Guard(Brief ist nicht erstellt)`) die unterschiedlichen Ergebnisse der Operation berücksichtigt werden. Über die Nachbedingung können auch mehrere unterschiedliche Ergebnisse abgebildet werden, die nach üblichem Sprachgebrauch als erfolgreiche Ausführung bezeichnet werden:

```
Post( (Brief. State = erstellt;
       Typ des Briefs ist Formbrief;
       Betrag X1 ist auf Kostenstelle KS verbucht;
       Betrag X2 ist für Kostenträger KT verbucht) OR
```

[1] In [Meyer 1997] kritisiert MEYER die Verwendung von ablauforientierten Verfahren im Rahmen des Objektansatzes.

```
(Brief. State = erstellt;
Typ des Briefs ist Individualbrief;
Betrag X1 * Multiplikator ist auf Kostenstelle KS verbucht;
Betrag X2 * Multiplikator ist für Kostenträger KT verbucht))
```

Bei der Spezifikation von Operationen durch Vor- und Nachbedingungen sind zwei Regeln zu beachten:[1]

- Die *Integritätsregel* besagt, dass Operationen integritätswahrend sein müssen. Wenn eine Vorbedingung im aktuellen Zustand erfüllt ist, muss nach der erfolgreichen Ausführung der Operation im Folgezustand die zugehörige Nachbedingung gelten. Ein korrekter Zustand muss wieder in einen korrekten Zustand überführt werden.
- Die *Minimalitätsregel* bzw. *implizite Rahmenregel* besagt, dass die Änderung durch Operationen möglichst minimal sein sollte, um die Nachbedingung zu erfüllen. Nach SAAKE sollte die Faustformel angewendet werden, dass durch eine Operation nur explizit erwähnte Objekte verändert werden sollten. Alle Objekte, die in der Nachbedingung nicht erwähnt werden, werden auch nicht verändert.

Die Rückgaben an den Initiator erfolgen über die im Prädikat `Out()` aufgeführten Bezeichner. Die folgenden Klauseln demonstrieren die Definition einer Rückgabe durch die Nachbedingung:

```
Lektor :Class
korrigieren :Operation
In( Eingabe: Dokument)
Out( Eingabe: Dokument, Anmerkung: Kommentar )
Pre( True )
Post( (Eingabe. Korrektur = True;
      Eingabe. Inhalt = Veränderter Text;
      Anmerkung. Inhalt = Anmerkungen zur Korrektur.) OR
      (Eingabe. Korrektur = False;
      Anmerkung. Inhalt = NIL.))
```

Vor- und Nachbedingungen reduzieren erheblich den Modellierungsaufwand. Statt alle möglichen Situationen abbilden zu müssen, die bei der Ausführung einer Operation auftreten können, werden die Bedingungen dokumentiert, unter denen eine Operation erfolgreich ausgeführt werden kann. Nur wenn es im Rahmen der Modellierung von Interesse ist, mögliche unterschiedliche Ergebnisse bei der Ausführung einer Operation zu betrachten, finden diese Eingang in das Modell. Abhängig von den unterschiedlichen Ergebnissen kann dann der weitere Verlauf differenziert werden.

6.6.6 Fehler

Wenn die Nachbedingung die – wie auch immer formulierte – erfolgreiche Ausführung der Operation dokumentiert, stellt sich die Frage, was beim Scheitern einer Operation geschieht. Man benötigt eine Abstraktion, um zu dokumentieren, unter welchen Bedingungen die Operation scheitert, obwohl die Vorbedingung eingehalten wurde. In der UML werden solche Situationen in Form von Ausnahmen als Abhängigkeitsbeziehung zwischen der Operation und besonderen Ausnahmeklassen mittels des Stereotyps «send» abgebildet.[2] Das hier verwendete Konzept basiert auf dem allgemeineren Prinzip des *Fehlerbedingungsblocks* (*error definition block*; hier durch das Prädikat `Err(Error Condition)` dokumentiert), wie er von LISKOV und GUTTAG für CLU entwickelt wurde und auch in der Spezifikationssprache VDM existiert.[3] Diese Fehlerbedingung ist eine Liste von Situationen, die eintreten können, wenn eine Operation trotz eingehaltener Vorbedingung ihre Nachbedingung nicht erfüllen kann. Dies soll an folgendem Beispiel erläutert werden:

```
Tür :Class
schließen :Operation
Pre( Tür. geschlossen =FALSE)
Post( Tür. geschlossen =TRUE)
```

Üblicherweise kann eine Tür geschlossen werden, wenn sie nicht schon geschlossen ist. Wenn die Operation erfolgreich abgeschlossen ist, ist die Tür geschlossen. Wenn die Türangel klemmt oder die Tür aus den Angeln fällt, stellt dies einen Fehler dar, da die Operation trotz eingehaltener Vorbedingung nicht ausgeführt werden kann. Eine Fehlerbedingung wird in der Form

```
<Error Condition>::=1{<Bezeichner>
                     (+ „:“ +<Fehlerbedingung>)
                     (+ „->“ +<Ausdruck>)
                     + „;“ }N.
```

---

[1] Vgl. [Saake 1993: S. 32] und [Lipeck 1989: S. 98].

[2] Vgl. [Booch 1999: S. 285].

[3] Selbst anspruchsvolle Arbeiten über objektorientierte Modellierung befassen sich nicht oder nur oberflächlich mit der Thematik von scheiternden Operationen und ihrer Fehlerbehandlung. Vgl. [Balzert 1996], [Desfray 1994], [Saake 1993], [Booch 1995], [Rumbaugh 1993]. EMBLEY und COOK erörtern die Fehlerbehandlung zumindest innerhalb von Zustandsdiagrammen. Vgl. [Embley 1992: S. 82 ff.], [Cook 1994: S. 212]. Aus diesem Grund wurde auf die Konzepte der formalen Spezifikationssprache VDM und VDM++ und dem Konzept der objektorientierten Entwurfs- und Programmiersprache EIFFEL zurückgegriffen. Vgl. [Jones 1990: S. 221], [Dawes 1991: S. 134 ff.], [Sheppard 1994: S. 332 ff.], [IACS 1996: S. 58], [Dürr 1995: S. 25], [Turner 1994: S. 40 ff.], [Meyer 1997: S. 411 ff.], [Liskov 1986: S. 99 ff.].

angegeben. Durch den Bezeichner erhält der Fehler einen Namen. Die Angabe eines Namens ist immer notwendig. Die Fehlerbedingung dokumentiert, durch welchen Umstand dieser Fehler eintreten soll. Ihre Angabe ist optional. Ebenfalls optional kann ein weiterer Ausdruck angegeben werden, durch den beispielsweise Attribute des Objekts gesetzt werden. Auf diese Weise kann die Spezifikation der Operation `schliessen` wie folgt erweitert werden:

```
Err( Tür verklemmt: Tür. Lage = klemmt -> Tür. geschlossen =FALSE;
      Tür hingefallen: Tür. Lage = waagerecht -> Tür. geschlossen =FALSE;)
```

Durch den Bezeichner (`Tür verklemmt` und `Tür hingefallen`) erhält die Fehlersituation einen eindeutigen Namen. Dieser Name wird im Skript verwendet, um in Abhängigkeit von einem eingetretenen Fehler zu reagieren.[1] Die Fehlerbedingung dokumentiert, durch welchen Umstand die Operation nicht erfolgreich ist. Der Ausdruck dokumentiert die Folge der Fehlerbedingung. Eine solche Fehlerbedingung kann, wie SHEPPARD anmerkt,[2] auch als Term der Nachbedingung formuliert werden:

```
Tür :Class
schließen :Operation
Pre( Tür. geschlossen = FALSE)
Post( (Tür. geschlossen =TRUE) OR
      (Tür. Lage = klemmt AND Tür. geschlossen =FALSE) OR
      (Tür. Lage = waagerecht AND Tür. geschlossen =FALSE))
```


Formal tritt ein Fehler dann ein, wenn eine Operation trotz eingehaltener Vorbedingung ihre Nachbedingung nicht erfüllt. Bei der Formulierung dieser Nachbedingung wird davon ausgegangen, dass die Klasse `Tür` zwei Attribute mit den Namen `Lage` und `geschlossen` besitzt. Syntaktisch richtig, aber kontraintuitiv ist hier modelliert, dass die Operation `schließen` selbst dann erfolgreich ist, wenn die Tür nicht geschlossen ist. Dies verdeutlicht noch einmal, dass die Definition eines Fehlers vom Standpunkt des Modellierenden abhängig ist, der darüber entscheidet, was er als Fehler ansieht.

6.6.7 Ziele

Ein *Ziel* ist die Vorstellung eines erstrebenswerten bzw. zu verwirklichenden Zustandes.[3] Ziele existieren nicht als Konstrukt in der objektorientierten Modellierung. Zwar findet in der UML der

---

[1] Vgl. Abschnitt 6.6.10.

[2] [Sheppard 1994: S. 334].

[3] Vgl. [Nordsieck 1955: S. 28], [Kosiol 1962: S. 45, 100], [Schulte-Zurhausen 1999: S. 71]. Für eine weitergehende Darstellung der Begriffe Ziel, Zielsystem, Zielformulierung vgl. [Nagel 1992: Sp. 2626 –

Nachrichtenaustausch einer Kollaboration statt, um ein bestimmtes Ziel zu erreichen,[1] aber das Ziel selbst wird nicht erfasst, sondern ist implizit im Namen der Kollaboration enthalten. Betrachtet man Ziele als anzustrebende Zustände, liegt die Ursache auf der Hand: Das Ziel einer Operation kann durch ihre Nachbedingung mittels des Prädikats `Post()` formuliert werden. Meistens ist das Ziel, mit dem eine Operation ausgeführt wird, implizit im Namen der Operation enthalten. Dies führt häufig zu dem Problem, dass die Ziele nicht deutlich werden oder nur dem Autor des Modells deutlich sind. Die Verwendung der Nachbedingung hat den Nachteil, dass bei komplexen Nachbedingungen nicht erkennbar ist, welche Klauseln die Effekte der Operation beschreiben und welche Klausel das Ziel der Operation darstellt.[2]

Hier ist ein Vorschlag von BOCK als eine Verbesserung bestehender objektorientierter Modellierungsansätze zu werten.[3] Er verwendet Ziele als eine besondere Form der Nachbedingung einer Operation, in der explizit formuliert wird, welches Ziel eine Operation verfolgt bzw. welches Ergebnis gewünscht ist. Das Ziel einer Aktivität wird dabei durch das Prädikat `Goal(`*`Expression`*`)` abgebildet.

```
Position :Class
einstellen :Operation
Goal( Position ist besetzt)
```
[4]

Stellt der Ausdruck in der Nachbedingung einer Operation – oder ein Teil von ihr – ein Ziel dar, wird dieser Ausdruck redundant als Ziel festgehalten. Dadurch kann der „Sinn und Zweck“ einer Operation von dem, was allgemein und immer garantiert werden kann, unterschieden werden. Dies sei durch folgendes Beispiel erläutert:

```
Schreibkraft :Class
Brief erstellen :Operation
```

2652], [Hill 1994: S. 24], [Heinen 1985: S. 93 ff.], [Heinen 1971]. Für die Bewertung von Zielen im Führungsprozess vgl. [Adam 1996: S. 99 ff.]. Auf die Problematik von Zielbildung, Inhalt und Ordnung von Unternehmenszielen, Gestaltung des Zielsystems, Beziehungen zwischen Zielen etc. wird hier nicht weiter eingegangen, da diese Fragestellungen für den Fortgang dieser Arbeit von geringem Interesse sind. Vgl. dazu [Heinen 1971], [Hamel 1992]. Hier mag die vereinfachende Annahme genügen, dass durch Aufgaben und Prozesse ein Ziel erreicht werden soll, welches sich in Form eines Zustandes formulieren lässt. Es handelt sich damit um operationale Ziele [Heinen 1971: S. 115] in der Art, dass festgestellt werden kann, ob das als Zustand formulierte Ziel erreicht wird. Dabei können sowohl Sach- wie auch Formalziele formuliert werden. Vgl. [Wild 1966: S. 90].

[1] Vgl. [Rumbaugh 1999: S. 86, 194].

[2] Siehe die Nachbedingung auf Seite 313.

[3] Vgl. [Bock 2000], [Bock 2001].

[4] Das Beispiel stammt aus [Bock 2000: S. 48] und wurde in die hier verwendete Notation umgeformt.

```
Pre( Schreibkraft hat freie Kapazität)
Post( Brief.State = erstellt OR Brief.State = nicht erstellt)
Goal( Brief.State = erstellt)
```

Durch diese Kombination von Vor- und Nachbedingung wurde festgelegt, dass nach der Ausführung der Operation der Brief zwei unterschiedliche Zustände haben kann. Das Ziel der Operation ist dabei selbstverständlich, dass der Brief erstellt ist.

6.6.8 Klassenoperationen und Externe Referenzen

Eine Klassenoperation ist eine Operation, die nicht einem einzelnen Objekt, sondern allen Objekten einer Klasse zugeordnet ist. Man unterscheidet also Klassen- und Exemplaroperationen. Dies wird durch das Prädikat `Qualifier(`*`Identifier List`*`)` dokumentiert. Für Operationen werden die Terme `Class` und `Instance` verwendet, wobei `Instance` für Exemplaroperationen der Vorgabewert ist, der nicht angegeben werden muss.

Die Eingabe einer Operation umfasst nur die Objekte, die durch eine Interaktion übermittelt werden. Dies erscheint wenig spektakulär, hat aber weitreichende Konsequenzen für die Modellierung. Betrachten wir eine abweichend spezifizierte Operation `Brief erstellen` der Klasse `Schreibkraft`:

```
Schreibkraft :Class
Brief erstellen :Operation
In( Adressat: Kundennummer, Mahnung :Inhalt )
Pre( Schreibkraft hat freie Kapazität)
Post( Brief ist erstellt;
          Betrag X1 ist auf Kostenstelle KS verbucht;
          Betrag X2 ist für Kostenträger KT verbucht)
```

Die Schreibkraft erhält als Eingabe nur die Kundennummer und den Inhalt des Briefes. Die zugehörigen Adressdaten wird sie über eine Kundendatei in Erfahrung bringen. Das folgende Skript beinhaltet die von der Schreibkraft ausgeübte Aktion:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	...	...
	:Schreib-kraft	ermittelt Kundendaten :Action	Suche nach Kundendaten :Interaction Content(:Kundennummer) Answer(:Kunde)	:Kundendatei	Ermittle Kunde :Operation In(:Kundennummer) Out(:Kunde)

Skript 114: Interaktion zur Ermittlung der Kundendaten

Die Schreibkraft verwendet eine Kundendatei, um die Kundendaten in Erfahrung zu bringen. Diese Kundendatei stellt aber keine Eingabe im Sinne der Operationsspezifikation dar. Die Verwendung von Objekten wird nur sichtbar,

- wenn die zur Realisierung der Operation ausgeführten Aktivitäten modelliert werden.
- wenn die Objekte in der Nachbedingung der Operation erwähnt werden. Dies sollte nach der schon erwähnten Rahmenregel immer dann der Fall sein, wenn die Objekte durch die Operation verändert wurden.
- wenn die Objekte Ein- oder Ausgaben der Operation darstellen.

Ein Problem entsteht dann, wenn keine der genannten Bedingungen zutrifft. In diesem Fall bleibt die Verwendung verborgen. Dies ergibt sich aus dem *Information Hiding* und stellt insofern keinen konzeptionellen Fehler dar, da die Spezifikation ja gerade nicht vorwegnehmen soll, wie die Operation durch die Objekte der Klasse realisiert wird. Im Rahmen der Modellierung von Prozessen will man aber unter Umständen schon bei der Spezifikation der Operationen externe Referenzen dokumentieren.

Dieses Problem soll dadurch gelöst werden, dass ein Hinweis auf die Verwendung externer Objekte und Klassen durch eine Operation angegeben wird. Dies geschieht durch die Angabe der externen Referenzen (Prädikat `Ref(Identifier List)`), auf das eine durch Kommata getrennte Liste von Bezeichnern folgt.[1] Die Bezeichner verweisen entweder allgemein auf eine Klasse oder auf ein bestimmtes Objekt, welches im Rahmen der Operation verwendet wird. Für die Operation `Brief erstellen` würde gelten:

```
Ref( :Kostenstelle, :Kostenträger)
```

[1] Dies wird in VDM durch die Angabe der *external variables clause* und dem Schlüsselwort EXT realisiert. Auf diese Weise werden dort alle Zugriffe auf externe Variablen deklariert. Ähnliche Konstrukte werden auch in FUSION im Operationenmodell verwendet. Vgl. [Dawes 1991: S. 134], [Sheppard 1994: S. 332], [Coleman 1994: S. 269]. Da hier nur externe Referenzen festgehalten werden sollen, wird nicht zwischen Lese- und Schreibzugriff unterschieden.

Es soll darauf hingewiesen werden, dass dieses Prädikat aus pragmatischen Erwägungen eingeführt wurde und aufgrund folgender Gründe nur restriktiv eingesetzt wird:

- Der Hinweis auf externe Referenzen in der Schnittstelle einer Operation verletzt das Prinzip des Information Hidings.
- Der Hinweis auf externe Referenzen hat keine Auswirkungen auf das Konzeptuelle Schema, sondern stellt eine Art „syntaktischen Zucker“[1] dar.
- Die Information ist redundant, wenn die Realisierung der Operation durch weitere Interaktionen dokumentiert wird.

6.6.9 Vertragsprinzip

Vor- und Nachbedingungen werden zur Spezifikation von Verträgen verwendet. Das Vertragsprinzip ist eines der wichtigsten Konzepte des objektorientierten Entwurfs.[2] Die Wurzeln des

[1] Dieser Begriff wurde von SAAKE übernommen, der ihn in seiner Habilitation verwendet hat. Vgl. [Saake 1993].

[2] Vgl. [Meyer 1992], [Meyer 1992c], [Henderson-Sellers 1998: Appendix E, Stichwort „Contracting“]. Hierzu ein Exkurs: Der in der objektorientierten Modellierung verwendete Vertragsbegriff entspricht nur sehr weitläufig der juristischen Terminologie und ist eher im Sinne einer Analogie zu verstehen. Die Prinzipien des Rechtsgeschäfts – wie in §§ 104 – 185 BGB geregelt – sind dabei nur bedingt übertragbar. Akzeptiert man diese Analogie, entspricht der objektorientierte Vertragsbegriff in etwa dem schuldrechtlichen Vertrag zur Begründung einer Leistungsverpflichtung, der durch Übereinstimmung mehrerer Beteiligter geschlossen wird (vgl. §§ 145 ff. BGB). Genauer gesagt, handelt es sich um zweiseitig verpflichtende Verträge, wie sie z. B. in §§ 598, 662 BGB und §§ 320 ff. BGB (gegenseitiger Vertrag) geregelt sind. Der Inhalt des Vertrags entspricht den Vertragspflichten der Parteien. Sie basieren auf einer Willenserklärung gem. §§ 116 – 144 BGB. Auf das Vorliegen von Willensmängeln, wie z. B. Formmängel, Einigungsmängel etc., wird dabei nicht eingegangen. Auch auf Wirksamkeitsgrenzen, wie z. B. gesetzliche Verbote (§ 134 BGB), wird nicht eingegangen. Die in § 275 (nachträgliche Unmöglichkeit), § 280 (Schadenersatz bei zu vertretender Unmöglichkeit), § 306 (unmögliche Leistung), § 323 (nicht zu vertretendes Unmöglichwerden), § 324 (vom Gläubiger zu vertretendes Unmöglichwerden), § 325 (vom Schuldner zu vertretendes Unmöglichwerden) genannten Leistungsstörungen, der Verzug, die Mangelhaftigkeit der Leistung und die sonstigen Störungen bleiben im objektorientierten Vertragsprinzip unberücksichtigt bzw. werden stark vereinfacht. Darüber hinaus können beim objektorientierten Vertragsprinzip Verträge zwischen allen Objekten geschlossen werden, wodurch auf die Beschränkung der Geschäfts- und Rechtsfähigkeit auf juristische und natürliche Personen keine Rücksicht genommen wird. Stattdessen wird von der Fiktion ausgegangen, Verträge könnten auch unter Beteiligung von Rechtsobjekten geschlossen werden. Vgl. [STUD-JUR 1994], [Rüßmann 1994]. Damit sei dieser Aspekt im

Vertragsprinzips finden sich u. a. im Bereich der formalen Spezifikationssprachen und der Theorie abstrakter Datentypen.[1] MEYER empfiehlt die Verwendung des Vertragsprinzip ausdrücklich auch in der Analyse.[2] Obwohl sich die Verwendung von Vor- und Nachbedingungen in einer ganzen Reihe von Modellierungsmethoden findet,[3] wurde die Idee, das Vertragsprinzip im Bereich der GPM zu verwenden, erstmals vom Verfasser erwähnt.[4]

Das Vertragsprinzip wird in einem objektorientierten System zur Verteilung von Pflichten auf der Basis von Vor- und Nachbedingungen verwendet. Es beruht auf der Überlegung, unter welchen Bedingungen der Participant eine Operation erfolgreich ausführt.[5] Die Nachbedingung der Operation dokumentiert, welches Ergebnis der Participant nach dem erfolgreichen Abschluss der Operation zusichert. Die Vorbedingung dokumentiert, welche Bedingungen erfüllt sein müssen, damit der Participant die Nachbedingung herbeiführen kann. Auf diese Weise dokumentieren Vor- und Nachbedingung einen *Vertrag*, an den Initiator und Participant gebunden sind. Die Pflichtverteilung besteht darin, dass es die Pflicht des Initiators einer Operation ist, die Vorbedingung der von ihm aufgerufenen Operation sicherzustellen. Nur dann kann er das in der Nachbedingung zugesicherte Ergebnis – die Pflicht des Participants – erwarten. Wenn er sicherstellen will, dass der Participant seine Aufgaben erfüllt, fordert er die Operation nur an, wenn die Vorbedingung erfüllt ist. Das Vertragsprinzip wird für alle Interaktionen unabhängig von der Art der beteiligten Objekte verwendet. Ein Vertrag dokumentiert durch Vor- und Nachbedingungen ein abstraktes Leistungsverhältnis zwischen Initiator und Participant.[6] Der Vertrag wird bei jeder Aufforderung, eine Operation auszuführen, geprüft. Der Anbieter einer Leistung verspricht, die durch eine Operation manifestierte Nachbedingung herbeizuführen, wenn der Abnehmer der Leistung verspricht, die in der Vorbedingung definierten Bedingungen zu erfüllen.[7] Hierzu ein einfaches Beispiel:

Rahmen der vorliegenden Arbeit ausreichend gewürdigt. Im weiteren Verlauf wird auf Parallelen und Unterschiede zum bestehenden Rechtssystem nicht weiter eingegangen.

[1] Vgl. [Meyer 1997b].

[2] Vgl. [Meyer 1999].

[3] Vgl. [Desfray 1994: S. 81], [Schienmann 1997: S. 234], [Waldén 1995: S. 41 ff.], [Warmer 1999: S. 2].

[4] Vgl. [Linssen 1996].

[5] Ähnlich wie in MOSES [Henderson-Sellers 1994: S. 55] und EIFFEL [Meyer 1997] wird hier der Vertrag auf der Ebene einzelner Operationen definiert. Anders ist dagegen die Verwendung in der Methode RDD [Wirfs-Brock 1993]. Dort regeln Verträge die Bedingungen einer kohäsiven Menge von Operationen.

[6] Eine ähnliche Sichtweise findet sich auch in [Züllighoven 1998: S. 45 f.].

[7] Im Original: *„If you promise to call* r *with* pre *satisfied then I, in return, promise to deliver a final state in which* post *is satisfied.“* [Meyer 1997: S. 341].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person	füllt Tank :Action	:Interaction	:Tank	füllen :Operation Pre(Tank ist nicht gefüllt) Post(Tank ist gefüllt)

Skript 115: Spezifikation einer Operation über Vor- und Nachbedingungen

Dieser Vertrag sichert einer Person zu, dass ein Tank gefüllt ist, wenn die Operation `füllen` abgeschlossen ist. Dies geschieht aber nur unter der Vorbedingung, dass der Tank nicht schon gefüllt ist. Die Person hat die Verpflichtung, sich davon zu überzeugen, dass der Tank nicht schon voll ist.

Im juristischen Sinn existiert hier sicherlich weder ein Vertrag zwischen der Person und dem Tank des Kraftfahrzeugs noch eine Leistungsbeziehung. Als abstraktes Denkschema ist die Verwendung der Begriffe aber sinnvoll und in der Informatik weit verbreitet. Nach MEYER lässt sich so präzise dokumentieren, unter welchen Bedingungen ein System erfolgreich seine Aufgaben erfüllen kann. Das folgende Beispiel wäre eine sinnvolle Beschreibung einer Dienstleistung in einem Geschäftsprozess:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Flug-gast	fliegt zum Zielort :Action	:Interaction Mode(Request Service	:Fluggesell-schaft	befördert Fluggast :Operation Pre(5 Minuten vor Ende der Eincheckzeit am Abflugort; Gepäck akzeptabel; Flug bezahlt) Post(Transportiert zum Zielort) Err(Flug überbucht: Maschine.ausgebucht = True)

Skript 116: Spezifikation einer Dienstleistung durch einen Vertrag

Die einzelnen Bedingungen müssen – so weit nicht etwas anderes angegeben wird – alle erfüllt sein, damit der Vertrag erfüllt wird. Wenn der Fluggast seine Verpflichtungen nicht erfüllt, ist die Fluggesellschaft von der Verpflichtung, ihn zu befördern, befreit. Alle Ausdrücke von Vor- und Nachbedingungen stehen jeweils in einer Konjunktionsbeziehung. In anderen Fällen müssen die entsprechenden logischen Operatoren verwendet werden.

Man mag einwenden, die erwähnten Vorbedingungen seien innerhalb des Skripts zu überprüfen und sollten deshalb als Interaktionen abgebildet werden. Man mag außerdem einwenden, die Vorbedingungen seien unnötig und könnten durch Selektoren abgebildet werden. Diese Position geht in doppelter Weise an der Argumentation von MEYER und der Idee der Vorbedingungen vorbei:

1. Auch eine Testoperation wird wieder von bestimmten Annahmen ausgehen, um operieren zu können. Man verlagert also nur das Problem, ohne es zu lösen.
2. Das Vertragsprinzip dient der Komplexitätsreduktion. Es wird verwendet, um die Modelle nicht mit einer Vielzahl von Überprüfungen und Testabläufen zu überladen. Wenn die Prüfung einer Bedingung explizit abgebildet werden soll, kann sie als eigenständige Operation dargestellt werden.

6.6.10 Verhalten bei Nichterfüllung

Der Vertrag besteht aus Rechten und Pflichten, die von Initiator und Participant einzuhalten sind, damit die Operation erfolgreich ausgeführt wird. Wenn man einen Vertrag spezifiziert, muss man die Frage stellen, was passiert, wenn Initiator oder Participant den Vertrag nicht einhalten. Notwendig ist ein Verfahren, wie im Rahmen der Modellierung mit Nichterfüllung umgegangen wird. Dies wird als Ausnahmebehandlung (*exception handling*) bezeichnet.[1] Durch eine differenzierte Behandlung eingetretener Ausnahmen lässt sich die Zuverlässigkeit von Systemen erhöhen, da auf diese Weise verhindert wird, dass das System in einen undefinierten Zustand gerät.

In vielen Modellierungsansätzen wird dieses Thema sehr knapp oder überhaupt nicht behandelt. In VDM wird zwar angegeben, unter welcher Bedingung ein Fehler eintritt und welchen Zustand das System beim Eintreten eines Fehlers besitzt. Welches Verhalten das System in dieser Situation zeigt, ist jedoch nicht Bestandteil von VDM. Für WALDÉN und COLEMAN ist die Behandlung von Ausnahmen eine Implementierungsentscheidung und wird nicht behandelt.[2] DE CHAMPEAUX empfiehlt, das Scheitern eines Vertrags an den Initiator der Operation zu melden und die aktuelle Operation abzubrechen.[3] Dass sich dadurch das System möglicherweise in einem nicht definierten Zustand befindet, wird nicht problematisiert. D'SOUZA et. al. stellen wenigstens mehrere Strategien vor, wie das Eintreten einer Fehlerbedingung angezeigt wird.[4] Fehlerbehandelnde Operationen werden dadurch allerdings nicht getriggert. MEYER hat dagegen ein Konzept zur Behandlung von Fehlern ausgearbeitet, welches sich in ähnlicher Form schon bei LISKOV und GUTTAG findet.[5] Dabei werden von MEYER zwei Situationen unterschieden:[6]

---

[1] Vgl. [Louden 1994: S. 287]. Ein früher, aber immer noch lesenswerter Überblick über die Prinzipien der Ausnahmebehandlung auf der Ebene von Programmiersprachen findet sich in [Goodenough 1975].

[2] Vgl. [Waldén 1995: S. 212], [Coleman 1994: S. 117 ff.].

[3] [de Champeaux 1993: S. 377].

[4] [D'Souza 1999: S. 331 – 337].

[5] Vgl. [Liskov 1986].

[6] Vgl. [Meyer 1997].

- Nichterfüllung durch den Initiator liegt dann vor, wenn er die Vorbedingung einer Operation nicht erfüllt.
- Nichterfüllung durch den Participant liegt vor, wenn er die Nachbedingung der Operation nicht erfüllt, obwohl die Vorbedingung erfüllt worden ist.

Es wird davon ausgegangen, dass bei Nichterfüllung eines Vertrags durch den Initiator der Participant keine Leistung erbringt.[1] Mit anderen Worten: Die Operation wird nur ausgeführt, wenn die Vorbedingung erfüllt ist. Im Fall der Nichterfüllung durch den Participant erfolgt die Behandlung differenzierter. Wird festgestellt, dass ein Vertrag nicht erfüllt werden kann, existieren zwei Alternativen. In beiden Fällen gilt die eigentliche Operation als gescheitert:

- Er wird versucht, durch ein abweichendes Vorgehen den Vertrag doch noch zu erfüllen. Dies wird bei MEYER als *Neuversuch* (*retry*) bezeichnet.[2]
- Dem Initiator wird das Scheitern der Operation mitgeteilt. Dies wird von MEYER als *Versagen* (*failure*) bezeichnet.[3]

In der objektorientierten Modellierung wird ein Fehler als ein Ereignis bzw. als ein Signal angesehen. Auf dieses Signal soll von Objekten durch die Ausführung von Operationen reagiert werden. Da das Eintreten des Fehlers eine Reaktion bewirken soll, muss es auf der Initiator-Seite des Skripts abgebildet werden. Hier existieren zwei Möglichkeiten:

- Der Fehler selbst kann ebenfalls als Objekt modelliert werden.[4] In diesem Fall erhält es das Prädikat `Stpe(Exception)`.[5] Die Initiator Responsibility kann frei gewählt werden. Der in der Fehlerbedingung der Operation verwendete Bezeichner wird in diesem Fall als Argument des Prädikats `when()` für die Aktion übernommen.[6]
- Es besteht aber auch die Möglichkeit, dass ein „normales“ Objekt eine Ausnahme durch eine Aktivität vom Typ *EXCEPTION* anzeigt. Der in der Fehlerbedingung der Operation verwendete Bezeichner wird auch hier als Argument des Prädikats `when()` für die Aktion übernommen.

---

[1] Vgl. [Meyer 1997: S. 347].

[2] Vgl. [Meyer 1997: S. 417].

[3] Vgl. [Meyer 1997: S. 417].

[4] Vgl. [Rumbaugh 1999: S. 269].

[5] Dies entspricht der in der UML verwendeten Sichtweise.

[6] Auf die Bedeutung dieses Prädikats wird in Abschnitt 6.8 eingegangen.

In beiden Fällen erhält die stattfindende Interaktion das Prädikat `Kind(Exception)`. Dadurch soll erreicht werden, dass der initiierte Ablauf ausdrücklich als Behandlung einer besonderen Situation – eben als Ausnahme – erkennbar ist. Das folgende Skript beinhaltet Muster der beiden genannten Möglichkeiten:[1]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Überbuchung Stpe(Exception)	eingetreten :Action when(Flug überbucht = True)	:Interaction Kind(Exception)	...	...
	:Flug	überbucht :Exception when(Flug überbucht = True)	:Interaction Kind(Exception)	...	...

Skript 117: Fehlerbehandlung auslösende Aktionen

Verwendet man das Vertragsmodell von MEYER, besteht der Unterschied darin, wer auf das Eintreten des Fehlers reagiert. Dabei werden drei Situationen unterschieden:

- Das Objekt, durch dessen Operation die Ausnahmebedingung erzeugt wurde, versucht durch eine andere Operation den Vertrag zu erfüllen. Um zu verhindern, dass dies nicht als verkappte Iteration missbraucht wird, darf es sich nicht um die gleiche Operation handeln.
- Ein anderes Objekt reagiert auf die Ausnahmebedingung durch Ausführung einer Operation.
- Der Initiator der gescheiterten Operation wird durch die Ausnahmebedingung informiert.

Diese unterschiedlichen Möglichkeiten sollen anhand von Beispielen demonstriert werden. Die erste Möglichkeit besteht darin, dass der Participant die Ausnahmebedingung selbst behandelt. Im Fall des überbuchten Fluges kann dies wie folgt aussehen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Flug	überbucht :Exception when(Flug überbucht=True)	:Interaction Kind(Exception)	:Flug- gesellschaft	Umbuchung auf Ersatzflug :Operation

Skript 118: Vom Participant behandelte Ausnahmesituation

Im weiteren Verlauf des Skripts wird die Fluggesellschaft versuchen, den Passagier auf einen Ersatzflug umzubuchen. Wenn dies gelingt, wird die ursprünglich vereinbarte Dienstleistung erfüllt: Vereinbart war, den Passagier zum Zielort zu transportieren. Nicht vereinbart war, mit

[1] Das Prädikat `when()` wird in Abschnitt 6.8 eingeführt.

welcher Fluggesellschaft. Dies ist die Version des Neuversuchs, in der das Objekt der gescheiterten Operation die Ausnahmebehandlung selbst durchführt.

Eine weitere Möglichkeit besteht darin, dass ein anderes Objekt auf die Ausnahmebedingung reagiert, wie dies im folgenden Beispiel der Fall ist.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Zentrale	übermittelt Kundenanfrage :Action	:Interaction	:Hotline	entgegennehmen :Operation Pre(True) Post(Vorgang entgegengenommen) Err(Vorgang nicht angenommen: Vorgang.Status = unbearbeitet)
	:Annahme-verzug Stpe(Signal)	eingetreten :Action when(Vorgang nicht angenommen = True)	:Interaction Kind(Exception)	:Service-Management	entgegennehmen :Operation Pre(True) Post(Vorgang entgegengenommen)

Skript 119: Von einem anderen Objekt behandelte Ausnahmesituation

Wenn eine Hotlineabteilung einen übermittelten Vorgang nicht annimmt, wird dies als Fehler `Annahmeverzug` angesehen. In diesem Fall wird der Vorgang an das Servicemanagement übermittelt. Diese führt eine Operation mit dem gleichen Namen aus. Diese Kette kann nun durch Definition weiterer Fehlerbedingungen und Fehlerbehandlungen fortgesetzt werden.[1] Dies ist die zweite Version des Neuversuchs, in der ein anderes Objekt die Ausnahmebehandlung durchführt.

Was ist das beabsichtigte Ziel eines Neuversuchs? Nach MEYER muss durch einen erfolgreichen Neuversuch die ursprünglich vereinbarte Leistung erbracht werden. Wenn eine Klasse K_2 durch die Ausführung einer Operation y eine eingetretene Ausnahmebedingung behebt, muss sie die ursprüngliche Nachbedingung der Operation x einer Klasse K_1 herbeiführen.

Die dritte Möglichkeit stellt schließlich das *Versagen* des Participants und das Nichteinhalten des Vertrags dar. In diesem Fall wird der Initiator informiert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Flug-gast	fliegt zum Zielort :Action	:Interaction Mode(Request Service)	:Fluggesell-schaft	befördert Fluggast :Operation Pre(5 Minuten vor Ende der Eincheckzeit am Abflugort; Gepäck akzeptabel; Flug bezahlt) Post(Transportiert zum Zielort) Err(Flug überbucht: Maschine.ausgebucht=True)
:Flug	überbucht	:Interaction	:Fluggast	wird über Nicht-Beförderung informiert

[1] Vgl. die Darstellung von MEYER in [Meyer 1997: S. 420].

	:Exception	Kind(Exception)		:Operation
:Flug-gast	verlangt Flugpreis und Schadenersatz :Action	:Interaction Answer(:Geld-betrag)	:Fluggesell-schaft	zahlt Flugpreis und Schadenersatz Pre(Kann Fluggast nicht befördern, obwohl dazu verpflichtet) Post(Hat gezahlt)

Skript 120: Endgültiges Scheitern einer Dienstleistung

Ist kein Neuversuch definiert oder scheitert auch der Neuversuch, ist der Fall des *Versagens* eingetreten. Der Initiator der gescheiterten Operation wird informiert und reagiert nun. Aus der Ausnahmebehandlung durch den Participant oder eines anderen Objekts ist eine Ausnahmebehandlung durch den Initiator geworden.

In den ersten beiden Fällen ist dem Initiator das Eintreten der Ausnahmebedingung verborgen geblieben. Nur im letzten Fall wird er darüber informiert, dass die von ihm angeforderte Dienstleistung nicht erbracht wurde. Erhält ein Initiator-Objekt keine Nachricht in der besonderen Form der Ausnahmebenachrichtigung (`Kind(Exception)`), kann er davon ausgehen, dass die von ihm angeforderte Dienstleistung erbracht wurde.

6.6.11 Zeitliche Restriktionen

Nach BOOCH können in der UML den Operationen zeitliche Einschränkungen (*timing constraint*) zugeordnet werden, mit denen sich der Zeitverbrauch ihrer Ausführung beschreiben lässt.[1] Hierfür werden im Prädikat `TimingConstraint` wieder die Prädikate für den Startzeit (`startTime()`), Abschlusszeit (`stopTime()`) und die Ausführungszeit (`executionTime()`) verwendet, die schon in Abschnitt 6.2.14 für Interaktionsbeziehungen eingeführt worden sind.[2] Ist beispielsweise die Ausführungszeit einer Operation kürzer als 2 Sekunden, erhält sie die Einschränkung `TimingConstraint(executionTime()< 2 s)`. Das folgende Beispiel zeigt die Verwendung in einem Skript:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Lager	Erstellt Stückliste :Action	Auflösen der Stückliste :Interaction	:Material-wirtschafts-system	löst Stückliste auf :Operation TimingConstraint(executionTime() < 2 s)

Skript 121: Zeitverbrauch von Operationen

[1] Vgl. [Booch 1999: S. 467].

[2] Vgl. [Booch 1999: S. 324]. Aus den zur Verfügung stehenden Veröffentlichungen geht nicht eindeutig hervor, ob es sich um Funktionen oder Schlüsselwörter handelt.

Da bei gepufferten Operationen der Zeitpunkt, in dem die Nachricht beim Participant eintrifft, und der Zeitpunkt, in dem die Operation getriggert wird, voneinander abweichen können, entspricht der Eintreffenszeitpunkt der Interaktion nicht unbedingt der Startzeit der Operation. Die Verzögerung, mit der ein Objekt der Aufforderung entspricht, eine Operation auszuführen, ist die Differenz zwischen `Interaction.receiveTime()` und `Operation.startTime()`. Im Falle einer zeitabhängigen Interaktion mit der Frist 0 muss `Interaction.receiveTime()` und `Operation.startTime()` identisch sein.[1]

6.6.12 Kosten der Ausführung einer Aktivität

Neben den Ausführungszeiten von Aktivitäten ist die Betrachtung der verursachten Kosten ein weiteres wichtiges Kriterium zur Analyse. Trotzdem findet sich in keinem objektorientierten Modellierungsansatz eine Möglichkeit, die durch die Ausführung von Aktivitäten verursachten Kosten zu modellieren.

Eine solche Modellierung kann nun entweder auf der Modellebene durch entsprechende Klassen durchgeführt werden;[2] oder sie findet auf der Metamodellebene durch entsprechende Prädikate statt. Der hier entwickelte Ansatz dokumentiert die Kosten der Ausführung einer Aktivität durch ein entsprechendes Prädikat und basiert auf dem Ansatz der Prozesskostenrechnung.[3] Grundgedanke ist, dass die Aktivitäten unterschiedlicher Ressourcen Kosten verursachen.[4] Die Kosten der Ausführung einer Aktivität werden dabei durch das Prädikat `Cost(`*`Cost Expression`*`)` abgebildet.[5] Der Kostenausdruck setzt sich zusammen aus Aktivitätentreiber und Aktivitätenkostensätzen. Unter einem *Aktivitätentreiber* wird die Einflussgröße verstanden, mit deren Hilfe die

---

[1] Eine mögliche Erweiterung dieses Konzepts wäre die Formulierung von zeitlichen Restriktionen mit Hilfe von Wahrscheinlichkeiten, wie dies in [Eversheim 1995: S. 49 ff.] zu finden ist. Das dort zu findende Beispiel einer Aktivität, die in 30 % aller Fälle 0,1 Tage zur Ausführung benötigt und in 70 % aller Fälle weitere 10,2 Tage für Rückfragen und Liegezeiten, ließe sich wie folgt übertragen: `TimingConstraint(executionTime() = (0,3 p * 0,1 d + 0,7 p * (10,2 d + 0,1 d))`, wobei p die Wahrscheinlichkeit und d die Tage sind. Mit Hilfe von `executionTime()` ließe sich auch – unter Verwendung der sogenannten O-Notation (vgl. [Rechenberg 1999: S. 120 f.]) – das Wachstumsverhalten der Ausführungszeit einer Operation beschreiben. Dies ist späteren Erweiterungen vorbehalten.

[2] Vgl. Abschnitt 6.6.5.

[3] Vgl. [HeilmannM 1997], [Raster 1994b], [Heinen 1991c], [Glaser 1992], [Coenenberg 1991], [Scheer 1998b: S. 66 ff.].

[4] Vgl. [HeilmannM 1997: S. 26].

[5] Dieses Prädikat ist sowohl für alle Aktivitäten, also Aktionen, Ausnahmen und Operationen, definiert.

Inanspruchnahme von Objekten gemessen werden kann.[1] Aktivitätentreiber können Mengeneinheiten, Verarbeitungszeiten, Mitarbeiterstundensätze, Maschinenstundensätze etc. sein. Der *Aktivitätskostensatz* ist der Kostensatz, der bei der Ausführung der Aktivität für einen Aktivitätentreiber anfällt.[2] Für das Erstellen eines Briefes könnte beispielsweise festgelegt werden, dass der Aktivitätentreiber die Anzahl der Seiten des Briefs ist. Wenn man davon ausgeht, dass der Aktivitätskostensatz 20,00 Geldeinheiten (GE) beträgt, ergibt sich:[3]

```
Brief :Class
erstellen :Operation
Cost(Seitenzahl * 20,00 GE Seitenerstellungssatz)
```

Auch bei der Dokumentation der Kosten werden die Prädikate Startzeit (`startTime()`), Abschlusszeit (`stopTime()`) und Ausführungszeit (`executionTime()`) verwendet.

```
Schreibkraft :Class
Diktat aufnehmen :Operation
Cost(executionTime() * 30,00 GE Mitarbeiterstundensatz)
```

Die Kosten der Ausführung von Aktivitäten können sich auf alle Klassen beziehen. Dies können Geschäftsobjekte, Organisationseinheiten oder Sachmittel sein.[4]

```
Mitarbeiter :Class
Wareneingang vornehmen :Operation
Cost(Anzahl * 4,75 GE)


Angebot :Class
erstellen :Operation
```

---

[1] Dies wird von HEILMANN als *Ressourcentreiber* bezeichnet. Vgl. [HeilmannM 1997: S. 31].

[2] Dies wird allgemein als *Prozesskostensatz* bezeichnet. Vgl. [Raster 1994], [Coenenberg 1991], [Glaser 1992], [Schulte-Zurhausen 1999: S. 74]. Auf die Berechnung dieses Kostensatzes wird hier nicht weiter eingegangen. Üblicherweise wird er als Quotient der Prozesskosten und der Prozessmenge berechnet. Beispiele hierzu finden sich in [Coenenberg 1991] und [Raster 1994]. GLASER demonstriert die Berücksichtigung von Varianten bei der Prozessausführung. Vgl. [Glaser 1992]. Auf das Problem, in welcher Form Gemeinkosten bei einer solchen aktivitätsorientierten Betrachtung berücksichtigt werden, wird hier ebenfalls nicht eingegangen. Sie können durch prozentuale Zuschlagsätze berücksichtigt werden, wie dies von COENENBERG und GLASER vorgeschlagen wird. Mit der Analyse von Kostentreibern befasst sich HEILMANN in [HeilmannM 1997: S. 186 ff.].

[3] Aus Gründen der Anschaulichkeit wird hier davon ausgegangen, dass für eine Aktivität nur ein Aktivitätentreiber zu berücksichtigen ist.

[4] HEILMANN verwendet hierfür den Begriff *Kostenobjekt*.

```
Cost(Anzahl Angebotsposition * 12,00 GE)
```

Wenn sich die Kosten auf mehrere Objekte beziehen, wird dies durch mehrere Operationen abgebildet:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	/Kunde :Person	wünscht Angebot :Action	:Interaction	:SB	Angebot erstellen :Operation Cost(executionTime() * 30,00 GE Mitarbeiterstundensatz)
2.	:SB	erstellt Angebot :Action	:Interaction	:Angebot	erstellen :Operation Cost(Angebotsposition * 12,00 GE)
3.	:SB	verwendet PC :Action	:Interaction	:Arbeitsplatzrechner	Angebot erstellen :Operation Cost(executionTime() * 10,00 GE Maschinenstundensatz)

Skript 122: Modellierung der Kosten, die durch Aktivitäten entstehen

Auf diese Weise wird abgebildet, dass die im Rahmen eines Geschäftsprozesses verwendeten Ressourcen nicht isoliert zu betrachten sind, sondern im Verbund Aktivitäten ausführen. Auch die Abbildung von Kosten in Abhängigkeit der zeitlichen Inanspruchnahme ist möglich, wie die erste und letzte Zeile im Skript 122 zeigt. Die Ermittlung von Gesamtkosten oder die Prüfung der Konsistenz der Kostenausdrücke kann durch die gegebenen Freiheitsgrade in der Modellierung komplex werden und sollte ggf. durch ein Werkzeug unterstützt werden, da sie nicht durch die Notation erfolgt.

6.6.13 Rollenwechsel

Ein Rollenwechsel liegt vor, wenn beispielsweise ein Objekt einer Klasse `Person` zuerst als `Bewerber` und später als `Mitarbeiter` betrachtet wird. Im Gegensatz zur Zugehörigkeit zu einer Klasse kann die Zugehörigkeit zu einer Rolle also wechseln. Rollen stellen somit dynamische Typen dar. Die Veränderung von Rollen wird in der UML durch Interaktionen abgebildet.[1] Im folgenden Skript erhält eine `Person` die Rolle eines `Mitarbeiters`, wenn sie eingestellt wird. Durch den Objektbezeichner `Müller` wird deutlich, dass es sich in der Interaktion und dem Zustandsübergang um das gleiche Objekt handelt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Müller /Bewerber :Person	bewirbt sich :Action	:Interaction	:Personalchef	führt Bewerbungsgespräch :Operation
	:Personalchef	stellt Bewerber	:Interaction	Müller /Bewerber	wird eingestellt :Operation

[1] Vgl. [Booch 1999: S. 139, 166, 190, 253 f.].

		ein :Action		:Person	
	Müller /Bewerber :Person	nicht angestellt :State	:Statechange Act(wird eingestellt :Operation)	Müller /Mitarbeiter :Person	angestellt :State

Skript 123: Rollenwechsel

6.7 Invarianten

Das Vertragsmodell steht im engen Zusammenhang mit Invarianten und Ausnahmen. Invarianten sind Randbedingungen, die – unabhängig von der Ausführung einer Operation – immer gelten müssen. Eine Klasse besitzt keine, eine oder mehrere Invarianten. Jede Invariante besteht aus einem – innerhalb der Klasse – eindeutigen Namen und einem Ausdruck.[1] Der Ausdruck kann die Form natürlicher Sprache besitzen. Für eine Klasse Mitarbeiter könnte beispielsweise die Invariante `Altersgrenze` definiert sein, die besagt, dass das Alter des Mitarbeiters geringer als 65 Jahre sein muss. Dies würde dem OCL Ausdruck `Alter < 65` entsprechen.[2]

Der Zusammenhang zwischen Invarianten und Vor- und Nachbedingungen ist wie folgt definiert: Vor der Ausführung der Operation sind die Vorbedingung der Operation und die Invariante der Klasse erfüllt. Nach der Ausführung der Operation sind die Nachbedingung der Operation und die Invariante erfüllt. Zur Abbildung der Invarianten von Klassen werden in den Skripten interne Beziehungen verwendet. Die Dokumentation einer Invariante erfolgt durch das Prädikat `Expression(Expression)`. Das folgende Skript drückt aus, dass bei der Übernahme eines Auftrags durch einen Sachbearbeiter die Invariante tangiert wird, dass für alle Exemplare der Klasse Sachbearbeiter (SB) das Attribut `Anzahl Arbeitsaufträge` immer kleiner als 5 sein muss.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	:SB	übernimmt Auftrag :Operation
	:SB	übernimmt Auftrag :Action	:Internal	:SB	Maximale Arbeitsaufträge :Invariant Expression(Anzahl Arbeitsaufträge < 5)

Skript 124: Verwendung interner Beziehungen für Invarianten

[1] Die Forderung, dass eine Invariante einen Namen besitzen muss, wird durch den Abbildungsprozess auf das Konzeptuelle Schema erzwungen. In der UML besitzen Invarianten keinen Namen. In EIFFEL besitzt jede Zusicherung einen Namen.

[2] Siehe [OMG 1999], [Warmer 1999]. Weitere Beispiele für Invarianten finden sich im Metamodell des Konzeptuellen Schemas in Kapitel 4.

6.8 Aktionen und Ereignisse

Zeilen, die eine Interaktion darstellen, beinhalten in der Tabellenspalte Initiator Responsibility den aktiven Teil der Interaktion, der zum Auslöser der Operation wird. Dies wird als Aktion (*Action*) bezeichnet. Eine Aktion wird in der Form `Aktionsname :Action` festgehalten, wobei die Angabe eines Namens zwar optional ist, aber aus Gründen der Verständlichkeit angegeben werden sollte.[1] Durch Aktionen werden die Ursachen der Ereignisse spezifiziert, auf die reagiert werden muss.[2] Als Ereignis wird dabei das Eintreten eines zu beachtenden Umstandes betrachtet. Eine Ausnahme wird als Spezialfall eines zu beachtenden Umstandes angesehen. Soll ein Umstand als Ausnahme gekennzeichnet werden, erhält er im Skript den Typ Exception, verwendet in der Schreibweise `Ausnahmename :Exception`. Es gilt der folgende Zusammenhang: Die Operation ist die Reaktion auf ein eingetretenes Ereignis, dessen Ursache die Aktion ist.[3]

In der Arbeit von SAAKE wird das Prinzip gemeinsamer Ereignisse als Primitiv der Kommunikation zwischen Objekten verwendet.[4] Objekte kommunizieren, wenn Ereignisnamen als identisch deklariert sind.[5] Da diese Konstruktion davon ausgeht, dass Ereignisse zeitgleich für den Initiator und den Participant eintreten, ist sie auf den hier verwendeten Ansatz nicht übertragbar. Statt dessen wird in Interaktionsbeziehungen explizit die Aktion des Initiators mit der Operation des Participant über eine Interaktionsbeziehung verbunden. Sieht man Aktionen als Ursache für Ereignisses an, kann man die Ereignistypen der UML[6] als Ausgangspunkt für die Differenzierung möglicher Formen von Aktionen verwenden. Dabei sind allerdings folgende Aspekte zu beachten,

---

[1] Aus diesem Grund werden hier auch nur Aktionen mit einem Namen formuliert. Man sollte auf die Angabe des Namens nur dann verzichten, wenn nicht festgestellt werden kann, durch welche Aktion beim Participant eine Operation ausgelöst wird. Die Möglichkeit, keinen Aktionsnamen anzugeben, wird nur aus dem Grund ermöglicht, um Diagramme anderer Modellierungsansätze (z. B. Sequenzdiagramme der UML) in Skripte übersetzen zu können, obwohl in ihnen keine benannten Aktionen verwendet werden.

[2] Nach Kenntnis des Verfassers ist die Methode ROOM die einzige objektorientierte Methode, die einen ähnlichen Ansatz verfolgt. Auch dort wird immer ein Ereignis erzeugt, wenn eine Nachricht gesendet wird – und nicht wie in den meisten anderen Methoden, wenn die Nachricht beim Empfänger eintrifft. Vgl. [Selic 1994: S. 218].

[3] Man beachte, dass die Aktion nicht das Ereignis, sondern seine Ursache ist. Der Vollständigkeit halber sei erwähnt, dass die verbindende Interaktion und die Ausführung der Operation in vielen Ansätzen ebenfalls als Ereignis betrachtet wird. Die hier vorgenommene Abgrenzung sorgt für eine größere begriffliche Klarheit.

[4] Vgl. [Saake 1993]. Ähnlich auch in [Gillibrand 2000: S. 118].

[5] a. a. O. S. 89.

[6] Vgl. [Rumbaugh 1999: S. 267 ff.].

die die hier entwickelte Konzeption von der UML und anderen Modellierungsansätzen unterscheidet:

- In anderen objektorientierten Modellierungsansätzen ist die Verwendung von Ereignissen auf Zustandsdiagramme und damit auf die Dynamik einer Klasse beschränkt. Die Interaktion zwischen Objekten durch Nachrichten wird nicht wirklich als Ereignis aufgefasst, sondern eher als ein Prozeduraufruf. Damit nehmen die Ereignisse eine Randposition ein, was daran zu erkennen ist, dass in einigen objektorientierten Modellierungsansätzen auf die Ereignisse, auf die ein System reagieren können muss, überhaupt nicht Bezug genommen wird.[1] Im hier entwickelten Modellierungsansatz werden auch die Ereignisse, die zur Auslösung einer Operation führen, explizit modelliert. Eine Operation stellt somit immer die Reaktion auf ein Ereignis dar.
- Im Gegensatz zur UML sind Ereignisse immer Objekten bzw. Klassen zugeordnet.[2] Dabei spielt es keine Rolle, ob das Ereignis außerhalb oder innerhalb des Systems eintritt. Bei der Modellierung von offenen Systemen ist darauf zu achten, dass sie auf alle eintretenden Ereignisse reagieren können, die in der Umwelt eintreten und die als relevant in dem Sinne angesehen werden, dass das betrachtete System darauf reagieren können muss.[3] Auch die Umwelt eines Systems besteht aus Objekten, die sich identifizieren lassen und denen man die Ereignisse zuordnet, die sie erzeugen.[4] Wenn es von Bedeutung ist, ob das Ereignis innerhalb oder außerhalb des betrachteten Systems eingetreten ist, wird dies durch das Prädikat `System()` abgebildet.[5] Durch diese Verbindung ist man in der Lage, präzise festzuhalten, wann und von wem Ereignisse ausgelöst werden.

---

[1] Vgl. [Coad 1994]. Dieses Problem wurde 1992 auf einer Podiumsdiskussion in Kanada erörtert. Vgl. [Constantine 1992]. Eine der wenigen Methoden, in denen Ereignisse verwendet werden, stellt die Methode SYNTROPY [Cook 1994] dar. Die Ursache für die Ablehnung von Ereignissen ist schwierig festzustellen und hat möglicherweise wissenschaftssoziologische Ursachen. Es mag darin liegen, dass die ereignisorientierte Betrachtung von Systemen ursprünglich ein Element der strukturierten Methoden darstellte. Hier ist insbesondere die essenzielle Systemanalyse zu nennen. Vgl. [McMenamin 1988]. In objektorientierten Modellierungsansätzen hat man lange vermieden, deren Ergebnisse zu übernehmen, so sinnvoll dies möglicherweise im Einzelfall gewesen wäre. Die Verfechter des objektorientierten Ansatzes wollten sich konsequent vom herrschenden Denkansatz abgrenzen, um ihr neues Paradigma etablieren zu können. Zur Problematik des Paradigmenwechsels in der Informatik siehe [Quibeldey-Cirkel 1994].

[2] Ähnlich auch zu finden in [Chonoles 1999].

[3] Vgl. [McMenamin 1988: S. 47 ff., 175 ff.].

[4] Dies wird in der UML zumindest in der Form berücksichtigt, dass die Definition unterschiedlicher Akteure [Rumbaugh 1999: S. 144] möglich ist.

[5] Vgl. Abschnitt 6.1.3.

- In der UML lösen Ereignisse in Zustandsdiagrammen direkt Zustandsübergänge aus. Hier lösen Aktionen Operationen aus, die den Zustandsübergang herbeiführen. Eine Zustandsveränderung kann nur durch eine Operation durchgeführt werden. Dieser Zusammenhang wurde in Abschnitt 6.3 (Zustandsübergang) dargestellt.
- Verschiedene Objekte können auf das gleiche Ereignis in unterschiedlicher Art und Weise reagieren.
- Die Modellierung von Ausnahmen entspricht in der Syntax des Skriptmodells einem Ereignis. Dadurch wird berücksichtigt, dass in einem System Ausnahmen „die Regel sind".
- Das Eintreten eines Ereignisses wird nicht mit dem Triggern einer Operation gleichgesetzt. Zwischen beiden kann Zeit verstreichen, entweder weil die Interaktion Zeit benötigt oder weil der Participant noch nicht in der Lage ist, zu reagieren. Es ist auch möglich, dass ein Ereignis eintritt, kein Objekt reagiert und das Ereignis ergebnislos „verfällt".

Insbesondere der letzte Punkt stellt einen wesentlichen Unterschied zu Modellierungsansätzen dar, in denen das Senden einer Nachricht mit dem Empfang der Nachricht gleichgesetzt wird, wie dies beispielsweise in ROOM oder OORAM der Fall ist.[1] Durch die hier vorgenommene Differenzierung in Aktion, Interaktion und Operation kann detailliert die Aktivität des Initiators, die stattfindende Interaktion und die resultierende Aktivität des Participants modelliert werden. Während die Aktion die Spezifikation der Ursache von Ereignissen beinhaltet, werden durch das Prädikat `Content()` der Interaktion die Daten des aktuellen Ereignisvorfalls[2] abgebildet. Aus diesem Grund besitzen Aktionen weder ein Prädikat `In()` noch ein Prädikat `Out()`.[3]

6.8.1 Differenzierung von Ereignisarten

Ein Ereignis ist ein abstraktes Modell für eine diskrete Veränderung oder eine zeitlose Aktivität. Es wird zur Modellierung von Veränderungen oder als Ergebnis von Aktionen verwendet. Ereignisse können einem Zeitpunkt zugeordnet werden. Sie besitzen aber entweder keine Zeitdauer[4], oder ihre

[1] Vgl. [Selic 1994: S. 67], [Reenskaug 1996: S. 50, 62]. In der Methode ROOM wird dieser Unterschied wenigstens erwähnt. Vgl. [Selic 1994: S. 218].

[2] Vgl. [Martin 1999: S. 184].

[3] Versuche, sowohl auf der Initiator- wie auf der Participant-Seite den In- und Output festzulegen, führten zu hochgradig redundanten und häufig widersprüchlichen Modellen.

[4] Vgl. [Cook 1994: S. 30], [Brockhaus 2000].

zeitliche Ausdehnung wird als unerheblich angesehen.[1] Sie können plötzlich und unerwartet eintreten. Ein Ereignis kann bei unterschiedlichen Objekten zu unterschiedlichen Auswirkungen führen. In der UML werden Ereignisse sehr allgemein als *bemerkenswerte Begebenheit* (*noteworthy occurrence*) definiert.[2] Diese Aussage erhält mehr Inhalt, wenn man die Ereignisformen betrachtet, die im Metamodell der UML unterschieden werden:[3]

- Ein *Rufereignis* (*CallEvent*) liegt vor, wenn eine Operation aufgerufen wird.
- *Signale* bzw. *Signalereignisse* (*SignalEvent*) sind ein Sonderfall in der UML, in dem ein Ereignis als eigenständiges Objekt einer speziellen Ereignisklasse abgebildet wird.[4] Solche Signale werden häufig dazu verwendet, Störereignisse im Sinne von Fehlersituationen abzubilden, bei denen ein System seinen normalen Betriebsmodus verlässt.[5]
- Ein *Veränderungsereignis* (*ChangeEvent*) liegt vor, wenn festgestellt wird, dass ein bestimmter Ausdruck bzw. eine bestimmte Bedingung wahr wird.[6] Auch das *Eintreten eines Zeitpunkts* stellt ein Veränderungsereignis dar. Dieser Zeitpunkt kann in Form eines relativen oder absoluten Zeitpunktes angegeben werden.
- Ein *Zeitereignis* (*TimeEvent*) liegt vor, wenn festgestellt wird, dass ein bestimmter Zeitraum verstrichen ist. Dieser Zeitraum kann absolut oder relativ spezifiziert werden.[7]

Diese vier Ereignistypen zeigen den Unterschied zwischen objektorientierten und strukturierten Ansätzen, denn Rufereignisse und Signalereignisse wären in der strukturierten Analyse nur dann Ereignisse, wenn sie außerhalb des Systems eintreten würden. Im anderen Fall würden sie als Teile

---

[1] Das ist mit die Ursache, warum Aktionen nur die Ursache von Ereignissen, nicht aber die Ereignisse selbst darstellen. Aktionen können eine zeitliche Länge besitzen, Ereignisse besitzen (praktisch) keine zeitliche Ausdehnung.

[2] Vgl. [OMG 1999: S. 3-126]. Ähnlich auch in anderen Ansätzen. Vgl. exemplarisch [Martin 1993].

[3] Vgl. [OMG 1999: S. 2-125 ff., 3-126 f.], [Rumbaugh 1999: S. 189, 427, 474, 181].

[4] Vgl. [Booch 1999: S. 297], [Rumbaugh 1999: S. 427]. Der Normalfall ist, Ereignisse als zeitbezogenes Geschehen anzusehen, das im Zusammenhang mit Objekten eintritt.

[5] Bei einem Informationssystem ist dies z. B. das Scheitern einer Datenbanktransaktion. Bei einem Unternehmen könnte man den Zustand der kurzfristigen Zahlungsunfähigkeit als Verlust des normalen Betriebsmodus betrachten.

[6] Dies entspricht der Definition der DIN 69900, die ein Ereignis als Zeitpunkt definiert, in der ein definierter Systemzustand eintritt. [Kruse 1992: S. 14]. Auch MARTIN definiert Ereignis als beachtenswerte Zustandsänderung, was insofern nicht exakt zum objektorientierten Ansatz passt, in dem das Ereignis die Ursache der Zustandsveränderung ist. Vgl. [Martin 1999: S. 175].

[7] Zeitpunkt- und Zeitereignisse entsprechen den von MARTIN verwendeten Uhr-Ereignissen. Vgl. [Martin 1993: S. 117].

von Prozessen betrachtet werden.[1] In der hier verwendeten Sichtweise entstehen diese Ereignisarten als Ergebnis von Aktionen. Zu jedem Ereignistyp muss deshalb ein entsprechender Aktionstyp existieren. Daraus ergibt sich für Aktionen:

1. Durch Aktionen werden Ereignisse erzeugt, durch die Operationen aufgerufen werden.[2]
2. Durch Aktionen werden Signale erzeugt und das Eintreffen von Signalen angezeigt.
3. Durch Aktionen werden Bedingungen abgebildet. Diese Bedingung wird implizit ständig geprüft.[3] Wenn die Bedingung wahr wird, wird ein Ereignis erzeugt.
4. Durch Aktionen werden Zeitpunkte und Zeitfristen abgebildet. Wenn dieser Zeitpunkt erreicht oder die Frist abgelaufen ist, wird ein Ereignis erzeugt.[4]

Die Interaktion stellt in diesem Zusammenhang ein abstraktes Transportmedium dar,[5] durch welches das erzeugte Ereignis vom Initiator zum Participant befördert wird. Erst wenn es dort angekommen ist, wird es zum Ereignis des Participants, durch das er aufgefordert wird, eine Operation auszuführen. Das folgende Skript beinhaltet Beispiele für die unterschiedlichen Formen:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB Stpe(Position)	Umsatz berechnen :Action	Umsatzberechnung :Interaction	:Taschenrechner	Ergebnis ermitteln :Operation
:Maschinenausfall Stpe(Signal)	eintreten :Action	Defekt :Interaction	:Maschinenführer Stpe(Position)	Ausfall bemerken :Operation
:Kunde Stpe(Business Object)	Kontostand zu gering :Action	Veränderung :Interaction	:SB Stpe(Position)	informiert werden :Operation
:Student Stpe(Business Object)	Termin überschritten :Action	Zeitfrist :Interaction	:Prüfungsamt Stpe(Org-Unit)	Prüfer informieren :Operation

Skript 125: Beispiele für unterschiedliche Arten von Aktionen

[1] [McMenamin 1988: S. 63 ff.].

[2] Dies ist die Definition, die in der OBA nach RUBIN und GOLDBERG verwendet wird. Vgl. [Rubin 1992]. Man beachte, dass in nicht-objektorientierten Methoden wie ARIS Ereignisse danach unterschieden werden, ob sie eine Funktion oder einen Prozess auslösen. Darüber hinaus wird unterschieden, ob das Ereignis z. B. systemintern oder systemextern stattfindet. Zur Klassifikation von Ereignissen in ARIS siehe [Hoffmann 1992].

[3] Vgl. [Rumbaugh 1999: S. 268].

[4] Vgl. [Hoheisel 1999: S. 113].

[5] Technisch würde man von einem Kanal sprechen.

In der Interaktion Umsatzberechnung wird durch die Aktion Umsatz berechnen ein Rufereignis erzeugt, auf das der Participant Taschenrechner mit der Operation Ergebnis ermitteln reagiert. Da im Rahmen von Operationen Aktionen ausgeführt werden, ergibt sich, dass die Ausführung einer Operation die Ursache von weiteren Ereignissen sein kann.[1] Diesen Umstand macht man sich in der UML in den so genannten Aktivitätendiagrammen zu Nutze, in denen eine Aktivität implizit das auslösende Ereignis für die nächste Aktivität bzw. für die nächste Operation ist.[2] Ein hier behobener Mangel der UML ist, dass diese Ereignisse dort nicht sichtbar werden. Hier dagegen stellt die in jeder Interaktion vorhandene Aktion die Ursache für ein Ereignis dar. Nur die Aktionen in internen Beziehungen werden nicht als Ereignis betrachtet.[3]

In der Interaktion Defekt in Skript 125 tritt das als Objekt abgebildete Signal Maschinenausfall ein. Durch das Eintreten des Signals (Aktion eintreten) wird der Maschinenführer darüber informiert, dass die Maschine ausgefallen ist. Beliebige andere Objekte könnten ebenfalls auf dieses Signal reagieren, was in zusätzlichen Zeilen dokumentiert würde.[4]

In der Interaktion Veränderung bewirkt die Aktion Kontostand zu gering ein Veränderungsereignis. In dem Moment, in dem die Situation eintritt, dass der Kontostand des Kunden zu gering ist, wird das Ereignis erzeugt. Im Rahmen der dadurch initiierten Interaktion wird der Sachbearbeiter informiert. Das bedeutet, dass dies durch die Aktion dokumentierte Bedingung (konzeptionell) ständig geprüft werden muss.[5] Das Ereignis tritt ein, wenn der Wert dieser Bedingung wahr wird. Auf diese Weise kann auch eine Ausnahmebehandlung initiiert werden, die nicht explizit durch die Fehlerbedingung einer Operation ausgelöst wird. In diesem Fall erhält die Interaktion zusätzlich das Prädikat Kind(Exception).

Die Interaktion Zeitfrist beinhaltet das Eintreten eines Zeitereignisses. Wenn ein Student einen (hier nicht weiter spezifizierten) Termin überschritten hat, wird ein Ereignis ausgelöst, in dessen Rahmen das Objekt Prüfungsamt den oder die Prüfer informiert.

---

[1] Vgl. [Martin 1993: S. 112 f.].

[2] Vgl. [Booch 1999: S. 257 ff.].

[3] Die in einem Objekt intern stattfindenden Aktivitäten werden – mit Ausnahme der Methode SOM – im objektorientierten Ansatz allgemein nicht als Ereignis betrachtet.

[4] Signalobjekte werden in der UML insbesondere für die asynchrone 1-zu-N-Kommunikation zwischen Objekten verwendet, da in der UML keine explizite Multicast- und Broadcast-Interaktion definiert ist, wie dies hier der Fall ist. Vgl. [Booch 1999: S. 297], [Rumbaugh 1999: S. 427].

[5] Vgl. [Hitz 1999: S. 132].

Im Folgenden soll der Zusammenhang zwischen den unterschiedlichen Ereignistypen, der Synchronisation, unidirektionaler und bidirektionaler Interaktion und dem Modus dargestellt werden.

Der Aufruf einer Operation kann synchron oder asynchron erfolgen.[1] Synchrone Interaktion ermöglicht bidirektionale Interaktion zwischen Initiator und Participant.[2] Der Inhalt der Nachricht wird über `Content()` und `Answer()` übertragen und stellt die Ein- und Ausgabe der Operation (`In()` und `Out()`) dar. Das Erzeugen und Eintreffen eines Signals ist dagegen immer asynchron[3] und erlaubt deshalb nur unidirektionale Kommunikation. Der Inhalt der Nachricht wird über `Content()` übertragen und stellt die Eingabe (`In()`) der Operation dar. Dies kann beispielsweise die Angabe über den Ort und die Zeit des eingetretenen Ereignisses sein. Das Eintreffen eines Zeitpunktes stellt ebenfalls eine asynchrone Interaktion dar, da die Zeit weiter vergeht. Der eingetretene Zeitpunkt kann als Inhalt der Nachricht übertragen werden, wenn sie für die Operation des Participant benötigt wird:

```
Zeitfrist :Interaction
Content(31.12.1999 :Datum)
Concurrency(asynchronous)
```


Auch das Eintreffen einer logischen Bedingung wird als asynchrone oder synchrone Interaktion abgebildet. Die Art der eingetretenen Bedingung kann als Inhalt der Nachricht übertragen werden, wenn sie für die Operation des Participant benötigt wird:

```
Veränderung :Interaction
Content(Kontostand < Mindeststand :Bedingung)
Concurrency(synchronous)
```

Zwischen dem Modus der Interaktion und den Aktionstypen gilt folgender Zusammenhang: Die Informationsanfrage ist nur beim synchronen Aufruf einer Operation sinnvoll, da dies die Synchronisation von Initiator und Participant notwendig macht. Leistungsanforderung, Informationsübertragung und Inkenntnissetzung ist dagegen bei allen Formen der Aktion möglich: Damit ergibt sich folgender Zusammenhang:

[1] Vgl. [OMG 1999: S. 2-86]. Siehe auch den Abschnitt über Synchronisation (6.2.6).

[2] Vgl. [Rumbaugh 1999: S. 180].

[3] Vgl. [OMG 1999: S. 2-91].

Typ:	Aufruf	Signal	Bedingung	Zeitpunkt
Synchronisation				
Synchrone Interaktion:	Ja	Nein	Ja	Nein
Asynchrone Interaktion:	Ja	Ja	Ja	Ja
Absicht der Interaktion				
Informationsanfrage:[1]	(Ja) wenn synchron	Nein	Nein	Nein
Leistungsanforderung:	Ja	Ja	Ja	Ja
Informationsübertragung:[2]	Ja	Ja	Ja	Ja
Inkenntnissetzung:	Ja	Ja	Ja	Ja

Tabelle 10: Zusammenhang zwischen Typ der Aktion, Modus der Interaktion und Art der Synchronisation

Außerdem gilt: Die unidirektionale Übertragung des Nachrichteninhalts ist ebenfalls bei allen Aktionsformen und allen Absichten möglich. Die Übertragung eines Inhalts in umgekehrter Richtung ist nur bei der Informationsanfrage und der Leistungsanforderung möglich, wenn außerdem die Interaktion synchronisiert ist.

6.8.2 Zeit- und Veränderungsereignisse

Zeit- und Veränderungsereignisse werden verwendet, um bestimmte relevante Zeitpunkte oder das Verstreichen relevanter Zeiträume zu spezifizieren. Für die Spezifikation von Zeit- und Veränderungsereignissen wird auf entsprechende Primitive der UML zurückgegriffen.[3] *Zeitereignisse* werden in UML-Zustandsdiagrammen durch das Schlüsselwort `after()`

[1] Entspricht bidirektionaler Übertragung von Inhalten.

[2] dto.

[3] Eine Schwäche der UML ist, dass das man zwar relative und absolute Zeitangaben für Ereignisse formulieren kann, diese aber durch zwei unterschiedliche Primitive (`after` und `when`) abgebildet werden, von denen das letztere auch zur Formulierung logischer Bedingungen verwendet wird. Der Autor ist sich dieser Schwäche bewusst, hat hier jedoch der begrifflichen Übereinstimmung mit der UML den Vorzug gegeben.

festgehalten. Sie beschreiben das Verstreichen eines bestimmten Zeit*raums*. Dieser besteht aus der Angabe eines Zeitraums und optional einem Bezugspunkt.[1] Beispiele hierfür sind:

- `after(2 Sekunden)`
- `after(2 Sekunden, starttime())`

Veränderungsereignisse werden in UML-Zustandsdiagrammen durch das Schlüsselwort `when()` eingeleitet. Sie erhalten dort als Argument einen Ausdruck. Bei diesem Ausdruck kann es sich um den Bezeichner einer Fehlerbedingung[2], einen logischen Ausdruck oder um einen Zeit*punkt* handeln. Das Ereignis wird erzeugt, wenn der Ausdruck wahr wird. Beispiele hierfür sind:

- `when(11:49)`
- `when(Datum = 31.12.1999)`
- `when(wartende Kunden > 6)`
- `when(Maschine defekt)`
- `when(Tür hingefallen)`
- `when(Flug überbucht)`
- `when(Kontostand < Mindeststand)`

Signale werden in der UML sowohl als Operationen aktiver Klassen wie auch als klassenähnliche Konstrukte abgebildet.[3] Wenn Objekte über Signale kommunizieren, muss die Interaktion in zwei Schritte zerlegt werden. Im ersten Teil wird das Signalobjekt erzeugt, im zweiten Teil tritt das Signal ein, und Objekte können darauf reagieren. Dadurch, dass ein Objekt ein Signal erzeugt und beliebig viele Objekte darauf reagieren können, wird in der UML Multicast- und Broadcast-Interaktion realisiert.

Die beiden Schlüsselwörter `after` und `when` der UML werden hier als Prädikate für Aktionen verwendet. Zur Dokumentation der Bedingung für ein Veränderungsereignis wird das Prädikat `when(`*`Expression`*`)` verwendet. Die Spezifikation des Zeitereignisses geschieht über das Prädikat `after(`*`Period, Point Of Reference`*`)`.[4] Da die Aktion des Initiators die Ursache für das

---

[1] Vgl. [Rumbaugh 1999: S. 475 f.], [Booch 1999: S. 280], [OMG 1999: S. 2-126], [Hitz 1999: S. 132].

[2] Vgl. Abschnitt 6.6.10.

[3] Vgl. [Booch 1999: S. 283 ff.]. Nach KAPPEL haben Signale in der UML keine Operationen. Vgl. [Hitz 1999: S. 149]. Aus [Rumbaugh 1999: S. 428] geht jedoch hervor, dass Signale sehr wohl Operationen besitzen.

[4] Aus der von BOOCH verwendeten Formulierung eines Zeitereignisses mit „*after 1 ms since exiting Idle*“ [Booch 1999: S. 280] geht hervor, dass die Angabe eines Bezugspunktes möglich ist.

Ereignis darstellt, ist darauf zu achten, dass es nicht zu Widersprüchen zwischen den Prädikaten des Ereignisses (`after()` und `when()`) und der Aktion (`TimingConstraint()`) kommt.

Sich periodisch wiederholende Ereignisse sind kein Bestandteil der UML und werden hier durch eine besondere Form von Zeitereignissen abgebildet. Zur Abbildung wird das Prädikat `periodic(t,In,Int,n)` verwendet, welches vier Terme als Argumente erhält:

1. Einen relativen oder absoluten Startzeitpunkt t_0; abgebildet durch `t`.
2. Eine logische Konstante, ob das Ereignis zum Startzeitpunkt t_0 erzeugt werden soll. Diese wird als `In` bezeichnet.
3. Die Angabe eines Zeitintervalls; abgebildet durch `Int`.
4. Die Anzahl der Wiederholungen; abgebildet durch `n`.

Diese Möglichkeit, sich periodisch wiederholende Ereignisse zu spezifizieren, wurde von der Spezifikationssprache VDM++ übernommen.[1] Allerdings wird dort nur das Intervall angegeben. Die Verwendung von `periodic` hat zur Folge, dass ein Ereignis zu den Zeitpunkten t_0 + Int, t_0 + 2 * Int, t_0 + 3 * Int usw. erzeugt wird, bis n erreicht ist. Wird n nicht angegeben, soll das Ereignis unbegrenzt oft eintreten. Ob zum Zeitpunkt t_0 ein Ereignis ausgelöst wird, ist vom Schalter `In` abhängig. Ist dieser Schalter auf wahr oder *True* gesetzt, wird auch ein Ereignis zum Zeitpunkt t_0 erzeugt. Die Argumente können über ihre Position oder über ihr Schlüsselwort definiert werden. Ein Wecker, der morgens um 8:00 Uhr insgesamt 10 mal alle 9 Minuten klingelt, würde durch folgende Terme spezifiziert:

```
klingeln :Action
periodic(t0 = 08:00 h, In = True, Int = 9 Min, n = 10)
```

Dies kann zur Modellierung der Frequenz sich ständig wiederholender Interaktionen verwendet werden.[2] Auf diese Weise wird ausgedrückt, dass eine Verkaufsabteilung der Versandabteilung einmal am Tag die eingegangenen Bestellungen übermittelt:

---

[1] Vgl. [Lano 1995: S. 65]. Vergleiche auch [Hoheisel 1999: S. 114], der ein ähnliches, aber weniger mächtiges Konzept zur Modellierung periodischer Ereignisse vorstellt.

[2] Die hier entwickelten Prädikate ließen sich in der Art erweitern, dass man das *„Auslöseverhalten"* [Kruse 1994: S. 126] von Ereignissen durch entsprechende Prädikate abbildet. Das Prädikat `periodic()` würde dann zur Spezifikation von planbaren Ereignissen verwendet. Stochastische Ereignisse würde man durch ein Prädikat `prob()` abbilden, dessen Argumente Typ und Parameter einer Verteilungsfunktion (vgl. [Schlittgen 1999]) wären, durch die das Eintreten des Ereignisses beschrieben wird. VÖLKNER schlägt beispielsweise die Verwendung einer Dreiecksverteilung mit dem Bereich [a, b] für die Grenzen aller möglichen Ausprägungen und c für den Modalwert der Dichte vor. Vgl. [Völkner 1998: S. 137]. Dies ist späteren Erweiterungen vorbehalten.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB	übermittelt eingegangene Bestellungen :Action periodic(Int = 1 d)	:Interaction Content(:Eingegangene Bestellungen)	:Lager	verarbeitet Bestellungen :Operation

Skript 126: Interaktion mit Frequenzangabe

Um im Falle zeitabhängiger Interaktionen (abgebildet durch `Concurrency(timeout)`) angeben zu können, wie lange der Interaktionsversuch dauert, wird das für Interaktionen definierte Prädikat `Suspend()` verwendet. Auf diese Weise lässt sich auch bei periodisch wiederholenden Ereignissen eine Frist angeben, nach deren Ablauf die Interaktion nicht stattfindet, wenn der Participant nicht innerhalb der angegebenen Frist reagiert. Auf diese Weise können Gültigkeitszeiten für Ereignisse formuliert werden.[1] Hierzu als Beispiel:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Telefon	klingeln :Action periodic(startTime(), True, 3 s, 5)	:Interaction Concurrency(timeout) Suspend(12 s, klingeln.startTime())	:Person	Gespräch annehmen :Operation

Skript 127: Ungültig werdendes Ereignis

Die Spezifikation der Interaktion in Skript 127 hat die Bedeutung, dass das Telefon 5 mal im Abstand von 3 Sekunden klingelt. Das erste Klingeln findet statt, wenn die Aktion beginnt. Die Interaktion dauert 4 mal 3 Sekunden. Wenn der Participant bis dahin das Gespräch nicht annimmt, wird das Ereignis ungültig. Wenn also der Participant `Person` nicht innerhalb der angegebenen Frist reagiert, findet die Interaktion nicht statt.

Der Inhalt von Aktionen wird in der UML durch eine *Action Expression* dokumentiert.[2] Eine Aktion besteht dort aus einem oder einer Menge von Zielobjekten, einer Argumentliste, dem Namen des Signals oder der Operation. Solche Aktionen können die Zuweisung eines Wertes, der Aufruf einer Operation, das Erzeugen eines Objekts, das Zerstören eines Objekts, die Rückgabe eines Wertes, das Senden eines Signals, das Terminieren einer Aktion oder eine nicht weiter interpretierte Kontrollstruktur sein. Hier werden keine ausführbaren Programmanweisungen verwendet,[3] weil dies für einen Modellierungsansatz zur Analyse als nicht angemessen betrachtet

[1] Vgl. [Hoheisel 1999: S. 112].

[2] Vgl. [Rumbaugh 1999: S. 122 ff.].

[3] Hierfür ließen sich die *Action Semantics* der UML verwenden. Vgl. [OMG 2001b], [OMG 2002].

wird. Stattdessen wird der Inhalt der Aktion in Form natürlicher Sprache durch das Prädikat `ActExpr(String)` dokumentiert.

Das folgende Skript demonstriert die Verwendung von Objekten aus dem betrieblichen Rechnungswesen. Im Bereich der Finanzbuchhaltung soll folgender Geschäftsfall dokumentiert werden:

- Kauf einer Maschine auf Ziel zum Nettopreis von 94.000,- GE zzgl. Transportkosten in Höhe von netto 6.000,- GE. Außerdem wird die Umsatzsteuer in Höhe von 16.000,- GE gebucht.
- Das Geld wird an den Lieferanten überwiesen. Gebucht wird der Rechnungsausgleich mit 2 % Skonto durch Banküberweisung.

Die dabei tangierten Objekte sind Sachkonten, der zu buchende Geschäftsfall wird als Prozess modelliert. Mit Hilfe des Prädikats `ActExpr()` wird festgehalten, wie die zu buchenden Beträge entstehen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Lieferant	liefert Maschine :Action	:Interaction	:Maschinenanschaffung Stpe(Process)	starten :Operation
	:Maschinen-anschaffung Stpe(Process)	buche Anschaffungs-kosten :Action ActExpr(Nettopreis +Transportkosten)	:Interaction Content(100.000,- :Betrag, Soll :Seite)	0700 Technische Anlagen :SK	buche :Operation In(:Betrag, :Seite)
	:Maschinen-anschaffung Stpe(Process)	berechne Vorsteuer :Action ActExpr(Anschaffungs-kosten * 0,16 Mehrwertsteuersatz)	:Interaction Content(16.000,- :Betrag, Soll :Seite)	2600 Vorsteuer :SK	buche :Operation In(:Betrag, :Seite)
	:Maschinen-anschaffung Stpe(Process)	buche Verbindlichkeit :Action ActExpr(Anschaffungs-kosten + Vorsteuer)	:Interaction Content(116.000,- :Betrag, Haben :Seite)	4400 Verbindlichkeiten :SK	buche :Operation In(:Betrag, :Seite)
	:Buchhaltung Stpe(Departme nt)	bezahlt Maschine :Action	:Interaction	:Rechnungsausgleich Stpe(Process)	starten :Operation
	:Rechnungs-ausgleich Stpe(Process)	gleiche Verbindlichkeit aus :Action	:Interaction Content(116.000,- :Betrag, Soll :Seite)	4400 Verbindlichkeiten :SK	buche :Operation In(:Betrag, :Seite)
	:Rechnungs-ausgleich Stpe(Process)	ermittle Nettoskonto :Action ActExpr(Anschaffungs-kosten * 0,02)	:Interaction Content(2.000,- :Betrag, Haben :Seite)	0700 Technische Anlagen :SK	buche :Operation In(:Betrag, :Seite)
	:Rechnungs-ausgleich Stpe(Process)	buche Berichtigung der Vorsteuer :Action ActExpr(Nettoskonto * 0,16 Mehrwertsteuersatz)	:Interaction Content(320,- :Betrag, Haben :Seite)	2600 Vorsteuer :SK	buche :Operation In(:Betrag, :Seite)
	:Rechnungs-ausgleich	buche Überweisungsbetrag	:Interaction Content(113.680,-	2800 Bank :SK	buche :Operation In(:Betrag, :Seite)

	Stpe(Process)	:Action ActExpr(Verbindlichkeit - (Berichtigung +Nettoskonto))	:Betrag, Haben :Seite)		

Skript 128: Modellierung eines Geschäftsfalls im Bereich Finanzbuchführung[1]

Hier wurde ein Vorgang der Finanzbuchhaltung objektorientiert modelliert. Dargestellt werden die Aktivitäten in der Finanzbuchhaltung und das Ereignis, durch welches sie ausgelöst wurden. Der Vorgang wird dabei selbst als Objekt bzw. als ein Prozess modelliert. Dabei wird außer Acht gelassen, ob dieser Prozess von einem Aufgabenträger oder von einem Sachmittel ausgeführt wird. Dies kann später entschieden werden. Auf diese Weise beschränkt sich die Modellierung auf die notwendigen Aktivitäten und abstrahiert von eventuell notwendigen Anwendungssystemen zur Finanzbuchhaltung und Personen, die diese bedienen.

Dabei kommt außerdem ein Vorteil der Skriptnotation zur Wirkung, dass in den Skripten sowohl der generelle wie auch der aktuelle Verlauf dokumentiert wird: Lässt man die aktuellen Werte der Tabellenspalte Interaction unbeachtet, erhält man den generellen Ablauf des Geschäftsfalls. Betrachtet man diese Tabellenspalte, erhält man die aktuellen Werte des Geschäftsfalls.

Aktionen kann ebenfalls eine zeitliche Einschränkung zugeordnet werden, wie das folgende Skript zeigt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Verkauf	erstellt Angebot :Action TimingConstraint(executionTime() ≤ 1 h))	Angebotserstellung :Interaction	:Angebot	erstellen :Operation

Skript 129: Zeitverbrauch von Aktionen

Betrifft die zeitliche Einschränkung nur die Aktion oder Operation, wird sie als Prädikat der Initiator Responsibility bzw. Participant Responsibility erfasst. Bezieht sich die zeitliche Einschränkung auf die Interaktion oder setzt sie die Aktion und die Operation in Beziehung zueinander, wird sie in der Tabellenspalte Connection als Prädikat der Interaktion erfasst.

Auf die hier geschilderte Weise lässt sich modellieren, welche Ereignisse eintreten, auf welche Art und Weise eine Gemeinschaft von Objekten auf interne und externe Ereignisse reagiert, welche Objekte für die Reaktion verantwortlich sind und wie Abläufe durch Abfolgen von Ereignissen

realisiert werden.[2] Auch die in anderen objektorientierten Modellierungsansätzen definierten Ereignistypen lassen sich mit den hier definierten Ereignistypen abbilden.[3] Es ist zu beachten, dass in einigen Ansätzen zur GPM Ereignisse auch zur Abbildung von Zuständen verwendet werden,[4] hier aber zwischen diesen beiden Konzepten differenziert wird.

6.9 Aktive Objekte und Nebenläufigkeit

6.9.1 Aktive und reaktive Objekte

Es soll noch einmal auf die Frage zurückgekommen werden, wann ein Objekt eine Aktion ausführt. Zu Beginn von Abschnitt 6.6 wurde festgelegt, dass eine Aktion im Rahmen der Ausführung einer Operation stattfindet. Dies stellt aber nur eine Möglichkeit dar. Wenn Aktionen immer nur im Rahmen einer Operation stattfinden, kann kein Skript mit einer Interaktion beginnen, da immer eine vorhergehende Interaktion mit einer Operation notwendig ist. Die Lösung dieses Problems liegt im Konzept der *aktiven Objekte*.

Objekte werden als *aktiv* angesehen, wenn sie ohne ein vorhergehendes Ereignis Aktionen ausführen. Solche aktiven Objekte werden in aktuellen Modellierungsansätzen nur unzureichend berücksichtigt,[5] obwohl in der Analyse davon ausgegangen wird, dass alle Objekte aktiv sind.[6] Folgende Fälle werden von SAAKE unterschieden: [7]

---

[1] SK steht für Sachkonto. Die Buchungssätze basieren auf [Schmolke 1999: S. 214]. Verwendet wird der Industriekontenrahmen für die Aus- und Fortbildung.

[2] In [Winant 1996], [Winant 1996b], [Winant 1997] findet sich in der Gestalt des so genannten *Schlüsselereignisverzeichnisses* ein ähnliches Konzept in rudimentärer Form. Ebenfalls ähnlich sind die *Ereignistabellen* der Methode SYNTROPY. Vgl. [Cook 1994].

[3] Es wurde davon Abstand genommen, die von Martin verwendeten Klassifizierungsereignisse zu übernehmen, da im hier verwendeten Objektansatz Objekte ihre Rollen, nicht jedoch ihre Klassen wechseln. Vgl. [Martin 1999: S. 177].

[4] Vgl. [Ferstl 1993b: S. 143], [Ferstl 1998: S. 93], [Scheer 1995: S. 40 f.].

[5] Vgl. [Saake 1993: S. 175]. Dies wird durch die entsprechenden Kapitel und Abschnitte in [Booch 1999: S. 310 ff.] und [Rumbaugh 1999: S. 130 ff., 384] bestätigt. Oft wird auch nicht klar differenziert, ob ein Objekt aktiv oder nebenläufig ist. Jedes aktive Objekt ist nebenläufig, aber nicht jedes nebenläufige Objekt ist auch aktiv. Vgl. [Firesmith 1995: S. 12, 100 f.].

[6] Vgl. [de Champeaux 1993: S. 62, 277].

[7] Vgl. [Saake 1993: S. 175 f.].

1. Ein Objekt wird durch eine Operation dazu aufgefordert, aktiv zu werden.[1] Im Rahmen dieser Operation führt das Objekt eine oder mehrere Aktionen aus. In diesem Fall sind die Aktionen die Folge bzw. die Realisierung der Operation dieses Objekts. Ein solches Objekt ist *reaktiv*, da seine Aktionen als Reaktion auf ein eintreffendes Ereignis stattfinden.
2. Ein *aktives* Objekt führt Aktionen aus, ohne dass es durch einen Operationsaufruf dazu aufgefordert wird.[2]
3. Ein Objekt ist *spontan aktiv*, wenn es ohne jede äußere Einwirkung eine Aktion ausführt.[3]

Zwischen spontan aktiven Objekten und aktiven Objekten wird in der OBA++ nicht unterschieden. Die Modellierung aktiver Objekte spielt insbesondere dann eine große Rolle, wenn keine rein reaktiven Systeme untersucht werden. Dies trifft bei allen soziotechnischen Systemen zu: Dort ist die Existenz aktiver Objekte – die Mitarbeiter – die Regel. Aus diesem Grund wird hier davon ausgegangen, dass alle Objekte inhärent aktiv sein können. Solche Objekte werden in der Methode ROOM als *Akteure* bezeichnet.[4] Im Gegensatz zur UML wird hier nicht zwischen aktiven und passiven Objekten unterschieden,[5] was zur Folge hat, dass jedes Objekt prinzipiell als aktiv angesehen wird. Das bedeutet, dass jedes Objekt eine Aktion ausführen kann, ohne vorher im Rahmen einer Operation aktiviert worden zu sein.

---

[1] DE CHAMPEAUX bezeichnet dies als aktiven Zustand, wogegen der Zustand, in dem ein Objekt auf eine eingehende Nachricht wartet, als passiver Zustand bezeichnet wird. Vgl. [de Champeaux 1993: S. 63]. Hierzu ist zu bemerken, dass in der objektorientierten Modellierung nur passive Zustände als Zustand abgebildet werden. Aktive Zustände werden als Zustandsübergang abgebildet.

[2] Vgl. [Firesmith 1995: S. 12]. Ob ein Objekt als aktiv angesehen wird, ist eine Frage des betrachteten Zeitraums. Wenn SAAKE das Ticken einer Uhr als aktiv bezeichnet (vgl. [Saake 1993: S. 176]), kann man dagegen einwenden, dass die Uhr aufgezogen bzw. von einem Energielieferanten gespeist werden muss. Dies stellt eindeutig ein Ereignis für das Ticken dar. Auch das von ihm erwähnte Ausleihen eines Buches durch einen aktiven Benutzer ist letztlich eine Frage des Standpunktes. Für die objektorientierte Modellierung mag es aber genügen, Objekte, die innerhalb des betrachteten Realitätsausschnittes ein Ereignis auslösen, ohne selbst dazu aufgefordert worden zu sein, als aktiv zu bezeichnen. Als eine mögliche Lösung dieses Problems stellt er die Konstruktion vor, Objekte mit Eigeninitiative als reaktive Objekte darzustellen, die durch ein Geburtsereignis einen unendlichen Prozess beginnen, in dessen Rahmen sie Ereignisse für andere Objekte auslösen. Vgl. [Saake 1993: S. 180].

[3] Vgl. [Saake 1993: S. 176].

[4] Vgl. [Selic 1994: S. 149].

[5] Vgl. [Rumbaugh 1999: S. 131 f.].

6.9.2 Nebenläufigkeit von Operationen und Synchronisation der Interaktion

Das Prädikat `Concurrency()` wird nur scheinbar redundant für Aktivitäten und Interaktionen verwendet. Es muss zwischen Interobjekt- und Intraobjekt-Parallelität unterschieden werden. Die Synchronisation der Interaktion bestimmt, ob Initiator und Participant in irgendeiner Form aufeinander warten müssen, damit besagte Interaktion stattfindet. Die Synchronisation von Aktivitäten dagegen drückt aus, ob Operationen – unabhängig von einer bestimmten Interaktion – mehrfach parallel ausgeführt werden können. Dass diese Differenzierung sinnvoll ist, soll an folgendem Beispiel demonstriert werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Radio-sender	Programm ausstrahlen :Action	:Interaction Concurrency(synchronous)	:Radio	empfängt Programm :Operation Concurrency(Sequential)

Skript 130: Sequentielle Operation und asynchrone Interaktion

In diesem Skript ist die Interaktion zwischen dem Radiosender und dem Empfänger synchronisiert. Wenn der Radiosender aufhört zu senden, wird der Empfänger – mit einer hier vernachlässigten Verzögerung – das Radioprogramm nicht mehr empfangen. Der Umstand, dass der Empfänger in einem bestimmten Moment nur ein Programm empfängt, ist davon unabhängig und wird durch das Prädikat `Concurrency(Sequential)` der Operation festgehalten.

Eine andere Situation wird durch das nächste Skript abgebildet:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Abteilungs-leiter	erteilt Anweisung:Action	:Interaction Concurrency(asynchronous)	:SB	Auftrag ausführen:Operation Concurrency(guarded)

Skript 131: Parallele Aktion und asynchrone Interaktion

Diese Interaktion findet asynchron statt, da der Abteilungsleiter nicht wartet, bis der Sachbearbeiter den Auftrag ausgeführt hat. Der Sachbearbeiter wiederum kann mehrere Aufträge entgegennehmen, aber immer nur einen ausführen.

6.9.3 Nebenläufigkeit von Aktivitäten

Eine Anmerkung zum Problem der Nebenläufigkeit von Aktivitäten: Nebenläufigkeit ist in allen objektorientierten Modellierungsansätzen nur auf der Ebene einzelner Aktivitäten definiert.[1] Für jede Aktivität wird festgelegt, ob sie parallel ausgeführt bzw. angefordert werden kann. Bestimmte Aktivitäten können parallel ausgeführt werden (`atmen` und `telefonieren`), andere Aktivitäten schließen sich gegenseitig aus (`Vertrag ausfüllen` und `schlafen`). Daraus ergibt sich, dass die Möglichkeit der nebenläufigen Ausführung einer Aktivität nicht auf der Ebene der einzelnen Aktivität entschieden werden kann, sondern auf der Ebene des Objekts entschieden werden müsste, da alle nebenläufigen Aktivitäten untereinander in einem Abhängigkeitsverhältnis stehen. Dieses Problem wird auch als *Interferenz* bezeichnet.

Das bedeutet, dass dynamisch geprüft werden muss, ob ein Objekt zu einem bestimmten Zeitpunkt die Kapazität besitzt, eine weitere Aktivität auszuführen. Eine solche Modellierung, wann die Kapazität eines Objekts erschöpft ist, unterschiedliche Aktivitäten parallel auszuführen, findet sich bisher in keinem objektorientierten Ansatz. Hierfür kämen primär Zustandsdiagramme mit Parallelität in Form von HAREL-Statecharts in Frage.[2] Auf diese Weise könnten bewährte Prinzipien anderer Gebiete (beispielsweise aus dem Bereich der Betriebssysteme)[3] zur Untersuchung und Lösung von Nebenläufigkeitsproblemen auf die objektorientierte Modellierung übertragen werden.

Ein zusätzliches Problem entsteht bei der Festlegung, wie Parallelität genau verstanden werden soll. Beinhaltet die Aussage, ein Sachbearbeiter könne mehrere Vorgänge bearbeiten, tatsächlich Parallelität? Zu einem bestimmten Zeitpunkt t_n kann dieser Sachbearbeiter immer nur einen Vorgang bearbeiten. SHLAER weist denn auch darauf hin, dass man zwischen echter Parallelität und scheinbarer Parallelität unterscheiden muss, in der parallele Vorgänge sequentialisiert werden, wie dies zum Beispiel moderne Betriebssysteme tun.[4]

Insofern decken die in der objektorientierten Modellierung mit der UML verwendeten Konstrukte eher bestehende Unzulänglichkeiten auf, als sie zu lösen. Um einer sachlichen Diskussion willen

---

[1] Dies ist in allen bekannten Modellierungsansätzen so. Nur PAGE-JONES differenziert in einem Aufsatz zwischen Objekt-Nebenläufigkeit und Methoden-Nebenläufigkeit (vgl. [Page-Jones 1994b]). Diese Unterscheidung hat aber bis heute keine Verbreitung gefunden.

[2] Vgl. [Harel 1987].

[3] Vgl. [Wettstein 1993], [Tanenbaum 1994].

[4] Vgl. [Shlaer 1992: S. 104].

muss allerdings erwähnt werden, dass sich andere Ansätze diesem Problem überhaupt nicht stellen[1] und Fragen der Nebenläufigkeit entweder aussparen oder nur ansatzweise erörtern.[2] Dies sollte aber nicht dazu führen, mögliche Nebenläufigkeit in der Analyse überhaupt nicht zu betrachten. Die Fähigkeit zur Ausführung nebenläufiger Aktivitäten zu ignorieren beseitigt nicht das Problem, sondern verschleiert es.

6.10 Attribute

Als Attribut wurden die inhärenten Eigenschaften, Qualitäten oder Besonderheiten von Konzepten definiert. Sie stellen die Abstraktionen der Informationen dar, die zu einem Objekt gehören.[3] Auch wenn durch Skripte keine Datenmodellierung erfolgt, besteht die Notwendigkeit, die Attribute von Klassen spezifizieren zu können. Nach dem Geheimnisprinzip dürfen Attribute nur durch die Operationen der eigenen Klasse verändert werden. Sie sind zwar für den Modellierer sichtbar, nicht jedoch für andere Objekte und Klassen. Der Attributname wird in der Tabellenspalte Participant Responsibility in der Form `Attributname :Attribut` festgehalten. Der Attributname ist innerhalb der Klasse eindeutig und muss zwingend angegeben werden. Folgende Aspekte[4] von Attributen werden durch Prädikate abgebildet:

- Ob ein Attribut veränderbar ist, wird durch das Prädikat `Changeable(`*`Changeability Identifier`*`)` dokumentiert.[5] Attribute können veränderbar (`changeable`) oder nicht veränderbar (`frozen`) sein, oder es ist nur das Hinzufügen (`add only`) zugelassen. Das Attribut `Geburt` würde man beispielsweise als `changeable(frozen)` festlegen, da üblicherweise das Geburtsdatum einer Person nicht verändert werden kann.
- Wenn ein Attribut einen Anfangs- bzw. Initialisierungswert besitzt, wird dieser durch das Prädikat `Init(`*`Expression`*`)` dokumentiert.[6] Das Attribut Familienstand einer Person könnte beispielsweise mit `Init("ledig")` initialisiert werden.

---

[1] Vgl. [Kappel 1996], [Schienmann 1997], [Waldén 1995], [Coleman 1994], [Allen 1998], [Wirfs-Brock 1993].

[2] Vgl. [Embley 1992], [Martin 1995: S. 170 ff.], [Rumbaugh 1993: S. 120]. Die in [Cook 1994] und [Meyer 1997] entwickelten Konzepte werden hier nicht berücksichtigt, da sie sich mit den Problemen der Nebenläufigkeit auf der Ebene von Programmiersprachen-Konzepten befassen.

[3] Siehe Definition des Begriffs Attribut im Glossar.

[4] DE CHAMPEAUX bezeichnet diese Aspekte als Attribute für Attribute oder *Features*. Auf die von DE CHAMPEAUX erwähnte Möglichkeit, für die Ausprägungen eines Attributs die Wahrscheinlichkeitsverteilung anzugeben, wird hier nicht eingegangen. Vgl. [de Champeaux 1993: S. 29].

[5] Vgl. [Rumbaugh 1999: S. 166].

[6] Vgl. [OMG 2001: S. 3-45].

- Durch das Prädikat `Type(Type Identifier)` wird der Datentyp eines Attributs festgelegt. Hier erfolgt keine abschließende Definition eines Typsystems, da dieses immer vom zu untersuchenden Problembereich abhängig ist.[1] Einige grundlegende Unterscheidungen können jedoch vorgenommen werden. Attribute können Datentypen wie natürliche Zahlen (*Integer*), Zeichenketten (*String*), reelle Zahlen (*Real*), Wahrheitswerte (*Boolean*) sein oder aus dem Problembereich übernommene Fachbegriffe, die als Datentyp angesehen werden, wie z. B. Datum, Temperatur, Steuerklasse, Sehschärfe oder Zeitangaben. Als Typbezeichnung können auch Typ-Konstruktoren wie Menge (*Set*), Multimenge (*Bag*), Feld (*Array*), Liste (*List*), Kellerspeicher (*Stack*), Schlange (*Queue*) oder Aufzählung (*Enumeration*) verwendet werden. Stellt ein Attribut eine Schlange mit Integerwerten dar, wird dies als `Type(Queue of Integer)` abgebildet. Für ein Attribut `Papiergröße` kann mit Hilfe der Aufzählung `Type(Enumeration(A0, A1, A2, A3, A4))` festgelegt werden, dass die Papiergröße nur die aufgezählten Werte annehmen kann.
- Wird kein Typ-Konstruktor angegeben, kann das Prädikat `Card(Cardinality Expression)` verwendet werden, um die Kardinalität eines Attributs bestimmen zu können. Attribute müssen die Fähigkeit besitzen können, mehrwertig zu sein, da sonst die Modellierungsanomalie entstehen würde, dass einwertige Daten (Daten in erster Normalform) als Attribute, mehrwertige Daten (Daten nicht in erster Normalform) als Assoziationen modelliert würden. Soll ein Attribut `Telefonnummer` drei Werte beinhalten können, wird dies als Kardinalität `Card(1..3)` festgelegt. Durch die Angabe von `Card(0,1)` wird bestimmt, dass ein Attribut nicht zwingend einen Wert besitzen muss, also optional ist. Wenn die obere Grenze größer als 1 ist, kann angegeben werden, ob die Attributwerte geordnet (`ordered`) oder ungeordnet (`unordered`) sind.[2]
- Das Prädikat `Qualifier(Identifier List)` beinhaltet einen oder mehrere Bezeichner, durch die die Semantik des Attributs genauer bestimmt werden kann.[3] `Unique` wird für Attribute verwendet, deren Werte für alle Exemplare einer Klasse unterschiedlich sind, also einen Schlüssel darstellt.[4] `Common` wird für Attribute verwendet, deren Wert für alle Exemplare einer Klasse identisch sind, es sich also um ein Klassenattribut handelt.[1]

---

[1] Auch in der UML wird kein Typsystem für Attribute vorgeschrieben. Vgl. [OMG 2001: S. 3-45].

[2] Dies entspricht den Vorgaben der UML. Vgl. [OMG 2001: S. 4-45]. Es wird jedoch nicht angegeben, ob die Werte auf- oder absteigend geordnet sind.

[3] Vgl. [Schienmann 1997: S. 177 ff.], [Graham 1998: S. 94], [Cook 1994: S. 47], [de Champeaux 1993: S. 32].

[4] Der geringe Stellenwert identifizierender Attributwerte mag überraschen, wenn man ihre Bedeutung im Bereich relationaler Datenbanksysteme [Date 1990], [Kemper 1999] und der darauf auf basierenden Entity-

- Das Prädikat `Subrange(Expression)` schränkt die möglichen Werte eines Attributs durch Festlegung eines Auswahlbereichs ein. Für das Attribut Alter eines Geschäftskunden könnte beispielsweise `Subrange(x ≥ 18)` festgelegt werden, wobei x als Platzhalter für den Wert des Attributs betrachtet wird.
- Das Prädikat `Dimension(String)` wird verwendet, um eine Maßeinheit angeben zu können.

Weitere Prädikate zur Bestimmung von Attributen werden hier nicht verwendet, können aber bei Bedarf hinzugefügt werden.[2] Ein Prädikat zur Dokumentation der Sichtbarkeit eines Attributs erfolgt hier – im Gegensatz zur UML[3] – nicht, da auf Attribute nicht extern zugegriffen werden kann. Attribute besitzen somit immer implizit die Sichtbarkeitskategorie `private` .

6.11 Zeilennummern

6.11.1 Motivation

Im folgenden Skript prüft ein Sachbearbeiter zuerst anstehende Fristen und schreibt dann – bei nicht eiligen Terminen – dem Kunden einen Brief:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	prüft Termine :Action	:Interaction	:Kalender	anstehende Fristen prüfen :Operation
	:SB	schreibt Brief :Action	:Interaction Guard(Termin nicht eilig)	:Brief	erstellen :Operation
	:SB	verschickt Brief :Action	:Interaction	/Kunde :Person	erhält Brief :Operation

Skript 132: Erster Ablauf in einem Skript

In einem anderen Fall (bei eiligen Terminen) wird dem Kunden ein Brief geschrieben. Das folgende Skript ähnelt dem letzten im Inhalt:

Relationship-Modellierung [Linssen 1999e] vergleicht. In den Standardwerken zur objektorientierten Modellierung werden sie nicht behandelt, weil jedes Objekt *per se* über seine Objektidentität eindeutig identifizierbar ist. Diesem Ansatz wurde hier gefolgt.

[1] Dies wird in der UML durch den *owner scope* festgelegt. Vgl. [Rumbaugh 1999: S. 167].

[2] Vgl. hierzu die ausführliche Darstellung in [Schienmann 1997: S. 177 ff.].

[3] Vgl. [OMG 1999: S. 2-32].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	prüft Termine :Action	:Interaction	:Kalender	anstehende Fristen prüfen :Operation
	:SB	ruft an :Action	:Interaction Guard(Termin eilig)	/Kunde :Person	erhält Anruf :Operation

Skript 133: Zweiter Ablauf in einem Skript

Offensichtlich werden hier zwei Abläufe dokumentiert, die sich bis zu einem bestimmten Punkt gleichen. Die Skripte bilden die beiden Pfade einer Selektion ab. Es ist nahe liegend, beide Skripte zusammenzufassen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	prüft Termine :Action	:Interaction	:Kalender	anstehende Fristen prüfen :Operation
	:SB	ruft an :Action	:Interaction Guard(Termin eilig)	/Kunde :Person	erhält Anruf :Operation
	:SB	schreibt Brief :Action	:Interaction Guard(Termin nicht eilig)	:Brief	erstellen :Operation
	:SB	verschickt Brief :Action	:Interaction	/Kunde :Person	erhält Brief :Operation

Skript 134: Der gesamte Ablauf in einem Skript

Durch eine einfache Aneinanderreihung von Skript 132 und Skript 133 entsteht jedoch keine Selektion, sondern eine Sequenz. Es ist nicht gut zu erkennen, dass der Kunde entweder angerufen oder angeschrieben wird. So könnte der Eindruck entstehen, dass er in jedem Fall einen Brief erhält – was nicht beabsichtigt war.

Damit man ausdrücken kann, dass Zeilen alternativ durchlaufen werden sollen, ist das Skriptmodell entsprechend zu erweitern. Das erfordert eine Notation, um in der tabellarischen Darstellung der Skripte die Abfolge von Interaktionen bzw. Ereignissen durch Kontrollanweisungen darstellen zu können. Dieser Ereignis- bzw. Kontrollfluss soll im Konzeptuellen Schema abgebildet werden. Wenn prozeduraler Kontrollfluss dargestellt werden soll, erhalten die Zeilen eines Skripts in der Tabellenspalte Number einen so genannten Sequenzausdruck. Grundlage der Sequenzausdrücke sind Sequenznummern.

6.11.2 Sequenznummern

Das Prinzip der Sequenznummern basiert auf Konzepten, die in anderen objektorientierten Ansätzen dazu verwendet werden, in Interaktionsdiagrammen die Abfolge des Nachrichten-

austauschs zu dokumentieren. Diese Konzepte stellen den Ausgangspunkt des hier entwickelten Verfahrens dar, welches erweitert wird, um komplexere Situationen abbilden zu können. Zu nennen wären:

- Die in Kollaborationsdiagrammen der UML verwendeten Nachrichtenmarken (*message label*).[1]
- Die in den Objektinteraktionsgraphen der Methode FUSION verwendeten Pfeilmarken (*arrow label*).[2]
- Die Nachrichtennummern der Methode CATALYSIS.[3]
- Die in Objektdiagrammen (dort als *Mechanismen* bezeichnet) verwendeten Nachrichtennummern der Methode SYNTROPY.[4]
- Die Sequenznummern im Ereignismodell der Methode MOSES.[5]

Die Verwendung des Sequenzausdrucks ist für jede Zeile des Skripts optional. Ein Sequenzausdruck besteht aus einer Sequenznummer und optional einem Vorgängerausdruck, der durch einen Schrägstrich abgetrennt wird. Jede Sequenznummer ist innerhalb eines Skripts eindeutig. Die Sequenznummer basiert auf einer Dezimalklassifikation. Sie besteht aus einer Abfolge von Sequenznummernebenen, die durch einen Punkt beendet werden. Auf der obersten Ebene erhält man auf diese Weise `1.`, `2.`, `3.` usw. Die nächste Ebene hat die Form `1.1.`, `1.2.`, `1.3.`, `1.4.` usw. Abweichend vom üblichen Verfahren in anderen Methoden werden die Sequenznummern in der UML um Buchstaben erweitert. Eine Sequenznummernebene besteht folglich mindestens aus einer laufenden Zahl, die optional ein Präfix und ein Postfix in Form eines arabischen Buchstabens erhalten kann. `1.`, `2.`, `3.`, `A4.`, `B6.`, `1.2.`, `4.2.1.`, `A1.`, `B5.4b.6.` oder `1.B2.3b.` sind gültige Sequenznummern.

Der Vorgängerausdruck besteht mindestens aus einer Sequenznummer, die ebenfalls im Skript vorhanden sein muss. Alternativ besteht der Vorgängerausdruck aus einem Sequenznummernausdruck, der aus Sequenznummern, Klammern, dem Vorgängeroperator „/“ und den Operatoren Negation, Konjunktion, Disjunktion, und Adjunktion[6] gebildet wird. Hierfür werden folgende Zeichen verwendet:

---

[1] Vgl. [Booch 1999: S. 207, 213, 214, 217, 248 f., 251 – 254, 314, 324], [Rumbaugh 1999: S. 132, 201, 337], [Hitz 1999: S. 120], [Allen 1998: S. 129].

[2] Vgl. [Coleman 1994: S. 281 f.].

[3] Vgl. [D’Souza 1998: S. 216].

[4] Vgl. [Cook 1994: S. 158], [Cook 1994b].

[5] Vgl. [Henderson-Sellers 1994: S. 300].

[6] Vgl. [Kleinknecht 1976b: S. 74]

Operator	Mnemonik	Zeichen
Negation	NOT	!<Operand>
Konjunktion	AND	<Operand1> & <Operand2>
Disjunktion	XOR	<Operand1> ^ <Operand2>
Adjunktion	OR	<Operand1> | <Operand2>

Tabelle 11: Zeichen für Operatoren der Tabellenspalte Number

Die Sequenznummernausdrücke `1.`, `2.|5.`, `1.2.3.&3.2.4.`, `(3. & 1.)|(2. & 1.)`, `4.&6.` sind Beispiele für zulässige Sequenznummernausdrücke. Durch diese Sequenzausdrücke wird die Abfolge von Interaktionen, Zustandsübergängen und Attributzugriffen festgelegt. Zeilen, die statische Beziehungen ausdrücken, erhalten keinen Sequenzausdruck. Folgende Regeln zur Zeilennummerierung werden von den Nachrichtenmarken der UML übernommen:

- Die Sequenznummer besteht aus einer oder mehreren Sequenznummernebenen, die durch einen Punkt getrennt werden. Diese Ebenen bedeuten eine Schachtelung.[1] Führt der Participant im Rahmen der Ausführung einer Operation geschachtelt Interaktionen durch, wird dies durch eine neue Schachtelungsebene abgebildet. Die Abfolge `2., 2.1., 2.2.` bedeutet, dass im Rahmen der Interaktion `2.` von einem Participant eine Operation angefordert wurde, zu dessen Erfüllung dieser die Interaktionen `2.1.` und `2.2.` ausführt. Die Interaktion `2.` ist der Auslöser für die Interaktionen `2.1.` und `2.2.` Die Zuordnung geschachtelter Interaktionen ist eindeutig festzustellen, indem man die letzte Schachtelungsebene der Interaktion streicht.[2] Das bedeutet, dass man durch Weglassen der letzten Schachtelungsebene die zugehörige „Ober-Interaktion“ erhält.
- Erhält eine Zeile im Skript die Nummer n+1, bedeutet dies, dass der dokumentierte Umstand nach der Zeile mit der Nummer n eintreten kann.[3] Eine Zeile mit der Nummer n hat implizit den Vorgänger mit der Nummer n-1.
- Unterscheiden sich mehrere Zeilen mit identischer Nummer nur durch den Postfix (Beispiel: `1a.` und `1b.`, `3c.` und `3d.`), bedeutet dies, dass die Zeilen mit der Sequenznummer *n* und

[1] Vgl. [Booch 1999: S. 212 ff.].

[2] Vgl. [OMG 1999: S. 3-116, 3-119].

[3] Vgl. [Rumbaugh 1999: S. 337].

unterschiedlichem *Postfix*-Buchstaben parallel nach der Zeile mit der Sequenznummer *(n-1) Postfix* stattfinden werden.[1]

- Unterscheiden sich mehrere Zeilen mit identischer Nummer durch den Präfix (zum Beispiel `A1.` und `B1.`, `A3.` und `B3.`), bedeutet dies, dass die Zeilen zu unabhängig voneinander ausgeführten Prozessen gehören.[2]
- Unterscheiden sich zwei Sequenznummern mit identischem Präfix durch ihre laufende Zahl, bedeutet dies, dass die Zeilen innerhalb des Prozesses sequentiell ausgeführt werden (`G3.` und `G4.`).[3]
- Andere Folgebeziehungen werden explizit durch Angabe eines Vorgängers bestimmt.[4] Der Vorgänger einer Zeile `C2.` wäre die Zeile `C1.` Soll dagegen `C2.` auf `1.` folgen, wird dies durch den Vorgängeroperator „/“ festgehalten. Der entsprechende Ausdruck lautet also `1./C2.`
- Besitzt eine Interaktion `N.` eine geschachtelte Interaktion `N.1.`, ist keine Angabe eines Vorgängers notwendig.
- Nach Abschluss der letzten Interaktion einer untergeordneten Ebene findet ohne Angabe eines Vorgängers die nächste Interaktion der übergeordneten Ebene statt.[5]

6.11.3 Modellierung von Kontrollstrukturen mit Sequenznummern

Der Aufbau des Vorgängerausdrucks muss erweitert werden, um komplexere Kontrollflussgraphen abbilden zu können, als dies in der UML möglich ist.[6] Die Angabe eines oder mehrerer Vorgänger

---

[1] Vgl. [Hitz 1999: S. 120].

[2] Vgl. [Booch 1999: S. 214]. Hierbei ist zu beachten, dass keine Aussage darüber getroffen wird, dass zwei Interaktionen mit der gleichen laufenden Nummer in unterschiedlichen Prozessen zeitlich parallel stattfinden. Dies müsste über ein Prädikat vom Typ `TimingConstraint()` ausgedrückt werden. Es wird nur ausgedrückt, dass die Zeilen A3. und B3. die Ordnungsnummer 3 in zwei unabhängigen Abläufen besitzen.

[3] Vgl. [Booch 1999: S. 318].

[4] Vgl. [Rumbaugh 1999: S. 132].

[5] Vgl. [Rumbaugh 1999: S. 337], [Booch 1999: S. 213 f.].

[6] Durch objektorientierte Modellierung reduziert sich erfahrungsgemäß die prozedurale Komplexität von Systemkomponenten. Vgl. [Martin 1993: S. 43]. Dies lässt sich überprüfen, indem man mit Hilfe der McCabe-Metrik die zyklomatische Zahl des Kontrollflussgraphen einer Systemkomponente ermittelt. Die zyklomatische Zahl V eines Graphen G wird mit der Formel V(G) = e – n + 2p ermittelt, wobei e für die

bedeutet in der Notation der UML, dass diese Interaktionen alle eingetreten sein müssen, bevor die nächste Interaktion stattfinden kann. So bedeutet `1.2., 3.4. / 5.`, dass Interaktionen `1.2.` und `3.4.` stattgefunden haben müssen. Alle Vorgänger stehen folglich in einer Konjunktionsbeziehung. Da dieser Ansatz keine Abbildung von Iterationen zulässt, werden die schon erwähnten logischen Operatoren für Vorgänger eingeführt. Die Angabe `1.2. & 3.4. / 5.` entspricht der Formulierung `1.2., 3.4. / 5.` der UML. Dagegen bedeutet `5. | 3. / 4.`, dass die Interaktion `4.` entweder nach der Interaktion `5.` oder `3.` stattfindet.[1] In diesem Fall wird der direkte Vorgänger angegeben. Die Sequenznummer `1.` hat implizit den Vorgänger `0.`, der für keine Zeile eines Skriptes verwendet wird. Eine Zeile mit der Nummer n hat immer nur eine Zeile mit der Nummer n-1 implizit zum Vorgänger. Eine Zeile mit einer anderen Nummer (n-m) muss dagegen durch den Vorgängeroperator explizit gesetzt werden. Nummern brauchen auch nicht in geschlossener Folge verwendet zu werden. Die Zeilen eines Skripts können mit 1., 3., und 5. nummeriert werden. Dadurch entsteht aber keine Reihenfolgebeziehung. Diese entsteht erst durch Verwendung des Vorgängeroperators und lautet 1., 1./3., 3./5.

Damit bedeutet (vgl. Abbildung 110 auf Seite 354)

- `1., 2., 3.` eine sequentielle Abfolge von Zeilen (in der Abbildung A).
- `1., 1./ 3., 4., 1./ 6., 7., 1./ 8., 9.`: Nach der Zeile `1.` wird entweder die Sequenz `3.` und `4.`, die Sequenz `6.` und `7.` oder die Sequenz `8.` und `9.` ausgeführt. Welcher Zweig durchlaufen wird, bestimmt das Prädikat `Guard` der Interaktionen 3., 6. und 8 (in der Abbildung B).
- `1., 1./ 3., 4., 1./ 7., 8., 8./ 10., 8./ 11., 1./ 5., 6., 10./ 12., 10./ 13.`: Mehrfache Alternativen (in der Abbildung C).
- `1., 1./ A2., A3., 1./ B2., B3., 1./ C2., C3.`: Nach der Zeile `1.` werden die Zeilen `A2.`, `B2.`, und `C2.` sowie `A3.`, `B3.` und `C3.` als parallele Prozesse ausgeführt (in der Abbildung D).

Die folgenden UML-Aktivitätendiagramme verdeutlichen den Ablauf, wobei die Interaktionen als Aktivitäten dargestellt werden:[2]

Anzahl der Kanten des Kontrollflussgraphen, n für die Anzahl der Knoten und p für die Anzahl der verbundenen Komponenten steht. Vgl. [Balzert 1998: S. 481], [Henderson-Sellers 1994: S. 492 f.].

[1] In der UML ist das nicht möglich.

[2] Vgl. [Booch 1999: S. 257 ff.]. Hier ist zu beachten, dass keine Abfolge von Aktionen im Sinne der UML abgebildet wird, sondern die Abfolge von Interaktionen.

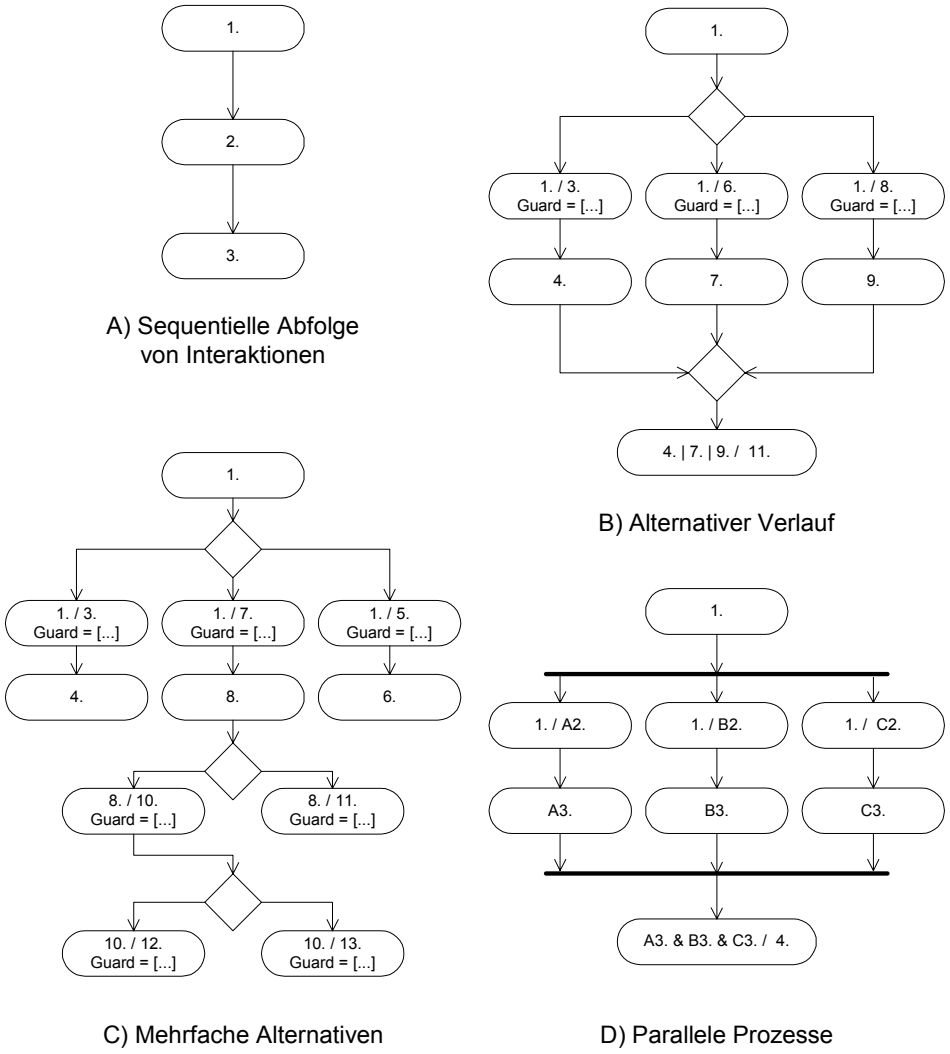


Abbildung 110: Graphische Darstellung von Sequenzausdrücken

Die Angabe von Vorgängern bedeutet eine Synchronisation einer Zeile eines Skripts mit den im Vorgänger angegebenen Zeilen. Der in der aktuellen Zeile angegebene Umstand tritt ein, wenn die angegebenen Vorgängerzeilen stattgefunden haben. Die in Abbildung 110 B) und 6 D) verwendeten Vorgänger sind wie folgt zu interpretieren:

- `4. | 7. | 9./ 11.` bedeutet, dass der in der Zeile `11.` geschilderte Umstand stattfindet, wenn einer der in den Zeilen `4.` oder `7.` oder `9.` geschilderten Umstände stattgefunden hat.[1]
- `A3. & B3. & C3./ 4.` bedeutet, dass der in der Zeile `4.` geschilderte Umstand stattfindet, wenn alle in den Zeilen `A3., B3.` und `C3.` geschilderten Umstände stattgefunden haben.[1]

[1] Siehe Abbildung 110, B).

Die Sequenznummern `1.`, `2.`, `3.`, `1.1.`, `1.2.`, `1.3.`, `1.1.1.`, `1.1.2.`, `1.1.3.` bedeuten: Im Rahmen der Ausführung der in Zeile `1.` geschilderten Umstände werden die in den Zeilen `1.1.`, `1.2.` und `1.3.` geschilderten Umstände ausgeführt. Zur Erfüllung der Zeile `1.1.` werden die Zeilen `1.1.1.`, `1.1.2.` und `1.1.3.` ausgeführt. Durch `1.1.3.` ist `1.1.` abgeschlossen. Durch `1.3.` ist `1.` abgeschlossen. Der Nachfolger von `1.1.3.` ist implizit `1.2.`, der Nachfolger von `1.3.` ist implizit `2`. Dies bezeichnet man als prozeduralen bzw. geschachtelten Kontrollfluss, da er durch geschachtelte Interaktionen realisiert wird.

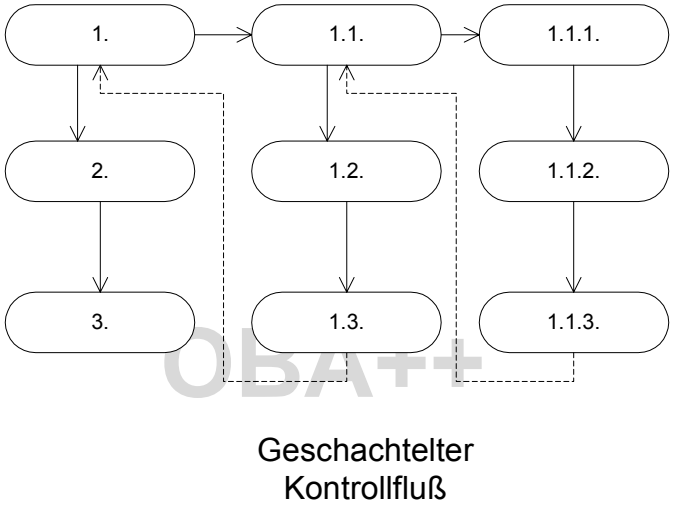


Abbildung 111: Graphische Darstellung der prozeduralen Abfolge

Folgendes Beispiel demonstriert die Verwendung von Nummern:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:AL	Erteilt Anweisung :Action	a :Interaction	:SB	erhält Anweisung :Operation
1. / 1.1a.	:SB	Erstellt Rechnung :Action	b :Interaction	:Rechnung	wird erstellt :Operation
1. / 1.1b.	:SB	Hält Rücksprache mit Kollege	c :Interaction	:SB	gibt Auskunft :Operation
1.1b.1.	:SB	Prüft Kunde :Action	d :Interaction	:Datenbank	Anzeige Daten :Operation
1.2.	:SB	Ruft Kunde an :Action	e :Interaction	/Kunde :Person	wird angerufen :Operation
1.3.	:SB	Verschickt Rechnung :Action	f :Interaction	/Kunde :Person	erhält Rechnung :Operation
2.	:AL	Ruft Kunde an :Action	g :Interaction	/Kunde :Person	wird angerufen :Operation
3.	:AL	Schließt Vorgang ab :Action	h :Interaction	:Vorgang	abschließen :Operation

Skript 135: Skript mit Sequenzausdrücken

[1] Siehe Abbildung 110, C).

Nach der Interaktion `a` (Sequenznummer `1.`) finden parallel die Interaktionen `b` (Sequenznummer `1.1a.`) und `c` (Sequenznummer `1.1b.`) statt. Beide Zeilen benötigen die explizite Angabe eines Vorgängers. Im Rahmen der Ausführung von `c` (Sequenznummer `1.1b.`) findet `d` (Sequenznummer `1.1b.1.`) statt. Eine Ebene höher folgen die Interaktionen `e` (Sequenznummer `1.2.`) und `f` (Sequenznummer `1.3.`). Beide Interaktionen benötigen keine Angabe eines Vorgängers, da die verwendeten laufenden Nummern auf einander folgen. Damit sind die durch `a` initiierten Aktivitäten abgeschlossen. Auf die Interaktion `a` (Sequenznummer `1.`) folgen die Interaktionen `g` (Sequenznummer `2.`) und `h` (Sequenznummer `3.`).

Die hier verwendeten Sequenzausdrücke ermöglichen die Abbildung alternativer Abläufe, ohne die Tabellenstruktur aufgeben zu müssen. Das folgende Skript demonstriert als Beispiel das Entstehen unterschiedlicher Ergebnisse bei der Bearbeitung eines Auftrags:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Auftrag	trifft ein :Action	Eintreffen des Auftrags :Interaction	:SB	nimmt Auftrag an :Operation
2.	:SB	trägt Auftrag ein :Action	Einfügen in Auftragseingang :Interaction	:Auftrags-eingang	Auftrag einfügen :Operation
3.	:Vorgang	Start :State	Vorgang erzeugt :StateChange	:Vorgang	erzeugt :State
4.	:SB	prüft auf Vollständigkeit :Action	Prüfung auf Vollständigkeit :Interaction	:Auftrag	wird auf Vollständigkeit geprüft :Operation
4. / 6.	:SB	bricht Vorgang ab :Action	Vorgang abbrechen :Interaction Guard(Auftrag nicht realisierbar)	:Vorgang	abbrechen :Operation
7.	:SB	löscht Auftrag :Action	Löschen aus Auftragseingang :Interaction	:Auftrags-eingang	löschen :Operation
8.	:Vorgang	erzeugt :State	Vorgang abgebrochen :StateChange	:Vorgang	Ende :State
4. / 10.	:SB	bittet um Vervoll-ständigung:Action	Vervollständigung erbeten :Interaction Guard(Auftragsdokumente unvollständig)	/Kunde :Person	erhält Nachricht :Operation
11.	:SB	legt Vorgang auf Wiedervorlage :Action	Auf Wiedervorlage legen :Interaction	:Kalender	Termin setzen :Operation
12.	:SB	setzt Status auf unvollständig :Action	Vorgang auf unvollständig setzen :Interaction	:Vorgang	auf unvollständig setzen :Operation
13.	:Vorgang	erzeugt :State	Vorgang zurückgestellt :StateChange	:Vorgang	Unvollständig :State
4. / 15.	:SB	setzt Status auf vollständig :Action	Vorgang auf vollständig setzen :Interaction Guard(Auftrag vollständig)	:Vorgang	auf vollständig setzen :Operation
16.	:Vorgang	erzeugt :State	Vorgang angenommen :StateChange	:Vorgang	Vollständig :State
		Eintreffen des	:EventFlow		Vorgang auf

		Auftrags :Interaction	TimingConstraint(Vorgang auf vollständig setzen. receiveTime( ) - Eintreffen des Auftrags.receiveTime( ) < = 30 Min.)		vollständig setzen :Interaction

Skript 136: Skript mit unterschiedlichen Ergebnissen

Betrachtet man den durch die Nummern festgelegten Ereignisfluss, also die durch Beziehungsknoten vom Typ *EVENTFLOW* verbundenen Modellelemente, ergibt sich beispielsweise für dieses Skript folgendes Konzeptuelles Schema:

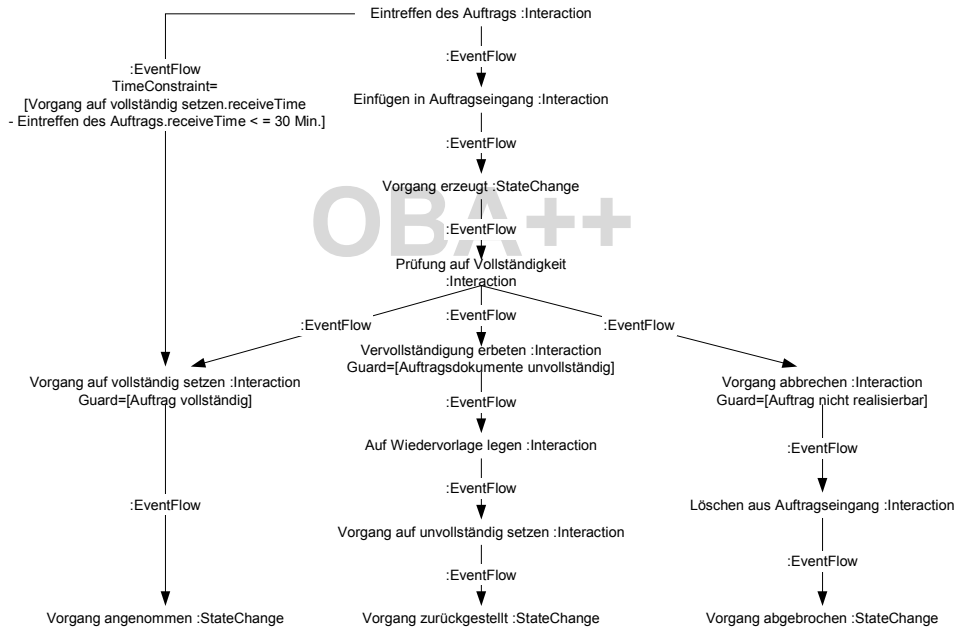


Abbildung 112: Ereignisfluss im Konzeptuellen Schema

Jede der abgebildeten Kanten stellt entweder

- eine Interaktion,
- einen Zustandsübergang oder
- den Zugriff auf die Interna eines Objekts dar.

Der Ereignisfluss kann auf diese Weise die Abfolge der im Rahmen eines Geschäftsprozesses stattfindenden Aktivitäten und zeitlichen Restriktionen darstellen. Die Knoten der Klasse *RESPONSIBILITYCOOPERATION* verbinden dabei die am Geschäftsprozess beteiligten Objekte.

Die Festlegung des Ereignisflusses auf der Basis der Abfolge von Interaktionen, Zustandsübergängen und internen Zugriffen auf Attribute bildet Abschnitte von Geschäftsprozessen auf der Grundlage der stattfindenden Interaktionen der beteiligten Objekte ab. Gleichzeitig wird durch die Notation der Skripte auch die Struktur des Systems abgebildet. Die Dokumentation zeitlicher Abhängigkeiten vervollständigt die Dokumentation der Geschäftsprozesse.

Durch die entwickelten Erweiterungen von Skriptnummern ist auch die Formulierung von Schleifen möglich:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
0. | 2. / 1.	:SB	prüft Eingangsmappe :Action	Prüfen der Eingangsmappe :Interaction	:Eingangs-mappe	Ist Vorgang vorhanden :Operation
2.	:SB	bearbeitet Vorgang :Action	Bearbeiten des Vorgangs :Interaction Guard(Vorgang vorhanden)	:Vorgang	wird bearbeitet :Operation
1. / 5.	:SB	fragt GL nach neuer Aufgabe :Action	Neue Aufgaben :Interaction	:GL	erteilt neue Aufgabe :Operation

Skript 137: Skript mit Schleife

Diesem Skript würde folgendes UML-Aktivitätendiagramm entsprechen:

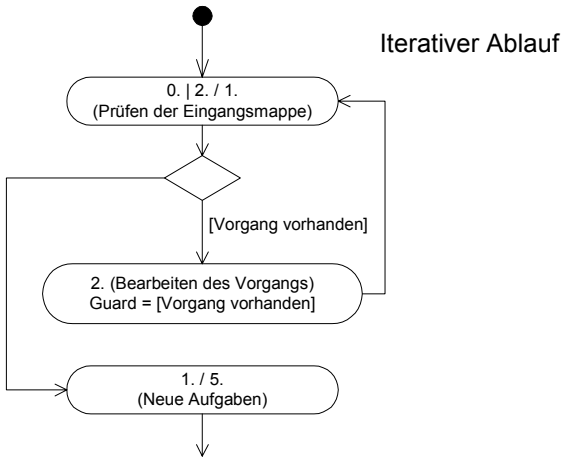


Abbildung 113: UML-Aktivitätendiagramm mit Schleife

Auch die Abbildung geschachtelter Schleifen ist auf diese Weise möglich:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
0.| 5. / 1.	:SB	Prüft, ob Aufträge zu addieren sind :Action	Prüfen Aufträge :Interaction	:Auftrags-mappe	Existenz nicht addierter Aufträge :Operation
1.| 3. / 2.	:SB	Prüft, ob Auftragspositio-nen zu addieren sind :Action	Prüfen Auftragspositionen :Interaction Guard(Auftrag zu addieren)	:Auftrags-positionen	Existenz nicht addierter Auftragspositionen :Operation
3.	:SB	Addiert Auftragsposition :Action	Addieren der Auftragspositionen :Interaction Guard(Auftragspos. zu addieren)	:Auftrags-position	Addieren :Operation
2. / 5.	:SB	Addiert Auftrag: Action	Addiert Auftrag :Interaction	:Auftrag	Addieren :Operation
1. / 7.	:SB	...	Neue Aufgaben :Interaction	...	...

Skript 138: Skript mit geschachtelter Schleife

Diesem Skript würde folgendes UML-Aktivitätendiagramm entsprechen:

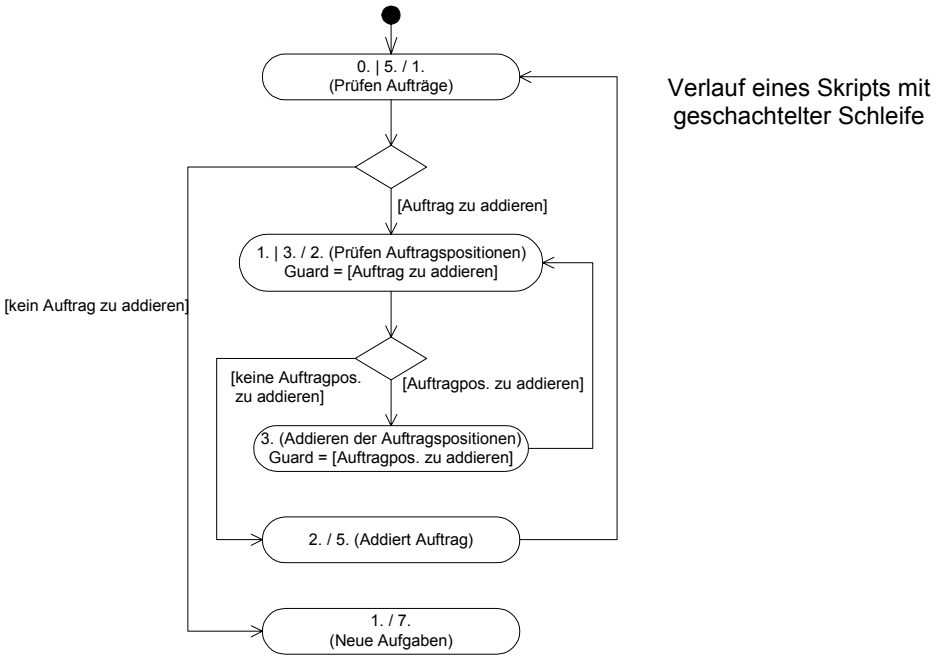


Abbildung 114: UML-Aktivitätendiagramm mit geschachtelter Schleife

6.11.4 Flacher und prozeduraler Kontrollfluss

Durch die Sequenznummern werden in der UML prozeduraler und flacher Kontrollfluss unterschieden.[1] Dabei bedeutet eine Abfolge aufsteigender Sequenznummern ohne Sequenznummernebenen einen *flachen Kontrollfluss*, geschachtelte Sequenznummern (also eine Abfolge von Sequenznummern mit Sequenznummernebenen) einen *prozeduralen Kontrollfluss*. Diese Differenzierung wird hier verwendet, um den Zusammenhang zwischen Operationen und Aktionen festzulegen. Das folgende Skript verwendet prozeduralen Kontrollfluss, das übernächste flachen Kontrollfluss:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	x:X	Akt0 :Action	:Interaktion	a:A	Op1 :Operation
1.1.	a:A	Akt1 :Action	:Interaktion	...	...
1.2	a:A	Akt2 :Action	:Interaktion	...	...
1.3	a:A	Akt3 :Action	:Interaktion	x:X	Opx :Operation
2.	x:X	Akt4 :Action	:Interaktion	...	...

Skript 139: Skript mit prozeduralem Kontrollfluss

Objekt x beauftragt in Zeile 1 Objekt a mit der Ausführung der Operation `Op1`. Im Rahmen dieser Operation führt a die Aktionen `Akt1` bis `Akt3` aus. Diese Aktivitäten sind dadurch als Inhalt bzw. als Realisierung der Operation `Op1` abgebildet. `Op1` ist durch den Abschluss der Interaktion 1.3. beendet. Die geschachtelten Interaktionen befinden sich in einer Art Klammer, die durch den Aufruf von `Op1` geöffnet wird und mit Beendigung der letzten geschachtelten Interaktion implizit geschlossen wird. Umgangssprachlich entspricht dies der Formulierung: *Im Rahmen der Ausführung der Operation Op1 übt Objekt a die Aktionen Akt1, 2, 3 und 4 aus.* Geschachtelte Interaktionen stellen Interna des Participants dar. Er (der Participant) delegiert (Teil-)Aufgaben an andere Objekte. Da in der OBA++ das Information Hiding beachtet wird, ist kein *navigierender Zugriff* möglich. Das bedeutet, dass die geschachtelten Interaktionen des Participants für den Initiator der Operation unsichtbar sind. Das bedeutet außerdem, dass eine Klasse nicht durch eine andere Klasse „hindurch“ auf die Operation einer dritten Klasse zugreifen kann.

Bei der GPM entspricht dies der Situation, dass ein Kunde davon ausgeht, dass der von ihm angesprochene Sachbearbeiter A seinen Vorgang bearbeitet. Tatsächlich hat dieser den Vorgang an einen Kollegen B zur Bearbeitung weitergereicht. Nachdem Sachbearbeiter B die Bearbeitung abgeschlossen hat, informiert er den Sachbearbeiter A über das Ergebnis der Fallbearbeitung. Sachbearbeiter A informiert den Kunden. Dieser interne Ablauf ist für den Kunden unsichtbar.

Flacher Kontrollfluss entspricht dagegen einem anderen Prinzip:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	x:X	Akt0 :Action	:Interaktion	a:A	Op1 :Operation
2.	a:A	Akt1 :Action	:Interaktion	...	...
3.	a:A	Akt2 :Action	:Interaktion	...	...
4.	a:A	Akt3 :Action	:Interaktion	x:X	Opx :Operation
5.	x:X	Akt4 :Action	:Interaktion	...	...

Skript 140: Skript mit flachem Kontrollfluss

Abgesehen von der Tabellenspalte mit den Nummern ist der Inhalt des Skripts identisch. Allerdings hat sich die Beziehung der von Objekt `a` ausgeführten Aktivitäten und der Operation `Op1` verändert. Sie stellen nicht mehr einen Bestandteil der Operation dar, sondern sind Folgeerscheinungen. Umgangssprachlich entspricht dies der Formulierung: *Nachdem Objekt a die Operation Op1 ausgeführt hat, übt es die Aktionen Akt1, 2, 3 und 4 aus.* Bei prozeduralem Kontrollfluss wird der Ablauf implizit durch die letzte Interaktion der entsprechenden Sequenznummernebene beendet. Danach wird der Ablauf auf der nächsthöheren Sequenznummernebene fortgesetzt. Aber es ist möglich, dass der Participant mit dem Initiator praktisch „zwischendurch" interagiert und anschließend weitere Aktivitäten ausführt. Nach DE CHAMPEAUX wird dieses Design als *Early Reply* bezeichnet.[2]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	x:X	AktX1 :Action	:Interaktion	a:A	Op1 :Operation
1.1.	a:A	Akt1 :Action	:Interaktion	x:X	Opx :Operation
1.2.	a:A	Akt2 :Action	:Interaktion	...	...
1.3.	a:A	Akt3 :Action	:Interaktion	...	...
1.4.	a:A	Akt4 :Action	:Interaktion	...	...
2.	x:X	AktX2 :Action	:Interaction	...	...

Skript 141: Early Reply

Die in Interaktion 1. angeforderte Operation `Op1` wird durch die Interaktionen 1.1. bis 1.4. realisiert. In Interaktion 1.1. findet aber eine Interaktion mit dem Initiator der Interaktion 1. statt. Wesentlich ist hier, dass das Objekt `a` anschließend in den Interaktionen 1.2. bis 1.4. weitere Aktivitäten ausführt. Die Operation `Op1` ist erst mit der Interaktion 1.4. abgeschlossen. Auf diese

[1] Vgl. [Booch 1999: S. 213 f., 248 f.].

[2] Vgl. [de Champeaux 1993: S. 374].

Weise kann abgebildet werden, dass der Initiator der Interaktion 1. schon weitere Aktivitäten ausführt, obwohl der Participant der Interaktion die angeforderte Operation noch nicht abgeschlossen hat.[1] Der zwischen Initiator und Participant geschlossene Vertrag muss allerdings erst erfüllt sein, wenn die Operation abgeschlossen ist. Insofern ist ein auf diese Weise gestalteter Prozess wenig zuverlässig. Erwartet eine Initiator vom Participant Zwischenergebnisse, für die ganz bestimmte Anforderungen gelten sollen, empfiehlt sich nicht die Verwendung des *Early Reply*-Prinzips. Es ist besser, die angeforderte Operation in Teilabschnitte zu zerlegen, deren Ergebnisse durch Verträge abgesichert sind.

Unter Verwendung geschachtelter Interaktionen können Schleifen auch durch Multicast- oder Broadcast-Interaktionen abgebildet werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:AL	Ermittelt Summe eingegangener Aufträge :Action	:Interaction Prop(Multicast, SB der Auftragsannahme, seq) Mode(Request) Answer(:Höhe Auftragseingang)	:SB	ermitteln der Höhe der Auftragseingänge im Regionalbereich :Operation
1.1.	:SB	ermitteln Auftragshöhe :Action	:Interaction Prop(Unicast, True, seq) Mode(Request) Answer(:Auftragseingangshöhe)	:Auftrags-eingang	ermitteln der Höhe des Auftragseingangs :Operation
1.1.1.	:Auftrags eingang	Ermittelt Summe der Aufträge :Action	:Interaction Prop(Multicast, Auftrag gehört zum jeweiligen Auftragseingang, seq) Mode(Request) Answer(:Auftragshöhe)	:Auftrag	gibt Auftragshöhe zurück:Operation
2.	:AL	übermittelt Höhe des Auftragseingangs :Action	:Interaction Mode(Supply)	:Geschäfts-führer	erhält Höhe des Auftragseingangs :Operation

Skript 142: Skript mit geschachteltem Kontrollfluss und Schleifen

Die Schleifen sind hier nicht offensichtlich, sondern sind in der Multicast-Interaktion der Zeilen `1.1.1.` und `1.` verborgen. Der Abteilungsleiter interagiert mit den Sachbearbeitern der Auftragsannahme (siehe. `Prop(Multicast, SB der Auftragsannahme, seq)` in Interaktion `1.`). Im Rahmen der von dieser Interaktion angeforderten Operation (`ermitteln der`

[1] Dadurch können natürlich Probleme der Synchronisation entstehen, auf die hier jedoch nicht eingegangen wird.

`Höhe der Auftragseingänge im Regionalbereich`) ermitteln die Sachbearbeiter durch eine Interaktion mit ihrem jeweiligen Auftragseingang die Höhe der Auftragseingänge (Interaktion `1.1.`). Im Rahmen der Operation `ermitteln der Höhe des Auftragseingangs` wird vom Objekt Auftragseingang die Summe aller Aufträge gebildet. Die Interaktion `1.1.1.` ist eine Multicast-Interaktion mit den zugehörigen Aufträgen, die jeweils ihre Auftragshöhe zurückgeben (`Answer( :Auftragshöhe)`). Nach Abschluss der Interaktion `1.1.1.` ist die Summenbildung der Aufträge abgeschlossen. Damit ist auch die Interaktion `1.1.` abgeschlossen, die die Auftragseingangshöhe zurück übermittelt. Dies geschah im Rahmen der Interaktion `1.`, die damit ebenfalls abgeschlossen ist. Der Abteilungsleiter übermittelt in Interaktion `2.` die Höhe des Auftragseingangs an den Geschäftsführer.

Prozeduraler Kontrollfluss darf nicht als Kopf und Körper einer Prozedur verstanden werden. Die Interaktionsbeziehung besteht aus einem Ereignis, welches als Ergebnis einer Aktion entsteht, und dem Feststellen eines Ereignisses, durch das eine Operation getriggert wird. Objekte produzieren Ereignisse und reagieren auf Ereignisse. Werden Interaktionen durch Sequenznummern in eine Sequenzrelation zueinander gesetzt, stellt dies immer nur eine von mehreren möglichen Abfolgen von Ereignissen dar. Inhalt der objektorientierten Modellierung ist nicht der Entwurf von Algorithmen im Sinne starrer Bearbeitungsvorschriften. Weder durch flache noch durch prozedurale Sequenznummern sollen Algorithmen vorweggenommen werden. Nur deshalb ist folgendes Skript überhaupt sinnvoll:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	...	...	:Interaction	:A	OpA :Operation
1.1.	:A	Akt1 :Action	:Interaktion	...	...
1.2.	:A	Akt2 :Action	:Interaktion	...	...
1.3.	:A	Akt3 :Action	:Interaktion	...	...
2.	...	...	:Interaction	:A	OpA :Operation
2.1.	:A	Akt4 :Action	:Interaktion	...	...
2.2	:A	Akt1 :Action	:Interaktion	...	...

Skript 143: Zweimaliger Aufruf einer Operation mit unterschiedliche Aktionen

In Skript 143 wird die Operation `OpA` des Objekts der Klasse `A` in der Interaktion 1. und 2. aufgerufen. Würde man die geschachtelt stattfindenden Interaktionen als starre Bearbeitungsvorschrift auffassen, wäre die Interpretation des Skripts problematisch, da die initiierten Abläufe voneinander abweichen. Die Skripte stellen in verlaufsorientierter Form die eingehenden und ausgehenden Ereignisse dar, auf die die Objekte einer Klasse reagieren, und mögliche Sequenzrelationen, in denen diese Ereignisse auftreten können. Die dokumentierten Sequenzrelationen sollten nicht als die einzigen Möglichkeiten aufgefasst werden, auf ein Ereignis zu reagieren.

Prinzipiell besteht in flexibel reagierenden und agierenden Systemen immer die Möglichkeit, dass auf ein und das gleiche Ereignis unterschiedlich reagiert wird. Diesem Phänomen wird so Rechnung getragen.

6.11.5 Zentrale und dezentrale Steuerung

Ein Problem der OBA war, wie man in einem Skript modelliert, dass als Reaktion auf ein Ereignis mehrere Aktivitäten in Folge ausgeführt werden. Ein typisches Anwendungsbeispiel ist, wenn ein Sachbearbeiter mit einem Anwendungssystem zur Auftragsbearbeitung eine Rechnung erstellt, einen offenen Posten erzeugt und die Rechnung anschließend ohne weitere Benutzerinteraktion vom System ausgedruckt wird. Eine Modellierung in der folgenden Form ist weder in der OBA, noch hier möglich, da in der Zeile 2 und 3 das auslösende Ereignis fehlt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Rechnung :Action	:Interaction	:Auftragsbearbeitung	Rechnung erstellen :Operation
2.				:Offene-Posten-Buchhaltung	Offenen Posten erzeugen :Operation
3.				:Auftragsbearbeitung	Rechnung drucken :Operation

Skript 144: Unzulässige Modellierung einer Sequenz

Das folgende Skript ist syntaktisch korrekt und drückt auch den Sachverhalt aus, dass der Sachbearbeiter zunächst eine Rechnung erstellt, einen offenen Posten erzeugt und dann durch eine weitere Aktion die Rechnung ausgedruckt wird. Der sequentielle Ablauf, der vom System ohne weitere Einwirkung des Anwenders ausgeführt werden soll, ist aber auch hier nicht modelliert worden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Rechnung :Action	A :Interaction	:Auftragsbearbeitung	Rechnung erstellen :Operation
2.	:SB	erzeugt offenen Posten :Action	B :Interaction	:Offene-Posten-Buchhaltung	Offenen Posten erzeugen :Operation
3.	:SB	druckt Rechnung :Action	C :Interaction	:Auftragsbearbeitung	Rechnung drucken :Operation

Skript 145: Zulässige Modellierung einer Sequenz

Zur Lösung dieses Problems gibt es zwei Möglichkeiten.[1] Die erste besteht darin, die Operationen der Auftragsbearbeitung aufzuspalten und untereinander zu verketten:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Rechnung :Action	:Interaction	:Auftragsbearbeitung	Rechnungslauf :Operation
1.1.	:Auftrags-bearbeitung	Rechnung soll erstellt werden :Action	:Interaction	:Auftragsbearbeitung	Rechnung erstellen :Operation
1.1.1.	:Auftrags-bearbeitung	offener Posten soll erzeugt werden :Action	:Interaction	:Offene-Posten-Buchhaltung	:Offenen Posten erzeugen :Operation
1.1.1.1.	:Offene-Posten Buchhaltung	Rechnung soll gedruckt werden :Action	:Interaction	:Auftragsbearbeitung	Rechnung drucken :Operation

Skript 146: Erste Version der Modellierung einer Sequenz

Innerhalb der Operation `Rechnung erstellen` wird in Interaktion 1.1.1. der offene Posten erzeugt. Die Offene-Posten-Buchhaltung fordert schließlich den Ausdruck der Rechnung an. Diese Modellierung bildet den gewünschten Ablauf ab, hat allerdings den Nachteil, dass die einzelnen Operationen stark miteinander verbunden sind. Die Operation `Rechnung drucken` ist so als fester Bestandteil der Operation `Offenen Posten erzeugen` modelliert worden. Dies hat den Vorteil, dass jeder Participant im Rahmen der von ihm ausgeführten Operation *entscheidet*, welche Operation als nächste ausgeführt wird. Übertragen auf die GPM bedeutet dies, dass beispielsweise jeder Sachbearbeiter, der zur Ausführung einer Operation aufgefordert wird, entscheidet, welche Operation als nächste ausgeführt wird. In diesem Fall kann von einer verteilten Steuerung des Ablaufs gesprochen werden, da es keine zentrale Stelle gibt, die den Ablauf steuert.[2] Diese Lösung hat allerdings den Nachteil, dass die Reihenfolge eines Prozesses in den Operationen *hart verdrahtet* ist, weil jede Operation ihre Folgeoperation aufruft.

Die andere Lösung geht von einer zentralen Steuerung des Ablaufs aus. Bei dieser Variante steuert ein Objekt den Ablauf der auszuführenden Aktivitäten.

[1] Dieser Ansatz wurde inspiriert von BOCK, der ein Konzept zur objektorientierten Umsetzung von Fluss- bzw. Aktivitätendiagrammen vorgestellt hat. Vgl. [Bock 1999].

[2] Dieses Prinzip entspricht dem von FERSTL und SINZ favorisierten Ansatz zur Gestaltung verteilter Geschäftsprozesse und Anwendungssysteme. Vgl. [Ferstl 1994c]. In diesem Ansatz existiert in einem verteilten System keine Komponente, welche die globale Kontrolle über das gesamte verteilte System besitzt. Vgl. [Ferstl 1994: S. 13].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Rechnung :Action	:Interaction	:Vorgangsbearbeitung	Rechnungslauf :Operation
1.1.	:Vorgangs-bearbeitung	Rechnung soll erstellt werden :Action	:Interaction	:Auftragsbearbeitung	Rechnung erstellen :Operation
1.2.	:Vorgangs-bearbeitung	offener Posten soll erzeugt werden :Action	:Interaction	:Offene-Posten-Buchhaltung	:Offenen Posten erzeugen :Operation
1.3.	:Vorgangs-bearbeitung	Rechnung soll gedruckt werden :Action	:Interaction	:Auftragsbearbeitung	Rechnung drucken :Operation

Skript 147: Zweite Version der Modellierung einer Sequenz

Der Unterschied besteht nun darin, dass das Objekt der Klasse `Vorgangsbearbeitung` zentral den Ablauf der Rechnungserstellung steuert und keine der darin ausgeführten Operationen die Ausführung ihrer Folgeoperation direkt anfordert.[1] Bezogen auf die Operation `Rechnung erstellen` bedeutet das, dass nicht diese die Operation `Offenen Posten erzeugen` anfordert, sondern nach Abschluss der Operation die Kontrolle an das Vorgangsbearbeitungsobjekt zurückgegeben wird. Übertragen auf die GPM bedeutet dies, dass ein Sachbearbeiter, der zur Ausführung einer Operation aufgefordert wird, *nicht entscheidet*, welche Operation als nächste ausgeführt wird, sondern dies von einer zentralen Stelle entschieden wird. In diesem Fall ist von einer zentralen Steuerung des Ablaufs zu sprechen, die für die Synchronisation der Aktivitäten sorgt. Diese Synchronisation wird durch dezidierte Objekte übernommen, die entscheiden, welches Objekt wann eine Aktivität auszuführen hat.[2] Diese Lösung hat den Nachteil, dass eine zentrale Steuerung in Form eines Objekts notwendig ist. Sie hat aber den Vorteil, dass die Definition eines betrieblichen Ablaufs, die sich in der Praxis erfahrungsgemäß recht häufig ändert, nur an einer zentralen Stelle gepflegt und verändert werden muss. Außerdem ist die Reihenfolge der Aktivitäten nicht mehr in den Operationen zu finden, da diese keine Folgeoperationen mehr aufrufen. Dies führt erfahrungsgemäß zu einem flexibleren System.

6.11.6 Ereignisfluss

Zeitliche Abhängigkeiten zwischen den Zeilen eines Skripts werden durch den Beziehungstyp *EVENTFLOW* abgebildet.[3] Für *EVENTFLOW* ist das Prädikat `TimingConstraint(`*`Time`*

[1] In der Methode SOM entspräche dies einem Regelstreckenobjekt, welches eine Reihe von Reglerobjekten steuert. Vgl. 6.2.4.

[2] Vgl. hierzu auch die Synchronisation durch Modellprimitive in Abschnitt 6.2.6.

[3] Ein ähnliches Konstrukt findet sich auch in der Methode OSA. Vgl. [Embley 1992: S. 188]. Das hier entwickelte Prinzip entspricht jedoch nicht den von BOOCH skizzierten Zeitbudgets (vgl. [Booch 1994: S. 271]), weil hier die Bezugspunkte der zeitlichen Einschränkung beliebige Interaktionen sind.

Expression) definiert, um zeitliche Bedingungen auszudrücken. Dies entspricht der Verwendung in Interaktionen mit dem Unterschied, dass hier nicht zeitliche Abhängigkeiten der Aktivitäten und der Interaktion einer Zeile eines Skripts ausgedrückt werden, sondern zeitliche Beziehungen zwischen zwei unterschiedlichen Zeilen eines Skripts.

Im folgenden Skript wird ausgedrückt, dass von dem Moment, in dem die Post das Angebot erhalten hat, bis zu dem Moment, in dem der Kunde das Angebot erhalten hat, maximal drei Tage vergangen sind. Die Formulierung `Eintreffen Angebot.receiveTime()` bedeutet, dass von der Interaktion mit dem Namen `Eintreffen Angebot` der Inhalt der Tabellenspalte Connection, also die Interaktion selbst referenziert wird.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Verkauf	erstellt Angebot :Action	Angebotserstellung :Interaction	:Angebot	wird erstellt :Operation
	:Verkauf	versendet Angebot :Action	Angebotsversand :Interaction Content(:Angebot)	:Post	transportiert Angebot :Operation
	:Post	stellt Angebot zu :Action	Eintreffen Angebot :Interaction Content(:Angebot)	/Kunde :Person	Annehmen Angebot :Operation
		Angebotsversand :Interaction	:EventFlow TimingConstraint(Eintreffen Angebot.receiveTime() – Angebotsversand.receiveTime() ≤ 3 Tage)		Eintreffen Angebot :Interaction

Skript 148: Zeitverbrauch eines Transportvorgangs

Dieses Skript verdeutlicht, warum die Verwendung eines Namens für die Interaktion zwar in der UML nicht vorgesehen ist, hier aber keinesfalls als redundant betrachtet werden kann. Ohne einen Bezeichner für die Interaktionen wäre die Formulierung solcher zeitlichen Beziehungen nicht möglich. Die UML muss an dieser Stelle die Hilfskonstruktion der *Ereignismarken* verwenden, um innerhalb eines Diagramms eine Interaktion zwischen zwei Objekten referenzieren zu können.[1] Zusammen mit den Zeitrestriktionen für Interaktionen (Abschnitt 6.2.14) lassen sich hier dagegen mit Hilfe des Prädikats `TimingConstraint()` Aussagen über das zeitliche Verhalten folgender Art formulieren:

- Wie lange eine Aktivität oder Interaktion dauert.
- In welchem (maximalen/minimalen) zeitlichen Abstand innerhalb einer Interaktion Aktion und Operation stattfinden.
- In welchem (maximalen/minimalen) zeitlichen Abstand zwei Interaktionen stattfinden.

Zu beachten ist dabei, dass durch dieses Prädikat nur zeitliche Restriktionen in Interaktionen formuliert werden. Wann ein durch eine Aktion spezifiziertes Ereignis oder wann ein Zustandswechsel eintritt, wird – wie in der UML festgelegt – durch die Attribute `when()` und `after( )` spezifiziert.[2]

Das folgende Beispiel demonstriert, wie durch den Ereignisfluss zeitliche Anforderungen an das Eintreffen von Ereignissen formuliert werden können. Ein weit verbreitetes Konstrukt zur GPM ist in den Ereignisgesteuerten Prozessketten der Methode ARIS das Ausführen einer Funktion abhängig vom Eintreten eines oder mehrerer Ereignisse.[3] Eine Funktion wird dann ausgeführt,

- wenn alle aufgeführten Ereignisse eintreten (konjunkte Ereignisse),
- wenn mindestens eines der aufgeführten Ereignisse eingetreten ist (adjunkte Ereignisse),
- wenn exakt eines der aufgeführten Ereignisse eintritt (disjunkte Ereignisse).

Dabei wird übersehen, dass diese Modellierung von Funktionen, die abhängig von mehreren Ereignissen sind, eigentlich zwei Aspekte beinhaltet:

- den Aspekt der Synchronisation einer Aktivität mit dem Eintreffen eines oder mehrerer Ereignisse;
- den Aspekt der zeitlichen Abhängigkeiten bzw. des zeitlichen Abstands zwischen den Ereignissen. Dieser Aspekt wird völlig ignoriert. Deshalb ist die Definition von Zeitintervallen sinnvoll, innerhalb derer zwei Ereignisse als „zeitgleich“ eingetreten angesehen werden.

Der Aspekt der Synchronisation wird durch Zeilennummern abgebildet. Exemplarisch soll dies für konjunkte Ereignisse demonstriert werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Ereignis1	tritt ein :Action	E1 :Interaction	:Klasse1	stellt Ereignis1 fest :Operation
2.	:Ereignis2	tritt ein :Action	E2 :Interaction	:Klasse1	stellt Ereignis2 fest :Operation

[1] Vgl. [Douglas 1999: S. 229, 343] und Abschnitt 5.3.4.

[2] Vgl. Abschnitt 6.8.

[3] Vgl. [Keller 1992], [Scheer 1995: S. 50 ff.], [Scheer 1998: S. 124 ff.], [Keller 1998], [Staud 1999: S. 51 ff.], [Becker 2000: S. 59 ff.]. Dieses Prinzip findet auch in den objektorientierten Erweiterungen der EPK Verwendung. Vgl. [Nüttgens 1998], [ZimmermannV 1999].

1.& 2. / 3.	:Klasse1	handelt :Action	:Interaction	...	
		E1 :Interaction	:EventFlow TimingConstraint(|(E1.receiveTime() – E2.receiveTime())| ≤ 5 s)[1]		E2 :Interaction

Skript 149: Modellierung zusammengesetzter Ereignisse[2]

Durch die Angabe des Vorgängerausdrucks wird festgelegt, dass die Zeile mit der Sequenznummer `3.` nur dann eintritt, wenn sowohl das Ereignis in Zeile `1.` wie auch das in der Zeile `2.` eingetreten ist. Disjunkte oder adjunkte Ereignisse werden entsprechend abgebildet, wobei nur der Vorgängerausdruck zu modifizieren ist.

Zusätzlich wird über die Angabe von Zeitrestriktionen festgelegt, innerhalb welchen Zeitraums die Ereignisse eintreten müssen, damit sie als parallel angesehen werden. Hier soll gelten, dass die Interaktion (also das Eintreten des Ereignisses) `E1` und die Interaktion `E2` innerhalb von fünf Sekunden eintreten müssen. Die Zeitrestriktion basiert auf dem absoluten Betrag[3] der Differenz der beiden Zeitpunkte, in denen das Empfängerobjekt die Nachricht erhält, dass das `Ereignis1` respektive `Ereignis2` eingetreten ist.

Das folgende Beispiel demonstriert, wie die Initiierung paralleler Prozesse durch Sequenzausdrücke modelliert wird. Die Projektabwicklung informiert zeitgleich die Abteilung Technik und die Abteilung Entwicklung über ein neues Projekt, worauf diese parallel mit der Arbeit beginnen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Projektab-wicklung	informiert :Action	Übergabe Technik :Interaction Content (Projektdaten)	:Technik	wird über Projekt informiert :Operation
2.	:Projektab-wicklung	informiert :Action	Übergabe Entwicklung :Interaction Content (Projektdaten)	:Entwicklung	wird über Projekt informiert :Operation
2 / A3.	:Technik	montiert Rechner :Action	:Interaction	:Rechner	wird montiert :Operation
2 / B3.	:Entwicklung	entwickelt Softwaresystem :Action	:Interaction	:Software-system	wird programmiert :Operation

[1] Durch den Operator „||“ wird der absolute Betrag eines Ausdrucks ermittelt.

[2] E1 und E2 sind willkürlich gewählte Namen.

[3] Vgl. [Opitz 1997: S. 7].

	Übergabe Technik :Interaction	:EventFlow TimingConstraint(Übergabe Technik.sendTime = Übergabe Entwicklung.sendTime)		Übergabe Entwicklung :Interaction

Skript 150: Skript mit zusätzlicher Zeitangabe

Durch die letzte Zeile dieses Skripts wird zusätzlich festgelegt, welchen zeitlichen Bezug die beiden parallel verlaufenden Vorgänge besitzen. Die Übergabe der Projektdaten geschieht an beide Abteilungen gleichzeitig. Auf diese Weise wird modelliert, wie unabhängig verlaufende Prozesse initiiert werden.

6.11.7 Zusätzliche Kontrollstrukturen

In Abschnitt 6.6.5 wurde festgestellt, dass im objektorientierten Ansatz Sequenzen, d. h. starre Abfolgen, einen erheblich geringeren Stellenwert besitzen, als dies z. B. bei Modellierungsverfahren, die Flussdiagrammen ähneln, der Fall ist. Bei der Modellierung treten aber auch Situationen auf, bei denen bestimmte Reihenfolgebeziehungen gegeben sind oder bei denen Aktivitäten nur optional ausgeführt werden. Als Beispiele seien genannt:

- Bei der Erstellung eines Dokuments mit Hilfe eines Textverarbeitungssystems ist irgendwann – spätestens aber vor dem Ausdruck – die Anschrift halbfett zu formatieren.
- Ein Sachbearbeiter hat an einem Arbeitstag eine Reihe von Aufgaben zu erfüllen, die Reihenfolge der Bearbeitung der Aufgaben ist ihm freigestellt.
- Die Ausführung einer bestimmten Aktivität ist optional.

Als optionale Interaktionen werden solche bezeichnet, die innerhalb einer sequentiellen Abfolge stattfinden sollen, deren Ausführung aber von einer Bedingung abhängt. Eine Möglichkeit besteht darin, dass man diese Bedingung durch die Überwachungsbedingung `Guard()` abbildet. Im folgenden Skript prüft ein Sachbearbeiter das Ergebnis seiner Arbeit, indem er optional einen Probeausdruck anfertigt. Durch die Überwachungsbedingung wird festgelegt, dass die Interaktion nur dann stattfindet, wenn eine zusätzliche Prüfung des Arbeitsergebnisses gewünscht ist.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Dokument :Action	:Interaction	:Textverarbeitung	erstellt Dokument :Operation
2.	:SB	Probeausdruck anfertigen :Action	:Interaction Guard(zusätzliche Prüfung gewünschtl)	:Textverabeitung	Dokument ausdrucken :Operation
3.	:SB	Dokument übermitteln :Action	:Interaction	:Belichtungseinheit	Dokument entgegennehmen :Operation

Skript 151: Eine optionale Interaktion

Soll eine Menge von mehreren Interaktionen in beliebiger Reihenfolge stattfinden, muss eine andere Modellierung verwendet werden. Im folgenden Beispiel ist die Reihenfolge der Interaktionen b, c, d unerheblich. Sie müssen aber alle ausgeführt werden, bevor e ausgeführt werden kann.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Text :Action	a :Interaction	:Textverarb.	erstellt Dokument :Operation
1. / 3.	:SB	fügt Textauszeichnungen ein :Action	b :Interaction	:Textverarb.	formatiert Dokument :Operation
1. / 5.	:SB	erstellt Grafiken :Action	c :Interaction	:Textverarb.	Grafiken einfügen :Operation
1. / 7.	:SB	erstellt Seitenumbruch :Action	d :Interaction	:Textverarb.	Seitenumbrüche einfügen :Operation
3.&5.&7./ 9.	:SB	druckt Text :Operation	e :Interaction	:Textverarb.	druckt Dokument :Operation

Skript 152: Interaktionen ohne festgelegte Reihenfolge

Durch die Vorgängerausdrücke in den Zeilen 3., 5. und 7. wird festgelegt, dass zuerst in der Interaktion 1. ein Text erstellt werden muss, damit diese Interaktionen stattfinden können. Die Interaktionen werden durch die abschließende Interaktion mit der Nummer 9. synchronisiert.

Es ist zu beachten, dass zwischen diesen Interaktionen keine Reihenfolgebeziehung festgelegt wurde, da die Sequenznummern nicht direkt aufeinander folgen. Sie werden aber auch nicht parallel ausgeführt, da dies durch ein Postfix abgebildet würde. Durch die Synchronisations-bedingung im Vorgängerausdruck der Interaktion 9. wird festgelegt, dass alle drei Interaktionen stattgefunden haben müssen, damit der Text ausgedruckt werden kann.

In ähnlicher Weise können auch optionale Interaktionen über Vorgängerausdrücke modelliert werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Text :Action	:Interaction	:Textverarb.	erstellt Dokument :Operation
1. / 3.	:SB	fügt Textauszeichnungen ein :Action	:Interaction	:Textverarb.	formatiert Dokument :Operation
1. / 5.	:SB	erstellt Grafiken :Action	:Interaction	:Textverarb.	Grafiken einfügen :Operation
1. / 7.	:SB	erstellt Seitenumbruch :Action	:Interaction	:Textverarb.	Seitenumbrüche einfügen :Operation
1.^(3.|5.|7.) / 9.	:SB	druckt Text :Operation	:Interaction	:Textverarb.	druckt Dokument :Operation

Skript 153: Interaktionen ohne festgelegte Reihenfolge, von denen keine ausgeführt werden muss

Nur der Vorgängerausdruck der Interaktion 9. ist in Skript 153 verändert worden. Der Sachbearbeiter kann nun entweder den Text ausdrucken, nachdem er ihn erstellt hat, oder er kann zusätzlich eine oder mehrere der Interaktionen 3., 5. und 7. ausführen. Auch hier wieder können diese aber erst ausgeführt werden, nachdem der Text erstellt wurde.

Die dritte Möglichkeit besteht darin, dass genau eine Interaktion ausgeführt werden soll:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:SB	erstellt Text :Action	:Interaction	:Textverarb.	erstellt Dokument :Operation
1. / 3.	:SB	fügt Textauszeichnungen ein :Action	:Interaction	:Textverarb.	formatiert Dokument :Operation
1. / 5.	:SB	erstellt Grafiken :Action	:Interaction	:Textverarb.	Grafiken einfügen :Operation
1. / 7.	:SB	erstellt Seitenumbruch :Action	:Interaction	:Textverarb.	Seitenumbrüche einfügen :Operation
3.^5.^7. / 9.	:SB	druckt Text :Operation	:Interaction	:Textverarb.	druckt Dokument :Operation

Skript 154: Interaktionen ohne Reihenfolge, von denen genau eine ausgeführt werden muss

Auch hier muss zuerst der Text erstellt worden sein, damit die Interaktionen 3., 5. oder 7. stattfinden können. Welche ausgeführt wird, könnte zusätzlich über eine Überwachungsbedingung dokumentiert werden. Die Interaktion 9. soll aber nur dann ausgeführt werden, wenn exakt eine der drei Interaktionen stattgefunden hat. Mit diesem Beispiel sei die Darstellung der Möglichkeiten von Zeilennummern und des Ereignisflusses abgeschlossen.

6.12 Strukturierung von Skripten

6.12.1 Konzepte zur Strukturierung in anderen objektorientierten Ansätzen

Ein Mangel der UML ist, dass in ihrem Metamodell zwischen Interaktionsdiagrammen (also Sequenz- und Kollaborationsdiagrammen) keine Beziehungstypen definiert sind.[1] Dadurch entstehen bei der Modellierung komplexer Zusammenhänge mindestens zwei Alternativen. Entweder man versucht, den gesamten Prozess in einem Diagramm darzustellen, oder man zerlegt ihn in irgendeiner Weise auf mehrere Diagramme.[2] Die erste Möglichkeit ist bei Problemstellungen mit großer Komplexität nicht praktikabel. Sequenz- oder Kollaborationsdiagramme mit einer großen Anzahl von beteiligten Objekten sind vielleicht noch mit entsprechender Werkzeugunterstützung zu erstellen, aber für den menschlichen Betrachter aufgrund ihrer Komplexität nicht überschaubar. Eine eindeutige Grenze ist zwar nicht zu ziehen, eine realistische Annahme dürfte aber wohl sein, dass ein Sequenzdiagramm mit mehr als zehn und ein Kollaborationsdiagramm mit mehr als 20 Objekten kaum noch überschaubar ist.[3]

Wählt man die zweite Möglichkeit, kann man das Problem so lange zerlegen, bis man einen Zerlegungsgrad erhalten hat, in dem die einzelnen Diagramme eine überschaubare Komplexität besitzen. Aber in diesem Fall muss der Zusammenhang zwischen den einzelnen Diagrammen definiert werden. Die Zusammenhänge des Gesamtprozesses existieren sonst nur noch im Kopf des Modellierers, was den Wert des Modells erheblich reduziert. Doch nur wenige objektorientierte Modellierungsansätze beinhalten Konstrukte für Beziehungen zwischen Interaktionsdiagrammen:

---

[1] Es ist ein generelles Problem der UML, dass im Metamodell keine Beziehungen zwischen Diagrammen und Modellen definiert sind. Aus diesem Grund muss man die implizit verwendeten Beziehungen aus der Standarddokumentation rekonstruieren. Zu den wenigen Veröffentlichungen, die sich mit dieser Problematik auseinandersetzen, gehören [Henderson-Sellers 1999] und [Linssen 1999].

[2] In der UML wird eine andere Vorgehensweise favorisiert. Die Funktionalität eines Systems soll durch Use Cases dokumentiert werden. Diese Use Cases stellen textorientierte Beschreibungen der Systembenutzung durch externe Objekte dar. Jeder Use Case wird durch eine Anzahl von Szenarien in Form von Interaktionsdiagrammen modelliert. Vgl. [Linssen 1999]. Diese Vorgehensweise wird hier nicht weiter betrachtet, weil Use Cases nicht objektorientiert sind und kein Zusammenhang zwischen den Interaktionsdiagrammen hergestellt wird. Erstaunlicherweise wird – trotz aller Kritik an den Use Cases der UML – auch in der OML der Zusammenhang zwischen Interaktionsdiagrammen über das Use-Case-Modell erklärt (vgl. [Henderson-Sellers 1998: Appendix E, Stichwort „Scenario, task scripts and use cases“], [Linssen 1999b]).

[3] Nach MILLER ist die Grenze sogar noch erheblich geringer anzusetzen, nämlich bei maximal neun Objekten. Vgl. [Miller 1956].

- In der OML werden alle Diagrammtypen, die die Interaktion zwischen Objekten ausdrücken, als *Szenarien* bezeichnet.[1] Es sind zwei Beziehungstypen definiert, um Szenarien miteinander in Beziehung zu setzen. Der Beziehungstyp *invokes* verbindet zwei Szenarien in der Art, dass in einem Szenario an einer definierten Stelle ein anderes Szenario aufgerufen wird. Der Beziehungstyp *precedes* verbindet zwei Szenarien in der Art, dass ein Szenario der Vorgänger eines anderen Szenarios ist.[2]
- In der UML werden in der Regel keine Beziehungen zwischen Interaktionsdiagrammen verwendet. Ein oder mehrere Interaktionsdiagramme spezifizieren oder beschreiben einen Use Case. Die Use Cases werden untereinander in Verwendet- („*include*“) oder Erweitert-Beziehungen („*extend*“) gesetzt.[3] Zwischen Kollaborationsdiagrammen ist eine Realisierungsbeziehung definiert. Ein grobkörniges („*coarse-grained* “) Kollaborationsdiagramm wird durch ein feinkörnigeres präzisiert, in dem eine Operation in ein Kollaborationsdiagramm expandiert wird.[4] Für Sequenzdiagramme existiert diese Möglichkeit nicht. Auch eine Reihenfolgebeziehung existiert für beide Diagrammtypen nicht. Das in der UML definierte Konstrukt *Paket* ist für dieses Problem nicht geeignet, da Pakete nur Container für Modelle[5] darstellen und zwischen Paketen keine Reihenfolgebeziehungen existieren.
- In der Methode CATALYSIS wird der Begriff *Aktion* (im Original: *Action*) verwendet, um von den einzelnen Schritten einer Abfolge von Objektinteraktionen zu abstrahieren.[6] Eine Aktion stellt den Vorgang dar, der von einem Ausgangszustand zu einem Zielzustand führt. Vererbungsbeziehungen werden verwendet, um Aktionen zu verbinden, die die gleiche Nachbedingung erfüllen. Aktionen werden durch Aggregationsbeziehungen funktional zerlegt.[7] Die Aggregationsbeziehung beinhaltet aber keine Aussage über die Anordnungsfolge der Aktionen. Es wird eine Abstraktionsbeziehung verwendet, um in Sequenzdiagrammen eine einzelne Aktion durch ein weiteres Sequenzdiagramm zu verfeinern.[8]
- Obwohl sich die Arbeit von BEHRINGER mit der Modellierung globalen Systemverhaltens durch Szenarien in der objektorientierten Analyse befasst, existiert keine Makrostruktur für die

---

[1] Vgl. [Firesmith 1997: S. 173].

[2] Der dritte Beziehungstyp *uses* ist hier nicht von Interesse, da er die Beziehung zwischen einem externen Benutzer und einem Szenario ausdrückt.

[3] Vgl. [Rumbaugh 1999: S. 272 ff., 297 ff.].

[4] Vgl. [Rumbaugh 1999: S. 197].

[5] Das im Metamodell der UML definierte Modellelement *Model* ist eine Spezialisierung von Paket und bietet keine zusätzlichen Modellierungsmöglichkeiten.

[6] [D'Souza 1998: S. 4-183], [D'Souza 1999: S. 7, 15].

[7] [D'Souza 1998: S. 4-186].

[8] Vgl. [D'Souza 1999: S. 224].

verwendeten Interaktionsdiagramme. Eine *Detaillierungsbeziehung* wird verwendet, um Interaktionen in „elementarere“ Interaktionen zu zerlegen.[1] Diese wird aber nicht verwendet, um Beziehungen zwischen Interaktionsdiagrammen zu knüpfen.

- In der Methode OBA existieren zwei Arten, Beziehungen zwischen Skripten zu definieren. Sie können in einer Reihenfolgebeziehung untereinander stehen. Außerdem kann eine Operation eines Participants durch ein weiteres Skript verfeinert werden.[2] Allerdings wird weder die Semantik noch die Anwendung dieser Verfeinerung erläutert.
- In der Methode SELECT von ALLEN und FROST werden Operationen eines Sequenzdiagramms durch ein weiteres Sequenzdiagramm verfeinert.[3] In einem Sequenzdiagramm kann ein anderes Sequenzdiagramm als Erweiterung oder Ablaufalternative eingebunden werden.[4]

Um komplexe Modellierungssituationen handhaben zu können, wird in den folgenden Abschnitten eine Makrostruktur für Skripte entwickelt. Diese Makrostruktur beinhaltet zwei Arten von Beziehungen zwischen Skripten. Dies sind Abstraktionsbeziehungen und Ereignisflussbeziehungen, die hier zur Modellierung von Reihenfolgen verwendet werden. Abstraktionsbeziehungen werden im nächsten Abschnitt entwickelt, Ereignisflussbeziehungen im übernächsten Abschnitt. Die grafische Darstellung der Beziehungen zwischen den Skripten wird als nebensächlich betrachtet und hier nicht erörtert.

6.12.2 Verfeinerung und Realisierung durch Subskripte

Durch die Definition von Beziehungen zwischen Skripten ist man in der Lage, bei der Modellierung komplexer Prozesse die einzelnen Skripte in einem überschaubaren Umfang zu halten. Die Komplexität der einzelnen Skripte bleibt beherrschbar, während der Gesamtzusammenhang zwischen den Skripten durch Beziehungen festgehalten wird. Diese Beziehungen dienen der Strukturierung komplexer Prozesse durch Skripte. Auf dieser Betrachtungsebene werden Skripte als „elementare“ Bausteine und *blackboxes* angesehen.[5]

Ausgangspunkt sind geschachtelte Interaktionen. Durch prozeduralen Kontrollfluss wird in Skripten der Umstand ausgedrückt, dass eine Operation durch eine Abfolge von Interaktionen

---

[1] Vgl. [Behringer 1997: S. 129 ff.].

[2] Vgl. [Rubin 1992], [Rubin 1994b].

[3] Vgl. [Allen 1998: S. 133 ff.].

[4] Vgl. [Allen 1998: S. 127].

[5] Dies entspricht der ursprünglichen Konzeption der OBA: „*Scripts are groupings of activities that can be viewed as singular events.*“ [Rubin 1992: S. 61].

realisiert wird.[1] Eine Operation ist abgeschlossen, wenn ihre geschachtelten Interaktionen abgeschlossen sind. Dies entspricht dem in der UML definierten Zusammenhang zwischen einer Operation und einer Kollaboration bzw. Interaktion: Dort realisiert eine Kollaboration oder Interaktion eine Operation.[2] Dieses Prinzip ist sowohl in der UML wie auch hier rekursiv anwendbar, wenn eine im Rahmen der Realisation ausgeführte Operation selbst durch weitere Interaktionen realisiert wird:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	...	...	:Interaction	...	X :Operation
1.1.	...	...	Realisierung von X :Interaction	...	XX :Operation
1.2.	...	...	Realisierung von X :Interaction	...	XY :Operation
1.3.	...	...	Realisierung von X :Interaction	...	XZ :Operation
2.	...	...	:Interaction	...	Y :Operation

Skript 155: Geschachtelte Realisierung von Operationen

Die Zeilen 1.1., 1.2. und 1.3. stellen die im Rahmen der Operation X in der Zeile 1. stattfindenden Interaktionen dar. Diese Zeilen kann man aber auch als separates Skript formulieren. Verwendet man die Terminologie der UML, dann besteht zwischen Operationen und Skripten eine Abhängigkeitsbeziehung, durch die eine Veränderung der Abstraktionsebene abgebildet wird.[3] Das Subskript beschreibt nun entweder

1. die Interaktionen, die im Rahmen der Operation ausgeführt werden,
2. die durch die Operation bewirkte Zustandsveränderungen von Objekten,
3. die im Rahmen einer Operation stattfindenden Attributveränderungen oder
4. die für die Operation notwendigen statischen Beziehungen.

Der Aufbau und Inhalt eines Skripts in der Rolle des Subskripts weicht in keiner Weise von den bisher verwendeten Skripten ab. An der Realisierung einer Operation können eine Vielzahl von Objekten beteiligt sein, die Interaktionsbeziehungen eingehen, um die durch die Operation spezifizierte Aufgabe zu erfüllen.[4] Die Operation stellt dann implizit ein Ereignis dar, auf welches in dem Subskript reagiert wird. Dies entspricht einer dynamischen Kollaboration in der UML. Zur Realisierung der Operation bestehen zwischen Objekten statische Beziehungen, was einer

[1] Vgl. Abschnitt 6.11.4.

[2] Vgl. [Rumbaugh 1999: S. 39, 196], [Booch 1999: S. 371, 375, 378].

[3] Vgl. [Rumbaugh 1999: S. 118 f., 250 f.]. So auch in der Methode CATALYSIS zu finden. Vgl. [D'Souza 1999: S. 213 ff.].

[4] Dies wurde schon als *Delegation* bezeichnet.

statischen Kollaboration in der UML entspricht. Jedes der an der Realisierung einer Operation beteiligten Objekte kann Zustandsveränderungen vollziehen. Dies entspricht der Realisierung einer Operation durch ein Statechart in der UML.[1]

Hierzu folgende Beispiele, wobei zuerst die Realisierung dargestellt wird. In einem Skript findet sich die folgende Operation:

```
Auftrag :Class
anlegen Auftragsposition :Operation
```

Diese Operation wird in einem anderen Skript durch eine statische Kollaboration in Form einer referentiellen Beziehung realisiert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	Auftrag		:Containment DomCard(1) RngCard(1..n) Lifespan(dependent) Var(constant)	Auftragsposition	

Skript 156: Verfeinerung einer Operation durch eine statische Kollaboration

Im folgenden Beispiel wird eine Operation durch Zustandsveränderungen realisiert:

```
Licht :Class
Schalter betätigen :Operation
```

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Licht	brennt :State	:StateChange Act(Schalter betätigen :Operation)	:Licht	brennt nicht :State
	:Licht	brennt nicht :State	:StateChange Act(Schalter betätigen :Operation)	:Licht	brennt :State

Skript 157: Verfeinerung einer Operation durch Zustandsveränderungen

Die im Rahmen einer Operation stattfindenden Attributzugriffe werden als Skript formuliert:

```
Person :Class
heiraten :Operation
```

[1] Vgl. [Rumbaugh 1999: S. 439]. Auf die in der UML verwendeten Zusammenhänge kann hier nicht im Einzelnen eingegangen werden. Eine Übersicht über die Zusammenhänge zwischen den Diagrammen findet sich in [Partsch 1998] und [Linssen 1999].

Im Rahmen dieser Operation findet eine Veränderung der Attribute `Familienstand` und `Steuerklasse` statt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Person	heiratet :Action	:Internal change(write) Content(”verheiratet”)	:Person	Familienstand :Attribute Type(String) Init(“ledig”)
	:Person	heiratet :Action	:Internal change(write) Content(vier)	:Person	Steuerklasse :Attribute Type(Integer) Init(1)

Skript 158: Aus einer Operation resultierende Attributveränderungen

Eine Operation kann durch weitere Interaktionen realisiert werden.

`Sachbearbeiter :Class`
`Rechnung erstellen :Operation`

Im Rahmen dieser Operation führt der Sachbearbeiter Interaktionen aus. In der Regel wird dabei der Participant der Operation zum Initiator der ersten Interaktion des Subskripts:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:SB	prüft Zahlungsmoral des Kunden :Action	:Interaction	:Kundendatei	Zahlungsmoral prüfen :Operation
	:SB	prüft Lieferungen :Action	:Interaction	:Auslieferungslager	Lieferstatus prüfen :Operation
	:SB	schreibt Rechnung	:Interaction	:Fakturierungsprogramm	Rechnung erstellen :Operation

Skript 159: Im Rahmen einer Operation stattfindende Interaktionen

Das Verfahren, Operationen durch Skripte zu realisieren, kann rekursiv angewendet werden, wenn in dem Skript mindestens eine Interaktion enthalten ist.

Nun soll die Verfeinerung betrachtet werden. Durch ein Subskript kann auch ein Wechsel der Abstraktionsebene vorgenommen werden, wie das folgende Beispiel demonstriert. In dem Skript werden mit Interaktionen eine Reihe von Geschäftsprozessen dokumentiert, die von einem Unternehmen ausgeführt werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :Person	bestellt Artikel :Action	:Interaction	:Unternehmen	Bestellung bearbeiten :Operation
	/Kunde :Person	wünscht Produktinformation :Action	:Interaction	:Unternehmen	liefert Werbematerial :Operation
	/Kunde :Person	reklamiert Lieferung :Action	:Interaction	:Unternehmen	Reklamation bearbeiten :Operation
	/Kunde :Person	storniert Bestellung :Action	:Interaction	:Unternehmen	Bestellung stornieren :Operation

Skript 160: Geschäftsprozesse eines Unternehmens

Eine Operation `Bestellung bearbeiten` wird nun exemplarisch in einem Subskript durch Operationen der einzelnen Abteilungen verfeinert.[1]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :Person	sendet Bestellung :Action	:Interaction	:Vertriebs-abteilung	erhält Bestellung :Operation
	:Vertrieb-abteilung	übergibt Bestellung :Action	:Interaction	:Auftrags-bearbeitung	prüft Bestellung :Operation
	:Vetriebs-abteilung	veranlasst Lieferung :Action	:Interaction Guard(Bestellung.Status = lieferbar)	:Versand-abteilung	liefert Bestelleung aus :Operation
	:Vetriebs-abteilung	veranlasst Nachbesserung :Action	:Interaction Guard(Bestellung.Status = unvollständig)	:Support-abteilung	Bestellung nachbessern :Operation
	:Vetriebs-abteilung	veranlasst Lieferung :Action	:Interaction Guard(Bestellung.Status = korrigiert)	:Versand-abteilung	liefert Bestelleung aus :Operation
	:Vetriebs-abteilung	lässt Bestellung stornieren :Action	:Interaction Guard(Bestellung.Status = nicht korrigierbar)	:Auftrags-bearbeitung	storniert Bestellung :Operation

Skript 161: Skript mit Wechsel der Abstraktionsebene

Dieses Skript stellt einen Wechsel der Abstraktionsebene in der Art dar, dass der Participant der Operation (hier: Unternehmen) in mehrere Objekte (hier: die beteiligten Abteilungen) zerlegt wird, die die Operation (hier: Bestellung bearbeiten) bilden. Dies wird als hier *objektorientierte Verfeinerung* bezeichnet. Sie lässt sich rekursiv anwenden. Im hier genannten Beispiel könnte die Operation einer bestimmten Abteilung in einem Skript in die Operationen zerlegt werden, die die zugehörigen Sachbearbeiter und ihre Sachmittel ausführen. Eine einzelne Operation dieses Skripts

[1] Die Zeilennummern wurden weggelassen.

kann dann in einer weiteren Verfeinerung durch ein weiteres Skript verfeinert werden, in dem beispielsweise die Abläufe in einem betrieblichen Anwendungssystem dargestellt werden. Man hätte auf diese Weise folgende Modellebenen erhalten:

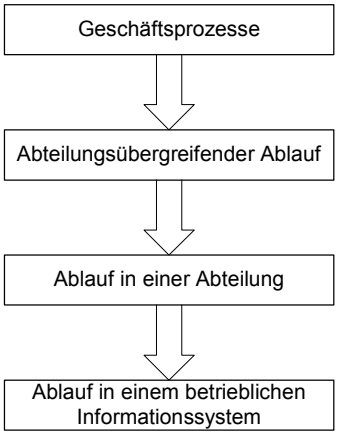


Abbildung 115: Abstraktionsebenen der Modellierung

Die andere Form wird hier *operative Verfeinerung* genannt. Sie entspricht dem in CATALYSIS verwendeten Verfahren, global beschriebene Interaktionen durch feingranularere Interaktionen zu präzisieren.[1] Während bei der objektorientierten Verfeinerung der Participant in einzelne Objekte zerlegt wird, wird bei der operativen Verfeinerung die Operation des Participants in einzelne Schritte zerlegt. Dies wird an folgendem Beispiel erläutert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	gibt Bestellung ein :Action	:Interaction	:Online- Bestellsystem	nimmt Bestellung entgegen :Operation

Skript 162: Zu verfeinernde Operation

In einem Subskript wird die Operation `nimmt Bestellung entgegen` durch die Dokumentation von Detailoperationen verfeinert:

[1] Vgl. [D'Souza 1998: S. 4-183], [D'Souza 1999: S. 7, 15, 224].

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :Person	wählt Artikel aus :Action	:Interaction	:Online-Bestellsystem	nimmt Artikelauswahl entgegen :Operation
	/Kunde :Person	gibt Kreditkartendaten ein :Action	:Interaction	:Online-Bestellsystem	nimmt Kreditkartendaten entgegen :Operation
	/Kunde :Person	bestätigt Bestellung :Action	:Interaction	:Online-Bestellsystem	schließt Bestellvorgang ab :Operation

Skript 163: Operative Verfeinerung einer Operation

In einem weiteren Schritt könnte nun beispielsweise die Operation `nimmt Kreditkartendaten entgegen` in einem Skript in einzelne Teiloperationen zerlegt werden, die die Kreditkartennummer und das Gültigkeitsdatum entgegennehmen und prüfen. Auf diese Weise erhält man eine Hierarchie von Operationen, in der Operationen durch „elementarere“ Operationen zerlegt werden. Beide Verfahren, Realisierung und Verfeinerung, können wiederholt und kombiniert eingesetzt werden. Die folgende Tabelle fasst die Zusammenhänge noch einmal zusammen:

Subskript enthält:	Entspricht in der UML:	Beziehung zwischen Skript und Subskript:
Realisierung		
Zustandsübergänge	Realisierung einer Operation durch eine Zustandsmaschine	Participant der Operation des Skripts ist Initiator und Participant des Subskripts.
Interaktionen	Realisierung einer Operation durch eine dynamische Kollaboration	Participant der Operation des Skripts ist Initiator mindestens einer Interaktion des Subskripts.
Referentielle und taxonomische Beziehungen	-	Participant der Operation des Skripts ist Initiator oder Participant mindestens einer Beziehung.
Interne Beziehungen	-	Participant der Operation des Skripts ist Initiator und Participant des Unterskripts.
Verfeinerung		
Objektorientierte Verfeinerung der Interaktion	-	Der Participant des Skripts wird zerlegt.
Operative Verfeinerung der Interaktion	-	Die Operation des Skripts wird zerlegt.

Tabelle 12: Inhalte von Subskripten

Subskripte und Abstraktionsbeziehungen ermöglichen sowohl die Modellierung sog. *High-Level*-Ansichten, wie sie von EMBLEY, KURTZ und WOODFIELD in der Methode OSA eingeführt wurden,[1] wie auch die Modellierung von *Whitebox*- und *Blackbox*-Ansichten.

Eine High-Level-Ansicht stellt die Abstraktionsebene eines Anwendungsfalldiagramms der UML dar. Unter Verbergung interner Details werden die Interaktionen eines Systems, Subsystems oder einer Gruppe von Objekten mit einem externen System abgebildet. Während bei Whitebox-Ansichten die Interna des Systems sichtbar sind, bleiben sie bei der Blackbox-Ansichten verborgen. Für eine fiktive Finanzbuchhaltung sähe dies etwa wie folgt aus:

[1] Vgl. [Embley 1992: S. S. 98 - 166].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Benutzer	erstellt Jahresabschluss :Action	:Interaction	:FiBu	Jahresabschluss erstellen :Operation
:Benutzer	bucht mit Buchungsschlüssel :Action	:Interaction	:FiBu	Buchen mit Schlüssel :Operation
:Benutzer	meldet sich am System an :Action	:Interaction	:FiBu	anmelden :Operation
:Bilanzbuchhalter	bucht :Action	:Interaction	:FiBu	buchen :Operation
:Bilanzbuchhalter	richtet die FiBu ein :Action	:Interaction	:FiBu	einrichten :Operation

Skript 164: High-Level- und Blackbox-Ansicht einer Finanzbuchhaltung

Das Skript stellt gleichzeitig eine *Blackbox*-Sicht auf das System dar, weil die interne Struktur, also die Objekte, die die Finanzbuchhaltung bilden, verborgen sind. Das folgende Skript verfeinert die letzte Interaktion aus Skript 164. Die Blackbox-Sicht wurde dabei beibehalten.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Bilanzbuch-halter	richtet Buchungschlüssel ein :Action	:Interaction	:FiBu	Buchungschlüssel einrichten :Operation
:Bilanzbuch-halter	richtet Kontenplan ein :Action	:Interaction	:FiBu	Kontenplan einrichten :Operation
:Bilanzbuch-halter	richtet Abschluss ein :Action	:Interaction	:FiBu	Abschluss einrichten :Operation
:Bilanzbuch-halter	richtet Geschäftsjahr ein :Action	:Interaction	:FiBu	Geschäftsjahr einrichten :Operation
:Bilanzbuch-halter	richtet Buchungsperiode ein :Action	:Interaction	:FiBu	Buchungsperiode einrichten :Operation
:Bilanzbuch-halter	richtet Mandant ein :Action	:Interaction	:FiBu	Mandant einrichten :Operation
:Bilanzbuch-halter	konfiguriert Sachkonto :Action	:Interaction	:FiBu	Sachkonto konfigurieren :Operation

Skript 165: Verfeinerung des letzten Skripts als Blackbox-Ansicht

In einer weiteren Verfeinerung wird an Stelle der Blackbox- nun eine Whitebox-Ansicht verwendet. Dadurch wird das Detail sichtbar, dass der Bilanzbuchhalter mit einem bestimmten Objekt der Finanzbuchhaltung interagiert. Dies wird wieder am Beispiel der letzten Interaktion aus Skript 165 demonstriert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Bilanzbuchhalter	richtet Buchungsschlüssel ein :Action	:Interaction	:Sachkonto	anlegen :Operation
	:Bilanzbuchhalter	löscht Sachkonto :Action	:Interaction	:Sachkonto	löschen :Operation
	:Bilanzbuchhalter	ändert Sachkonto :Action	:Interaction	:Sachkonto	Sachkontentyp ändern :Operation
	:Bilanzbuchhalter	ändert Einstellungen für Abschlüsse :Action	:Interaction	:Sachkonto	Abschlusseinstellungen ändern :Operation

Skript 166: Verfeinerung der letzten Interaktion durch Whitebox-Ansicht

Man hätte auch schon nach Skript 164 eine Whitebox-Ansicht verwenden können, da aus den Bezeichnungen der Operationen schon die Klassennamen zu entnehmen waren:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Bilanzbuchhalter	richtet Buchungschlüssel ein :Action	:Interaction	:Buchungs-schlüssel	einrichten :Operation
:Bilanzbuchhalter	richtet Kontenplan ein :Action	:Interaction	:Kontenplan	einrichten :Operation
:Bilanzbuchhalter	richtet Abschluss ein :Action	:Interaction	:Abschluss	einrichten :Operation
:Bilanzbuchhalter	richtet Geschäftsjahr ein :Action	:Interaction	:Geschäftsjahr	einrichten :Operation
:Bilanzbuchhalter	richtet Buchungsperiode ein :Action	:Interaction	:Buchungs-periode	einrichten :Operation
:Bilanzbuchhalter	richtet Mandant ein :Action	:Interaction	:Mandant	einrichten :Operation
:Bilanzbuchhalter	konfiguriert Sachkonto :Action	:Interaction	:Sachkonto	konfigurieren :Operation

Skript 167: Verfeinerung durch Whitebox-Ansicht

Spätestens auf dieser Ebene ist es angebracht, durch mehrere Skripte jede der aufgeführten Operationen zu modellieren. Wie viele Verfeinerungen notwendig sind, bis eine Whitebox-Ansicht möglich ist, ist vom Anwendungsfall abhängig. Aber auch bei komplexen Systemen sind erfahrungsgemäß drei bis vier Ebenen ausreichend. In dem trivialen Fall aus Skript 164 könnte man die Ebene der High-Level-Ansicht verlassen und durch mehrere Skripte die unterschiedlichen Abläufe dokumentieren, die beispielsweise beim `buchen` aus Skript 164 auftreten können. Dies würde für ein fiktives Finanzbuchhaltungssystem zu den Skripten

- einfache Buchung erstellen,
- zusammengesetzte Buchung erstellen,
- Korrekturbuchung erstellen
- Buchung mit Fremdwährung erstellen
- Buchung mit verschiedenen Steuersätzen erstellen

führen. Dabei existiert für die Länge eines Skripts keine Obergrenze. Orientiert man sich an BEHRINGER, ist bei der Modellierung von Interaktionen in folgenden Fällen das Ende eines Skripts erreicht:[1]

- Nach einer Eingabe oder Benachrichtigung durch den Initiator endet das Skript, wenn der Participant erneut eine Eingabe oder Benachrichtigung benötigt.
- Ein Skript beginnt mit einem als relevant angesehenen Systemzustand und endet mit einem anderen Systemzustand, der als relevant angesehen wird.
- Das Skript wird als zusammengehörende Aufgabe angesehen. Das Skript beginnt mit dem ersten Arbeitsschritt der Aufgabe und endet, wenn diese Aufgabe erfüllt ist.
- Durch ein Skript wird der Lebenszyklus eines Systems oder Objekts beschrieben. Das Skript beginnt mit dem Erzeugen des Systems oder Objekts und endet mit dem Löschen des Systems oder Objekts.

Durch Subskripte können sowohl Interaktionen als auch die an den Interaktionen beteiligten Objekte und deren Operationen zerlegt werden.[2] Abschließend sollen nun Reihenfolgebeziehungen zwischen Skripten dargestellt werden.

6.12.3 Reihenfolgebeziehungen für Skripte

Skripte besitzen einen Namen, der in der Form `Skriptname :Script` angegeben wird. Im Metamodell des Skriptmodells sind dem Metatyp *SCRIPT* zwei Prädikate *Vorkontext* und *Nachkontext* zugeordnet.[3] Der Vorkontext dokumentiert, welche Situation vor der Ausführung des Skripts gelten soll. Der Nachkontext dokumentiert, welche Situation nach der Ausführung des Skripts gelten soll. Wenn ein Skript genau einen Vorkontext besitzt, dokumentiert dies die Situation, die eingetreten sein muss, damit das Skript ausgeführt werden kann. Der Vorkontext wird durch das Prädikat `PreContext(`*`Text`*`)` dokumentiert.

---

[1] Vgl. [Behringer 1997: S. 21].

[2] In der Methode SOM werden ähnliche Konzepte verwendet. Vgl. [Ferstl 1993b: S. 139, 144, 147 f.], [Ferstl 1998 S. 39, 186 ff.], [Ferstl 1996: S. 53 – 57]. RÜFFER demonstriert die Anwendung dieser Konzepte bei der objektorientierten Modellierung am Beispiel der Geschäftsprozesse von Lebensversicherungsunternehmen. Vgl. [Rüffer 1999]. Die dabei verwendeten Zerlegungsprinzipien sind in [Ferstl 1995b] dokumentiert. Der hier konzipierte Ansatz ist aber umfassender, weil nicht nur Interaktionen durch Interaktionen zerlegt werden können, sondern auch durch Zustandsübergänge und statische Beziehungen. Darüber fehlt in der Methode SOM die explizite Definition der Abstraktionsbeziehungen.

[3] Vgl. Abschnitt 5.2.7.

```
Versicherungsantrag prüfen :Script
      PreContext(   Kunde hat Versicherungsantrag eingereicht.)
```

Damit das Skript `Versicherungsantrag prüfen` ausgeführt wird, muss die Situation eingetreten sein, dass ein Kunde einen Versicherungsantrag eingereicht hat. Der Nachkontext dokumentiert, welche Situation eingetreten sein soll, wenn das Skript ausgeführt wurde. Der Nachkontext wird durch das Prädikat `PostContext(`*`Text`*`)` dokumentiert.

```
Versicherungsantrag prüfen :Script
      PreContext(   Kunde hat Versicherungsantrag eingereicht.)
      PostContext( Versicherungsantrag ist geprüft.)
```

Wenn als Ergebnis eines Skripts unterschiedliche Situationen herbeigeführt werden können, wird dies durch einen entsprechenden Ausdruck dokumentiert. Das Skript `Versicherungsantrag prüfen` kann auch zu der Situation geführt haben, dass der Antrag nicht geprüft werden konnte. Diese Disjunktion wird durch den Operator XOR ausgedrückt:

```
Versicherungsantrag auf Vollständigkeit prüfen :Script
      PreContext(   Kunde hat Versicherungsantrag eingereicht.)
      PostContext(( Versicherungsantrag ist vollständig.) XOR
                  ( Versicherungsantrag ist unvollständig.))
```

Ein anderes Skript, welches einen entsprechenden Vorkontext besitzt, wird als Folge ausgeführt:

```
Versicherungsantrag vervollständigen :Script
      PreContext(   Versicherungsantrag ist unvollständig.)
      PostContext(( Versicherungsantrag ist vollständig ausgefüllt.) XOR
                  ( Versicherungsantrag ist angenommen.))
```

Ein weiteres Skript wird ausgeführt, wenn der Versicherungsantrag angenommen worden ist:

```
Versicherungsantrag auf Realisierbarkeit prüfen :Script
      PreContext(   Versicherungsantrag ist angenommen.)
      PostContext(( Versicherungsantrag realisierbar.) XOR
                  ( Versicherungsantrag ist nicht realisierbar.))
```

Ist der Versicherungsantrag nicht realisierbar, wird in einem weiteren Skript dem Antragsteller die Ablehnung zugesandt:

```
Versicherungsantrag ablehnen :Script
      PreContext(   Versicherungsantrag ist nicht realisierbar.)
      PostContext(  Ablehnung versendet.)
```

Die Ähnlichkeit mit den Vor- und Nachbedingungen, mit denen die Semantik von Operationen spezifiziert wird, ist beabsichtigt. Während die Vor- und Nachbedingungen auf eine einzelne Operation bezogen sind, beziehen sich die Vor- und Nachkontexte auf Skripte. Durch die unterschiedlichen Nachkontexte ist es möglich, in separaten Skripten Standardverläufe und besondere Situationen getrennt zu dokumentieren. Dies soll an folgendem Beispiel demonstriert werden:

```
Vertrag ausstellen :Script
      PreContext(   Einigkeit der Vertragsparteien.)
      PostContext(( Vertrag ist ausgestellt.) XOR
                  ( Vertrag ist gegenzuzeichnen.))
Vertrag gegenzeichnen :Script
          PreContext(         Vertrag ist gegenzuzeichnen.)
          PostContext(        Vertrag ist ausgestellt.)
Vertrag zustellen :Script
          PreContext(         Vertrag ist ausgestellt.)
          PostContext( Vertrag ist Kunden zugestellt.)
```


Dieser Prozess beinhaltet zwei mögliche Abläufe. Entweder wird der Vertrag im Skript `Vertrag ausstellen` fertig ausgestellt und anschließend im Skript `Vertrag zustellen` dem Kunden zugestellt, oder er wird im Skript `Vertrag gegenzeichnen` fertiggestellt und dann zugestellt. Der Operator XOR in einem Nachkontext drückt einen exklusiv-alternativen Verlauf aus.

Der Operator AND wird verwendet, um eine logische Konjunktion auszudrücken. Beide Ausdrücke im Term des Nachkontext sind nach der Ausführung des Skripts wahr. Dies wird an folgendem Beispiel demonstriert:

```
Ware fertiggestellt: Script
      PreContext( ...)
      PostContext( Ware ist zur Auslieferung bereit.) AND
                  ( Rechnung kann gestellt werden.))
```

Der Vorkontext ist an dieser Stelle nicht von Bedeutung. Der Nachkontext besteht aus zwei Ausdrücken, die nach der Beendigung des Skripts wahr sind. Als Folge dieses Skripts werden zwei andere Skripte ausgeführt:

```
Ware liefern: Script
      PreContext( Ware ist zur Auslieferung bereit.)
      PostContext( Ware ist geliefert.)
Rechnung stellen: Script
      PreContext( Rechnung kann gestellt werden.)
```

```
        PostContext( Rechnung wurde versendet.)
```

Auch der Vorkontext eines Skripts kann aus mehreren Ausdrücken bestehen, die durch OR, XOR und AND verbunden werden. Durch OR wird ausgedrückt, dass ein Skript in unterschiedlichen Situationen ausgeführt wird:

```
Schriftstück bearbeiten: Script
        PreContext(( Brief ist eingetroffen.) OR
                    ( Fax ist eingetroffen.) OR
                    ( Email ist eingetroffen.))
        PostContext( Schriftstück ist Posteingang zugeordnet.)
```

Dagegen wird AND verwendet, um auszudrücken, dass beide Ausdrücke des Vorkontexts eingetreten sein müssen:

```
Offenen Posten auflösen: Script
        PreContext(( Ware ist geliefert.) AND
                    ( Kunde hat bezahlt.)
        Postcontext( Offener Posten ist aufgelöst.)
```

Eine Disjunktion wird verwendet, wenn exakt eine der Bedingungen des Vorkontexts eingetreten sein muss, damit dass Skript ausgeführt wird. Mit den Operatoren XOR, OR, AND und der unbedingten Sequenz lassen sich alle notwendigen Reihenfolgebeziehungen ausdrücken.[1]

Für eine mögliche Abfolge von Skripten wird der Begriff *Szenario* verwendet. Die Wahl des Begriffes Szenario verdeutlicht, dass man bei der Analyse von Prozessen in der Lage ist, unterschiedliche Verläufe zu erfassen. In der ursprünglichen Konzeption der OBA entsprach ein Skript einem Szenario.[2] Die meisten Modellierungsansätze, die Interaktionsmodelle einsetzen, verwenden diese Einschränkung. Daraus resultieren zwei erhebliche Nachteile, die die Anwendung für komplexe Probleme nahezu unmöglich werden lassen:

1. Auch ein einzelnes Szenario kann so komplex werden, dass es nicht mehr in einem Diagramm dargestellt werden kann.

---

[1] Diese Operatoren stammen aus der von SHANNON entwickelten Schaltalgebra. Vgl. [Fischermanns 1997: S. 44]. Die hier verwendete Konzeption, Prozesse auf diese Weise zu strukturieren, wurde von den Ereignisgesteuerten Prozessketten (EPK) der Methode ARIS übernommen, die allgemeine Akzeptanz bei der Modellierung betriebswirtschaftlicher Prozesse finden. Vgl. [Keller 1992], [Hoffmann 1992], [Scheer 1995: S. 49], [Vossen 1996], [Jablonski 1997], [Keller 1998: S. 158 ff.], [Becker 2000].

[2] Vgl. [Rubin 1992].

2. Es besteht keine Möglichkeit, mehrfach verwendete Abläufe (= Skripte) zu modularisieren und in unterschiedlichen Szenarien einzusetzen.

Aus diesem Grund wird diese Einschränkung nicht übernommen. Ein Skript ist Teil keines, eines oder mehrere Szenarien.[1] Die Abfolge der Skripte innerhalb eines Szenarios ist festgelegt, d. h., die einzelnen Skripte stehen in einer Reihenfolgebeziehung.[2]

6.13 Zusammenfassung

Der Inhalt von Kapitel 5 war der Aufbau des Skriptmodells. Das Skriptmodell ist eine Repräsentationsform des im Kapitel 4 entwickelten Konzeptuellen Schemas. In diesem Kapitel werden die Skripte zur objektorientierten Modellierung verwendet. Schwerpunkt der Darstellung sind die unterschiedlichen Prädikate, die in den einzelnen Tabellenspalten der Skripte verwendet werden. Durch diese Prädikate existieren bei der Modellierung größere Möglichkeiten als in der UML. Entsprechend dem in Abschnitt 1.4 formulierten *Primat der Interaktionsbeziehung* wurde besonderer Augenmerk auf die Prädikate der Interaktionsbeziehung gelegt. Außerdem sind die im dritten Kapitel aufgezählten Probleme und Unklarheiten der OBA gelöst bzw. beseitigt worden:

[1] Vgl. das Klassendiagramm in Abbildung 93 auf Seite 205.

[2] SCHWEGMANN hat mehrere Ansätze entwickelt, die Modelle komplexer Prozesse zu strukturieren. Darauf wird hier jedoch nicht eingegangen. Vgl. [Schwegmann 2000].

OBA	OBA++
Die Konzepte Objekt und Rolle werden vermischt.	Es wird zwischen Objekt, Klasse und Rolle unterschieden. Der Zusammenhang ist durch das Konzeptuelle Schema festgelegt.
Die Erstellung der Objektmodell-karten ist unklar	Statt Objektmodellkarten können aus Skripten UML-Klassendiagramme erzeugt werden.
Die Semantik der Aktionen ist unklar.	Aktionen sind die Ursache von Ereignissen.
Die Bedeutung des Begriffs Contract ist unklar.	Verträge werden durch Vor- und Nachbedingungen beschrieben.
Keine Kontrollanweisungen in Skripten vorhanden.	Kontrollanweisungen werden durch Zeilennummern dargestellt.
Die Verwendung von Attributen ist unklar.	Attribute sind ein Metatyp des Konzeptuellen Schemas. Auf Attribute wird durch objektinterne Beziehungen zugegriffen.
Skripte beinhalten nur Interaktions-beziehungen	Skripte enthalten auch referentielle und taxonomische Beziehungen.
Der Inhalt einer Interaktion kann nicht abgebildet werden.	Der Inhalt der Interaktion wird durch die Prädikate `Content()` und `Answer()` abgebildet.
Es ist unklar, wie Unterskripte verwendet werden.	Unterskripte stellen Realisierungen und Verfeinerungen dar.
Strukturierung von Skripten unklar.	Skripte werden durch Verfeinerungen, Realisierungen und Reihenfolgebeziehungen strukturiert.
Keine Ebenen- bzw. Abstraktions-bildung.	Bildung von Betrachtungsebenen durch Unterskripte.
Keine Aussage, ob alle Spalten einer Zeile in einem Skript besetzt sein müssen.	Durch den Typ der Connection wird vorgegeben, welche Spalten besetzt sein müssen.

Tabelle 13: Schwächen der OBA und ihre Lösung in der OBA++

Mit diesem Kapitel ist die Entwicklung des Modellierungsansatzes OBA++ abgeschlossen. Inhalt des nächsten Kapitels ist die Anwendung des Skriptmodells zur GPM.

7 Die Geschäftsprozessmodellierung mit der OBA++

„Wer sich die Gliederung einer Betriebsaufgabe als eine Struktur vorstellt und dabei im Geiste die Hierarchie einer Stellenfolge vor sich sieht, muss sich darüber im klaren sein, dass er eine Vereinfachung, eine Abstraktion, vorgenommen hat. Er hat nämlich weggelassen die Vorstellung der Tatsache, dass der Betrieb in Wirklichkeit ein fortwährender Prozess, eine ununterbrochene Leistungskette ist."

[Nordsieck 1972: Sp. 9 ff.].

Mit dem letzten Kapitel ist die Entwicklung der erweiterten Object Behavior Analysis (OBA++) abgeschlossen. Es wurde gezeigt, wie die OBA++ aufgebaut ist und wie man mit Skripten modelliert. Das Ziel ist aber, einen objektorientierten Ansatz zur Modellierung von Geschäftsprozessen zu entwickeln. Hierfür ist es notwendig zu demonstrieren, wie Geschäftsprozesse mit der OBA++ modelliert werden. Daraus ergibt sich die Aufgabe für dieses Kapitel: Zunächst werden die Phänomene zusammengefasst, die bei der GPM zu berücksichtigen sind. Dies sind die für den Modellierungsansatz zu erfüllenden Anforderungen.[1] Anschließend wird jeweils erläutert, wie diese Phänomene mit Hilfe der OBA++ abgebildet werden. Außerdem wird auch deutlich gemacht, welche Unterschiede zwischen der Objektorientierung und der Geschäftsprozessorientierung bestehen und bei der Modellierung zu beachten sind. Dabei werden weder neue Modellprimitive eingeführt, noch wird das Metamodell erweitert. Alle hier verwendeten Konzepte sind prinzipiell aus dem letzten Kapitel bekannt. Die Darstellung konzentriert sich auf die Anwendung und eventuelle Besonderheiten, die sich aus der GPM mit der OBA++ ergeben.[2]

[1] Diese Phänomene werden aus den relevanten Veröffentlichungen zum Thema Geschäftsprozessmanagement und Geschäftsprozessmodellierung gesammelt, die zu diesem Thema in großer Zahl vorliegen.

[2] Ansonsten begnügt sich die Darstellung mit Querverweisen auf das letzte Kapitel.

7.1 Ausgangspunkt

Ausgangspunkt bei der GPM sind die Abgrenzungen, die im ersten Kapitel getroffen wurden. Ein Geschäftsprozess ist ein Ablauf in einem soziotechnischen System. Das bedeutet, dass die innerhalb der Prozesse auszuführenden Aktivitäten teilweise von CIS und teilweise von Menschen ausgeführt werden. Die personellen Aufgabenträger verwenden computergestützte Informations- und Kommunikationssysteme, um ihre Aufgaben zu erfüllen. Welche Aktivitäten im Rahmen einer Prozessanalyse von CIS ausgeführt werden können oder sollen, ist das Ergebnis der Analyse, nicht ihr Ausgangspunkt. Das hat zur Folge, dass auch rein organisatorische Abläufe als Prozesse modelliert werden können müssen. In einem objektorientierten Ansatz handelt es sich bei solchen Prozessen um eine zum Teil geordnete Menge von Interaktionen, die zwischen Organisationseinheiten stattfinden. Die beteiligten Objekte zeigen im Rahmen der Interaktionen Verhalten und tauschen Materie, Energie und Informationen aus.

Durch die OBA++ kann die GPM mit unterschiedlichen Präzisionsgraden erfolgen. Mit zunehmendem Erkenntnisfortschritt werden im Modell in Form von zusätzlichen Prädikaten weitere Details hinzugefügt und damit sukzessive präzisiert. Für die Erstellung einer Interaktion muss man zu Beginn der Modellierung nur wissen, wer miteinander interagiert und welche Aktivitäten die Beteiligten ausführen:[1]

:Abteilungsleiter	veranlasst Brieferstellung :Action	:Interaction	:Schreibkraft	erstellt Brief :Operation

Im weiteren Verlauf der Modellierung definiert man den In- und Output der Operation, ihre Nebenläufigkeit, ihre Sichtbarkeit, Ziele, Kosten, zeitliche Restriktionen usw. Auch für die Interaktion werden zeitliche Restriktionen, die Koordination, die Nebenläufigkeit usw. bestimmt. Der Vorteil der OBA++ besteht darin, dass mit den Skripten immer der gleiche Formalismus verwendet wird. Zwischenergebnisse der Modellierung werden – so lange sie sich nicht als falsch herausstellen – immer weiter präzisiert, indem man weitere Prädikate hinzufügt.

[1] Auf die Möglichkeit, auch die Bezeichnung der *ACTION* wegzulassen, wird hier bewusst nicht eingegangen, da diese Möglichkeit nur eingeführt wurde, um UML-Interaktionsdiagramme in Skripte übersetzen zu können.

7.2 Elemente von Geschäftsprozessen

7.2.1 Gestaltungselemente von Geschäftsprozessen

Üblicherweise bezeichnet man Aufgaben, Aufgabenträger, Informationen und Sachmittel als die Gestaltungselemente organisatorischer Systeme.[1] Davon leicht abweichend werden als *Bausteine von Geschäftsprozessen* Personen, Organisationseinheiten, Aktivitäten, Geschäftsobjekte und die verwendeten Sachmittel bezeichnet.[2]

Abgesehen von den Aufgaben und Aktivitäten, sind alle Gestaltungselemente und Bausteine von Geschäftsprozessen als Klassen bzw. als Objekte von Klassen in den Spalten Initiator und Participant zu finden. Dabei wird gegebenenfalls zusätzlich ein Stereotyp (Abschnitt 6.1.2) verwendet, um die Klasse zu kategorisieren. Stereotype werden insbesondere für die Objekte und Klassen verwendet, bei denen es sich nicht um Informationsobjekte handelt. Das sind die Objekte und Klassen, die sich außerhalb der Sphäre der CIS befinden. Die Verwendung von Stereotypen ist in der OBA++ deshalb so wichtig, weil die Betrachtung nicht auf die CIS beschränkt ist. Während man bei der Softwareentwicklung nur Informationsobjekte und maximal Akteure modelliert, werden bei der GPM organisatorische Abläufe modelliert, die unter Umständen vollständig außerhalb des CIS stattfinden.[3] Die Grenze zwischen Diskurswelt und Umwelt ist bei der GPM anders gezogen. Bei der Softwareentwicklung liegt sie zwischen Anwender und Softwaresystem. Bei der GPM ist die Grenze die betrachtete Organisation und kann im Fall externer Prozessverkettung (vgl. Abschnitt 7.3.10) sogar jenseits dieser Grenze liegen. Im Gegensatz zur oEPK[4], in der Organisationseinheiten nicht als Klassen abgebildet werden, werden hier die Gestaltungselemente von Geschäftsprozessen als Klassen und Objekte abgebildet, unabhängig davon, ob sie zum betrieblichen Informationssystem gehören oder nicht. Damit ist aber die Notwendigkeit gegeben, zwischen den unterschiedlichen Arten von Klassen differenzieren zu können, was hier durch Stereotype erfolgt. Diese Notwendigkeit ist im Bereich der objektorientierten Methoden zur Softwareentwicklung nicht gegeben, weswegen auch auf keine verwendbare Typisierung aus der UML zurückgegriffen werden kann. Die folgende Tabelle fasst die in Abschnitt 6.1.2 definierten Terme des Prädikats `Stpe()` zusammen und gibt Kriterien zu deren Verwendung an:

---

[1] Vgl. [Schmidt 1985: S. 19], [Krüger 1994: S. 21], [Bleicher 1991: S. 35]. Teilweise werden auch nur Aufgaben, Personen und Sachmittel genannt. Vgl. [Bleicher 1993: S. 116].

[2] Vgl. [Schulte-Zurhausen 1995: S. 50, 59 ff.], [Striening 1988: S. 59], [Völkner 1998: S. 10], [Staud 1999: S. 13].

[3] Vgl. Abschnitt 2.7.

[4] Vgl. [ZimmermannV 1999], [Scheer 1997], [Nüttgens 1998].

Betriebswirtschaftlicher Begriff	**Stereotyp**	**Wird verwendet, ...**
Abteilung	`Department`	... wenn es sich bei dem Aufgabenträger um mehrere Personen, Gruppen oder Stellen handelt.
Aktionseinheit	`Action Unit` bzw. `Act-Unit`	... wenn der Aktionsträger eine aus einem Aufgabenträger und Sachmitteln bestehende Einheit darstellt.
Aktionsträger	`Action Bearer`	... nicht festgelegt wird, ob eine Aktivität von einem Sachmittel oder von einem Aufgabenträger ausgeführt wird.
Arbeitsmittel	`Work Instrument`	... wenn ein Sachmittel seine Aktivitäten in Verbindung mit einem Aufgabenträger ausführt, also kein Arbeitsträger ist.
Arbeitsträger	`Work Bearer`	... wenn ein Sachmittel eine Aktivität relativ selbständig ausführt.
Aufgabenträger	`Task Bearer`	... wenn es sich um einen oder mehrere menschliche Aktionsträger handelt, aber nicht festgelegt wird, ob es sich um eine Stelle, Abteilung oder Gruppe handelt.
Geschäfts- oder Arbeitsobjekt	`Business Object` bzw. `BO`	... wenn es sich um ein im Rahmen eines Prozesses erstelltes, be- oder verarbeitetes Objekt handelt.
Gruppe	`Group`	... wenn es sich bei dem Aufgabenträger um mehrere Stellen handelt.
Individuum	`Individual`	.. wenn es sich um eine natürliche Person handelt, die nicht als Aufgabenträger innerhalb einer Organisation abgebildet werden soll.
Informationsobjekt	`Entity`	... wenn es sich bei einem Geschäftsobjekt um Informationen handelt.
Organisation	`Organization`	... wenn eine Organisation als Ganzes betrachtet werden soll.
Organisationseinheit	`Org-Unit`	... wenn ein Element der Aufbauorganisation betrachtet wird.
Problem	`Problem`	... wenn ein Problem als Objekt modelliert wird, z. B. weil

Betriebswirtschaftlicher Begriff	**Stereotyp**	**Wird verwendet, ...**
		dessen Zustand von Bedeutung ist.
Prozess	`Process`	... wenn ein Prozess als eine eindeutig identifizierbare Entität mit Eigenschaften, Zuständen und Zustandsveränderungen betrachtet wird oder zwischen Prozessen Spezialisierungsbeziehungen modelliert werden sollen.
Sachmittel	`Resource`	... wenn Sachmittel abgebildet werden sollen, aber nicht genauer festgelegt wird, ob es sich um einen Arbeitsträger oder ein Arbeitsmittel handelt.
Stelle	`Position`	... wenn es sich bei dem Aufgabenträger um eine einzelne Person handelt.
Wissen	`Knowledge`	... wenn es sich um eine besondere Form von Arbeitsmitteln handelt, die in Form von Methoden, Regeln, Verfahren oder Kenntnissen vorliegt.
Ziel	`Goal`	... wenn ein Ziel als Objekt modelliert wird, z. B. weil dessen Zustand von Bedeutung ist.

Tabelle 14: Stereotype zur GPM

Die gewählte Konzeption für Stereotype hat den Effekt, dass sich bei der Modellierung Freiheitsgrade abbilden lassen. Soll noch nicht festgelegt werden, ob eine Aktivität von einem Sachmittel oder von einem Aufgabenträger (`Task Bearer`) ausgeführt wird, verwendet man das Stereotyp Aktionsträger (`Action Bearer`). Wird die Aktivität von einem Sachmittel ausgeführt, verwendet man das Stereotyp Sachmittel (`Resource`). Man kann schließlich noch unterscheiden, ob dieses Sachmittel die Aktivität relativ selbständig (Arbeitsträger bzw. `Work Bearer`) oder nur in Verbindung mit einem Aufgabenträger (Arbeitsmittel bzw. `Work Instrument`) ausführt. Computergestützte Informationssysteme werden dabei in der Regel als Arbeitsträger abgebildet. Auf diese Weise kann man

- offen lassen, ob ein Prozess von Menschen oder von Sachmitteln ausgeführt wird,
- festlegen, dass der Prozess von Arbeitsträgern ausgeführt wird, oder
- festlegen, dass der Prozess von Aufgabenträgern ausgeführt wird.

Auch bei den Aufgabenträgern im Speziellen hat man ähnliche Möglichkeiten. Die Verwendung des Stereotyps `Action Bearer` lässt offen, ob es sich um eine Stelle (`Position`), eine Gruppe (`Group`) oder eine Abteilung (`Department`) handelt. Das Stereotyp Organisationseinheit (`Org-Unit`) wird verwendet, wenn es sich um ein Element der Aufbauorganisation handelt, ohne sich präziser festlegen zu können oder zu wollen. Wenn beispielsweise ein Unterschriftsberechtigter ein identifizierbares und abgrenzbares Verhalten zeigt, ist es durchaus sinnvoll, ihn als Klasse abzubilden.[1] Stellt er kein Element des Stellenplans eines Unternehmens dar, bildet man ihn als Organisationseinheit ab und verbindet ihn gegebenenfalls mit den entsprechenden Stellen. Im folgenden Beispiel sind Unterschriftsberechtigte als Teilmenge der Abteilungsleiter abgebildet:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:SB Stpe(Position)	erfragt Genehmigung :Action	:Interaction	:Unterschrifts- berechtigter Stpe(Org-Unit)	erteilt Genehmigung :Operation
:Unterschrifts- berechtigter Stpe(Org-Unit)		ist Teilmenge von :Membership SubType(Member-Bunch) DomCard(0) RngCard(1)	:Abteilungsleiter Stpe(Position)	

Skript 168: Die Organisationseinheit Unterschriftsberechtigter als Teilmenge der Abteilungsleiter

Organisatorische Abläufe sind also daran zu erkennen, dass die beteiligten Objekte eines der Stereotype `Organization`, `Org-Unit`, `Action Bearer`, `Task Bearer`, `Group`, `Department`, `Position`, `Action Unit` oder `Individual` besitzen.

Das Stereotyp `Knowledge` wurde definiert, um neben materiellen Sachmitteln auch geistige Hilfsmittel, die von den Aufgabenträgern bei der Erfüllung von Aufgaben Verwendung finden, abgrenzen zu können. Häufig sind diese geistigen Hilfsmittel bestimmende Faktoren in betrieblichen Abläufen, werden jedoch übersehen, weil sie nicht unmittelbar den Charakter von Daten besitzen. Sie sind bei der GPM wichtig, weil überprüft werden muss, ob die Akquisition, Weitergabe und Speicherung von Wissen und Kenntnissen nicht durch ein CIS verbessert werden kann. Ähnlich verhält es sich mit Zielen (`Goal`) und Problemen (`Problem`). Ziele und Probleme sind wichtige Einflussfaktoren für die Abwicklung von betrieblichen Abläufen. Auch ihre Bedeutung wird bei der Untersuchung betrieblicher Abläufe übersehen, da sie häufig keinen Datencharakter besitzen.

[1] Bei organisatorischen Rollen liegt dieser Fall vor.

Die Gegenstände, an denen sich das Handeln in Geschäftsprozessen vollzieht, wurden als Geschäftsobjekte (`Business Object` oder `BO`) bezeichnet. Dies ist unabhängig davon, ob es sich um informationelle oder physische Objekte[1] handelt. Wenn Geschäftsobjekte eindeutig als Informationen gekennzeichnet werden sollen, wird das aus der UML entlehnte Stereotyp `Entity` verwendet.[2] In der Regel wird auf die Kennzeichnung verzichtet, da die Verwendung von Stereotypen darauf abzielt, die Klassen zu kennzeichnen, bei denen es sich um keine Geschäftsobjekte handelt. Hier hat sich die Heuristik bewährt, dass Geschäftsobjekte tendenziell häufiger in der Spalte Participant zu finden sind, weil an ihnen Verrichtungen vorgenommen werden.[3]

Bei der objektorientierten GPM ist außerdem bemerkenswert, dass die Klassen wesentlich feingranularer definiert werden. Je präziser man die Aufgaben einzelner spezifischer Organisationseinheiten definiert, desto tendenziell kleiner wird die Extension der einzelnen Klassen. Im Extremfall werden nur noch *Singletons* modelliert, weil jede einzelne Organisationseinheit – also jedes Objekt – spezifische Aufgaben bzw. ein spezifisches Verhalten besitzt. Dies ist die Ursache, warum – im Gegensatz zu anderen Modellierungsansätzen des Objektansatzes – bei der objektorientierten GPM die Verwendung von Stereotypen wesentlich wichtiger ist. Würde man keine Stereotype verwenden, müsste für jede spezifische Organisationseinheit eine Vererbungsbeziehung zu allgemeinen Klassen für Stellen und Abteilungen modelliert werden. Diese Lösung wurde aber als zu umständlich angesehen, wie das folgende Beispiel zeigt:

Fertigung		:Specialisation	Org-Unit	
Bauteillager		:Specialisation	Org-Unit	
:Fertigung	ruft Bauteile ab :Action	:Interaction	:Bauteillager	Bauteile zur Fertigung liefern :Operation

Skript 169: Eine alternative Modellierung von Organisationseinheiten ohne Stereotype

[1] Bei den physischen Geschäftsobjekten können Spezialisierungen für Rohstoffe, Fabrikate und Werkstoffe eingeführt werden.

[2] Bei Bedarf können natürlich noch Spezialisierungen für Stamminformationen, Bestandsinformationen etc. eingeführt werden. Vgl. [Schulte-Zurhausen 1999: S. 65], [Stahlknecht 1995: S. 167 f.], [REFA 1991/1: S. 181 ff.], [Mertens 1988: S. 16 ff.].

[3] Vgl. folgenden Abschnitt.

7.2.2 Aktivitäten

Aktivitäten bilden den Kern der Prozessabwicklung.[1] Dabei werden in den prozessorientierten Ansätzen eine Vielzahl unterschiedlicher Formulierungen verwendet. VÖLKNER zählt unter anderem die Begriffe Tätigkeit, Vorgang, Verrichtung, Operation, Handlung, Aufgabe, Funktion und Transformation auf.[2]

Aktivitäten sind zunächst *Verrichtungen* an Objekten.[3] Die Objekte (hier: *Geschäftsobjekte*) sind dabei die materiellen oder immateriellen Gegenstände, auf die sich die Verrichtungen beziehen.[4] Dies können Rohstoffe, Halbfabrikate, Produkte, Zeichnungen, Telefonanrufe, Kundenwünsche, Daten oder Formulare sein. Aktivitäten werden durch Subjekt und Prädikat beschrieben.[5] Durch die Aktivitäten wird das Bearbeitungsobjekt in seinem Zustand, seiner Form oder Lage verändert.[6] Diese *Transformationen* stellen den Übergang eines Ausgangszustandes, einer Ausgangsform oder Ausgangslage in einen Zielzustand, eine Zielform oder eine Ziellage dar.[7] Dieser Transformationsbegriff wird sowohl bei der Fertigung von Materialien zu Gütern als auch bei der Verarbeitung von Informationen angewendet.[8] Verrichtungen werden von allen Formen von Aktionsträgern ausgeführt. Das bedeutet, dass die in einem Prozess verwendeten Sachmittel ebenfalls Aktivitäten ausführen.

Eine weitere Form der Aktivität stellen Aufgaben dar. Die Zuweisung der einzelnen Aktivitäten und Prozesse auf eine oder mehrere Personen wird durch die Festlegung von (Stellen-)Aufgaben vorgenommen.[9] Die *Aufgabe* stellt eine Aufforderung bzw. Verpflichtung eines Aufgabenträgers

---

[1] Vgl. [Schulte-Zurhausen 1999: S. 60], [Völkner 1998: S. 10].

[2] Vgl. [Völkner 1998: S. 10].

[3] Vgl. [Schulte-Zurhausen 1999: S. 40, 60]. *Verrichtungen* sind Aktivitäten, die unabhängig von bestimmten Bearbeitungsobjekten betrachtet werden. [Nordsieck 1955: S. 84]. Typische Beispiele sind *schreiben*, *bohren*, *fräsen*. Dies wird von ZIMMERMANN als *Bearbeitungsaktivitäten* bezeichnet. Vgl. [ZimmermannV 1999: S. 123].

[4] Vgl. [Schulte-Zurhausen 1999: S. 60].

[5] Vgl. [Schulte-Zurhausen 1999: S. 60].

[6] In Anlehnung an den Aufgabenbegriff von KOSIOL formuliert. Vgl. [Kosiol 1962: S. 44].

[7] Vgl. [Bertram 1996: S. 86], [Rittgen 1998: S. 41 f.]. KOSIOL beschränkt diese Definition auf die Veränderung realer Dinge, da er die Bestimmbarkeit geistiger Tätigkeiten für problematisch hält. Vgl. [Kosiol 1962: S. 44]. Hier wird der Sichtweise von WILD gefolgt, der auch geistige Objekte als Arbeitsobjekte ansieht, auf die sich Verrichtungen beziehen können. Vgl. [Wild 1966: S. 92].

[8] Vgl. [Schmidt 1997: S. 11].

[9] „*Durch die Verknüpfung der Funktionen mit Organisationseinheiten wird deutlich, wem Aufgaben, Kompetenz und Verantwortung der Funktionsdurchführung obliegen.*“ M. ROSEMANN in [Becker 2000: S. 59]. Vgl. auch [Schulte-Zurhausen 1999: S. 100].

dar, eine Veränderung von Bearbeitungsobjekten (ggf. unter Zuhilfenahme von Arbeitsmitteln) durch Aktivitäten in Raum und Zeit vorzunehmen.[1] Die Erfüllung der Aufgabe geschieht durch Handlungen[2] bzw. Aktivitäten. Es lassen sich körperliche (*korporale*), geistige (*mentale*) und gemischte Handlungen unterscheiden.[3] Aufgaben sind tätigkeitsorientiert in dem Sinne, dass eine Veränderung herbeigeführt werden soll.[4]

Die durch Aktivitäten vollzogene Prozessabwicklung findet sich in den Skripten in den Spalten Initiator Responsibility und Participant Responsibility. Im Metamodell des Konzeptuellen Schemas handelt es sich immer um Knoten des Metatyps *ACTIVITY*. Dabei werden Verrichtungen, Aktivitäten (im Sinne der GPM), Transformationen und Aufgaben unterschieden.

Die Verrichtungen sind als Aktionen in der Spalte Initiator Responsibility aufgeführt. Die Objekte, die die Verrichtung ausführen (z. B. Aktionsträger), sind dabei der Initiator. Das bearbeitete Objekt ist der Participant. Dadurch wird die Aussage getroffen, dass von dem Initiator die Verrichtung an dem Participant vorgenommen wird:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Programmierer Stpe(Position)	programmieren :Action	:Interaction	:Programmquelle Stpe(BO)	erstellt werden :Operation

Skript 170: Verrichtung eines Aufgabenträgers

Die dadurch entstehende Transformation des bearbeiteten Objekts ist dessen Operation.[5] In der Spalte Initiator Responsibility kann auch die Bezeichnung einer Aktivität (im Sinne der GPM: Verrichtung & Objekt) eingetragen sein. Dies ist dann der Fall, wenn der Participant ein Aufgabenträger ist. Die den Aufgabenträgern zugeordneten Aufgaben (siehe Abschnitt 7.2.3)

---

[1] Vgl. [Krüger 1992: Sp. 223], [Schulte-Zurhausen 1999: S. 71]. HOFFMANN unterstreicht eher den Zielbezug, wenn er Aufgabe als durch physische oder geistige Aktivitäten zu erfüllendes Handlungsziel bezeichnet. Vgl. [Hoffmann 1992b: Sp. 212]. Im Ansatz ARIS werden Aufgaben und *Funktionen* gleichgesetzt. Vgl. [Keller 1992: S. 8].

[2] Vgl. [Kosiol 1962: S. 100].

[3] Vgl. [Kosiol 1962: S. 100], [Wild 1966: S. 91].

[4] Vgl. [Schulte-Zurhausen 1999: S. 71], [Bleicher 1991: S. 35].

[5] Hier konnte beobachtet werden, dass die Verben einer Fachsprache tendenziell häufiger in der Spalte Initiator Responsibility zu finden sind, wogegen in der Spalte Participant Responsibility Domänen-unabhängige Verben verwendet werden. Hier sei ein typisches Beispiel aus dem Bereich der Finanz-buchführung genannt: Der Verrichtung `buchen` in der Spalte Initiator Responsibility steht die Operation `erzeugen` einer Klasse `Buchung` gegenüber.

werden als Operationen abgebildet. Dadurch wird die Aussage getroffen, dass die Aufgabe von dem Aufgabenträger übernommen wird:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Mitarbeiter Stpe(Org-Unit)	Reisekostenabrechnung einreichen :Action	:Interaction	:Personalabteilung Stpe(Department)	Reisekostenabrechnung erstellen :Operation

Werden Sachmittel von anderen Objekten aufgefordert, Aktivitäten auszuführen, stellt dies eine Operation des Sachmittels dar:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Programmierer Stpe(Position)	übersetzt Programmquellen :Action	:Interaction	:Compiler Stpe(Resource)	Programmquellen in Objektdateien übersetzen :Operation

Als objektorientierter Ansatz erzwingt die OBA++ immer die Zuordnung von Aktivitäten zu Klassen. Eine isolierte Betrachtung von Aktivitäten ohne die Betrachtung, wer, was oder woran die Aktivität vollzogen wird, ist damit unmöglich.

Die Aktionsspalte der Skripte ist deshalb so wichtig, weil hier die Aktivität des Senders und damit die Absicht dokumentiert wird, mit der die Interaktion stattfindet. Diese Seite wird in allen anderen objektorientierten Ansätzen ignoriert, weil sie sich auf die Modellierung der Operationen (= der Empfängerseite der Interaktion) beschränken. Damit wird aber auch ignoriert, warum eine Interaktion überhaupt stattfindet. Bei der objektorientierten Modellierung von Geschäftsprozessen sind die Aktionen so wichtig, weil sie die Kunden-Seite der Interaktion beinhalten. Handelt es sich um Akteure, können auf diese Weise deren Aktivitäten erfasst werden, was bei der UML *per definitionem* ausgeschlossen ist.

Bei der Modellierung von Geschäftsprozessen werden häufig unterschiedliche Verben für die gleiche Tätigkeit verwendet.[1] Der Versuch, sich mit einer größeren Personenzahl auf ein einheitliches Vokabular zu einigen,[2] scheitert häufig. Außerdem werden dadurch häufig die Skripte unverständlich, da die Fachlichkeit bestimmter Anwendungsgebiete die Verwendung bestimmter Verben beinhaltet. Durch den Modus der Aktivität (vgl. Abschnitt 6.6.4) kann festgelegt werden, dass es sich beispielsweise bei den Verben *anlegen*, *erzeugen*, *eröffnen*, *erstellen* immer um einen

[1] Diese Verben werden in [Rupp 2001: S. 226 ff.] als *Prozesswörter* bezeichnet.

[2] Die ist eine Beobachtung des Verfassers.

Konstruktor handelt, durch den ein Objekt erzeugt wird. Die Aktivität würde deshalb das Prädikat `Mode(Constructor)` erhalten.

7.2.3 Zuordnungen

KRÜGER unterscheidet sechs Arten der Zuordnung.[1] Der Schwerpunkt der Organisation stellt die Verteilung von Aufgaben auf menschliche Aufgabenträger dar. Diese *personale Zuordnung* wird traditionell als *Aufbauorganisation* bezeichnet. Die Aufgaben lassen sich durch die durchzuführenden Verrichtungen und die zu bearbeitenden Objekte bestimmen. Die Verrichtungen sind nicht beliebig mit Objekten kombinierbar, sondern beziehen sich auf bestimmte Objekttypen.[2] Dies stellt nach KRÜGER die *sachlogische Zuordnung* dar. Durch die *instrumentale Zuordnung* wird festgelegt, welche Sachmittel zur Aufgabenerfüllung verwendet werden. Bei der Ausführung ihrer Aufgaben verwenden die Personen als Aufgabenträger Sachmittel.[3] Durch die *informationelle Zuordnung* wird schließlich festgelegt, welche Informationen notwendig sind, um Aufgaben zu erledigen. Die räumliche und zeitliche Gestaltung, durch die geregelt wird, was wann in welcher Reihenfolge zu erledigen ist, stellt die *lokale und temporale Zuordnung* dar, was gewöhnlich als *Ablauforganisation* bezeichnet wird.

Alle genannten Zuordnungen lassen sich in Skripten abbilden. Die personale Zuordnung einer Aufgabe zu einem Aufgabenträger entspricht der Operation einer Klasse (vgl. unten Interaktion a). Die im Rahmen der Aufgabe stattfindenden Verrichtungen an den zu bearbeitenden Objekten – die sachlogische Zuordnung – stellen Aktionen der Klasse dar. Dadurch wird festgelegt, dass die Verrichtung an dem Objekt vorgenommen werden kann (vgl. unten Interaktion b). Auch die instrumentale und informationelle Zuordnung wird durch Interaktionen abgebildet (vgl. Interaktion c und d). Die Modellierung der lokalen und temporalen Zuordnung wird in Abschnitt 7.3.6 dargestellt.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	a :Interaction	:Systemtechniker Stpe(Task Bearer)	Email-Server einrichten :Operation
	:Systemtechniker Stpe(Task Bearer)	installieren :Action	b :Interaction	:Email-Server Stpe(Resource)	wird installiert :Operation

[1] Vgl. [Krüger 1994: S. 15 ff.]. Vgl. auch [Kosiol 1962: S. 43], [Wild 1966].

[2] Dies entspricht den Gliederungsmerkmalen Verrichtung und Objekt in der Aufgabenanalyse von KOSIOL. Vgl. [Kosiol 1962: S. 49 – 53].

[3] Dies deckt sich mit dem betriebswirtschaftlichen Begriff Ressource bzw. Produktionsfaktor. Vgl. [Klepzig 1997: S. 85], [Staud 1999: S. 6].

	:Programmierer Stpe(Task Bearer)	übersetzt Programmquellen :Action	c :Interaction	:Compiler Stpe(Resource)	Programmquellen in Objektdateien übersetzen :Operation
	:Sachbearbeiter Stpe(Task Bearer)	sucht Kundendaten :Action	d :Interaction	:Kunden	liefert Kundendaten :Operation

Skript 171: Muster von Zuordnungen

Das objektorientierte Prinzip, Aktivitäten niemals isoliert, sondern immer nur im Zusammenhang mit Objekten zu betrachten, hat auf der einen Seite den Nachteil, dass eine Aufgabenanalyse, dass heißt die Auflösung einer komplexen Gesamtaufgabe in Teilaufgaben[1], in der OBA++ nur möglich ist, wenn gleichzeitig personale, sachlogische, instrumentale oder informationelle Zuordnungen gebildet werden.[2] Dadurch gewinnt man aber auf der anderen Seite den Vorteil, dass der Verteilungszusammenhang, also die Verteilung der Aufgaben auf Personen als Aufgabenträger[3], als Nebenprodukt der Prozessanalyse entsteht. Durch den in Kapitel 5 definierten Abbildungsprozess auf das Konzeptuelle Schema kann außerdem jederzeit festgestellt werden, welche Aufgaben und Transformationen eine Klasse durchführt und welche Verrichtungen von dieser Klasse an anderen Klassen durchgeführt werden.

7.2.4 Rollen

Neben Aufgabenträgern werden in Geschäftsprozessen bestimmte *Aufgabenprofile* wie zum Beispiel Verkaufssachbearbeiter, Kostenrechner oder Maschinenbediener beschrieben.[4] Dies geschieht durch Rollen.[5] Rollen werden insbesondere dann verwendet, wenn Aufgaben keinem Aufgabenträger direkt zugeordnet werden.[6] *„Der Begriff der Rolle bezeichnet einen bestimmten Mitarbeitertyp mit einer definierten Qualifikation und Kompetenz.“*[7] *„Rollen beschreiben die notwendigen Erfahrungen, Kenntnisse und Fähigkeiten, um Aktivitäten durchzuführen.“*[8] Sie

[1] Vgl. [Kosiol 1962: S. 42], [Gaitanides 1983: S. 23 ff.], [Schulte-Zurhausen 1999: S. 40 ff.].

[2] Allerdings wird im Rahmen der GPM keine Aufgabenanalyse durchgeführt. Eine objektorientierte Lösung dieses Problems wird in Abschnitt 7.3.9 demonstriert.

[3] Vgl. [Kosiol 1962: S. 77].

[4] RAUSCHECKER definiert Rolle als die Erwartungen, die an einen Rolleninhaber gestellt werden. Vgl. [Rauschecker 2000: S. 137].

[5] Vgl. [Rauschecker 2000: S. 118], [Scheer 1998: S. 57].

[6] Vgl. [Rauschecker 2000: S. 138].

[7] [Scheer 1998: S. 57]. Vgl. auch die ausführlichere Darstellung in [Galler 1997: S. 52 ff.].

[8] [Balzert 1998: S. 105].

umfassen Mengen von Stellen mit gleichen Kompetenzen bzw. einer gemeinsamen Teilmenge von Kompetenzen und definieren so organisatorische Funktionen.[1]

Der Begriff Rolle wird sowohl im Geschäftsprozess- als auch im Objektansatz (OA) verwendet. Dies darf aber nicht darüber hinwegtäuschen, dass beide Konzepte unterschiedlich sind. Die Rolle im Objektansatz ist *per definitionem* keine Klasse, sondern ein eigenständiger Metatyp, der im Unterschied zum Metatyp *CLASS* keine eigene Intension und damit auch keinen Zustand besitzt. Im Geschäftsprozessansatz werden dagegen Rollen verwendet, wenn Aufgaben keinem Aufgabenträger zugeordnet werden sollen oder können. Sie stellen Elemente der Aufbauorganisation dar, die eine eigene Intension besitzen, und sind deshalb als Klasse abzubilden. Daraus ergibt sich, dass zwischen den Rollen der OBA++ und denen der GPM zu differenzieren ist.

OBA++-Rollen sind deshalb bei der GPM nur dann zu verwenden, wenn damit kein Element der Aufbauorganisation abgebildet werden soll. Wenn beispielsweise alle Abteilungsleiter in der Rolle des `Unterschriftsberechtigten` auftreten können, ergibt das in der Skriptnotation:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	/Unterschriftsberechtigter :Abteilungsleiter Stpe(Position)	Auftrag unterzeichnen :Operation

Skript 172: Eine Klasse als Rollenfüller

Auf diese Weise ist abgebildet, dass keine Position `Unterschriftsberechtigter` existiert, prinzipiell `Abteilungsleiter` aber diese Rolle ausfüllen können. Ist eine bestimmte Person ein Rollenfüller, ergibt sich:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	Frau Müller /Frauenbeauftragte :Mitarbeiter Stpe(Position)	Beschwerde bearbeiten :Operation

Skript 173: Ein Objekt als Rollenfüller

Wird dagegen eine Rolle im Sinne der GPM, also als Element der Aufbauorganisation, modelliert, handelt es sich um eine Klasse. In Skript 168 wurde eine solche organisatorische Rolle als Klasse mit dem Stereotyp Organisationseinheit (`Org-Unit`) modelliert.[1]

[1] [Rupietta 1992: S. 26].

7.2.5 Flüsse

Nach HAIST und FROMM kann in allen Arten von Prozessen (Fertigungs-, Verwaltungs- und Entwicklungsprozesse)[2] ein „Fließen“ von Dingen zwischen den Organisationseinheiten beobachtet werden. Diese Dinge können materieller Natur sein oder informationellen Charakter besitzen.[3] Dabei legen die Ansätze zur GPM wie SOM[4], BPI[5] oder der Ansatz von RUMMLER und BRACHE[6] besonderen Wert auf die Modellierung des Informationsflusses. Bei der Modellierung der Informationsflüsse spielt es keine Rolle, dass sie an ein materielles Übertragungsmedium gebunden sind.[7]

Alle Flüsse werden in der OBA++ durch Interaktionsbeziehungen abgebildet. Die unterschiedlichen Arten der Flüsse werden in der Spalte Connection des Skripts durch das Prädikat `Flow()`, die unterschiedlichen Übertragungsmedien werden durch das Prädikat `Form()` in der gleichen Tabellenspalte festgehalten.

7.2.6 Beziehungen

Man unterscheidet bei der Betrachtung von Organisationen die Betrachtung der vergleichsweise dauerhaften *Struktur* und die Betrachtung der in der Organisation ablaufenden *Prozesse*. Die *Struktur* bezeichnet die im Zeitablauf relativ stabilen („statischen“) Eigenschaften der Elemente und das Beziehungsgefüge zwischen den Elementen.[8] Diese den Organisationsaufbau beschreibende Struktur ist ein komplexes Gefüge verschiedenartiger Beziehungen.[9] Prozesse vollziehen

---

[1] Auf die Einführung eines Stereotyps Rolle (`Role`) wird aus pragmatischen Gründen verzichtet, da dies nach Meinung des Verfassers hier eher zur Verwirrung beitragen würde. Die Definition weiterer Stereotypen stellt aber keine Veränderung des Metamodells dar und ist bei Bedarf jederzeit möglich.

[2] Auf die unterschiedlichen Arten von Prozessen wird in Abschnitt 7.3.1 eingegangen.

[3] [Haist 1989: S. 100]. SCHEER verwendet in der Methode ARIS außerdem Ziel- und Organisationsflüsse. Vgl. [Scheer 1998b: S. 23].

[4] Vgl. [Ferstl 1998].

[5] Vgl. [Harrington 1991: S. 94].

[6] Vgl. [Rummler 1991].

[7] [Haist 1989: S. 100].

[8] NORDSIECK spricht von einer organisatorischen Beziehungslehre. Vgl. [Nordsieck 1955: S: 77]. Vgl. auch [Wild 1966: S. 30].

[9] Vgl. [Kosiol 1962: S. 76].

sich in dieser verhältnismäßig unveränderlichen Grundstruktur.[1] Die Struktur der Organisation und die in der Organisation ablaufenden Prozesse stehen in einem wechselseitigen Abhängigkeitsverhältnis.[2] Sie bedingen sich gegenseitig und sind nur idealtypisch voneinander zu trennen.[3] Aus diesem Grund ist die simultane Modellierung von statischen Beziehungen und dynamischen Beziehungen sinnvoll.

Skripte ermöglichen sowohl die Betrachtung der Struktur wie auch die der Prozesse. Prozesse werden in den Skripten durch Interaktionsbeziehungen abgebildet, statische Beziehungen zwischen den Bausteinen von Geschäftsprozessen werden in den Skripten durch referentielle und taxonomische Beziehungen abgebildet.

Beispielsweise geschieht die im Rahmen der Abteilungsbildung stattfindende Zuordnung von Aufgabenträgern durch referentielle Beziehungen in der Spalte Connection des Skripts. Welche referentielle Beziehung (Association, Membership, Whole-Part, Containment, Aggregation) vorliegt, ist vom Einzelfall abhängig und kann nicht generell festgelegt werden. Dabei können durch die Prädikate der referentiellen Beziehungen die Wechselwirkungen zwischen den stattfindenden Prozessen und der Struktur abgebildet werden.[4]

[1] [Heinen 1985: S. 62]. KOSIOL bezeichnet mit Struktur die *gefügehafte* Ordnung der Glieder eines Ganzen. Vgl. [Kosiol 1962: S. 19]. Für eine Darstellung des Gesamtsystems der Aufbauorganisation nach KOSIOL siehe [Kosiol 1962: S. 171 ff.].

[2] Vgl. [Schulte-Zurhausen 1999: S. 35].

[3] Vgl. [Nordsieck 1955: S. 76]. Auch KOSIOL sah die getrennte Behandlung von Aufbau (Struktur) und Ablauf (Prozess) eigentlich nur als *„verschiedene Betrachtungsweisen für den gleichen einheitlichen Tatbestand."* [Kosiol 1962: S. 187 f.]. Ähnlich auch in [Picot 1995: S. 4], [Bullinger 1996: S. 8] und [Krüger 1994: S. 119]. Man kann deshalb die gleichrangige Betrachtung der Aufbau- und Ablauforganisation als ein wesentliches Grundprinzip der Organisationsgestaltung ansehen.

[4] Da der Fokus dieser Arbeit auf der Betrachtung der in der Organisation ablaufenden *Prozesse* und nicht auf der Modellierung der Struktur der Organisation liegt, wurden keine Stereotype für die Modellierung der üblicherweise in einer Organisation existierenden statischen Beziehungen entwickelt. Beispielsweise könnten die Stereotype komplementär, konkurrierend oder indifferent für die referentiellen Beziehungen zwischen Zielen eingeführt werden. Vgl. [Heinen 1985: S. 101 ff.]. Stereotype könnten auch für die zwischen Stellen existierenden Beziehungen definiert werden, um beispielsweise die hierarchische Struktur der Leitungsbeziehungen (vgl. [Heinen 1985: S. 59], [Kosiol 1962: S. 77, 100 ff.]) zu kennzeichnen.

7.2.7 Leistungen

Die an einem Geschäftsprozess beteiligten Aufgabenträger (Organisationseinheiten) stehen zueinander in Leistungsbeziehungen.[1] Die „*Leistungen sind die Ergebnisse (der Output) eines Prozesses, die an interne oder externe Kunden gehen. Empfänger einer Leistung ist ein anderer Prozess innerhalb oder außerhalb des Unternehmens. Eine Leistung kann materiell oder immateriell sein.*“[2] Zu den Leistungen gehören Güter, Zahlungen und Dienstleistungen, die durch Flüsse ausgetauscht werden.[3] Auch die Ergebnisse im indirekten Bereich werden als Leistungen aufgefasst.[4]

Der dem OA entlehnte Leistungsbegriff unterscheidet sich von dem der GPM. Während in der Geschäftsprozessorientierung die Leistung mit einem erstellten Output gleichgesetzt wird, entspricht im OA die Leistung der Operation, die ein Objekt durch Veranlassung eines anderen Objekts ausführt (vgl. Abschnitt 2.2.8), weil im Rahmen von Dienstleistungsprozessen häufig kein Output erzeugt wird.[5] Auch in der OBA++ besteht die Leistung darin, dass der Participant die ihm zugeordnete Operation ausführt. Dabei spielt es keine Rolle, ob ein Output erstellt wird, wie man am Hotelbeispiel in Skript 196 erkennen kann.[6] Die Spezifikation der Leistung geschieht dabei durch Verträge (vgl. Abschnitt 7.2.10).

Es spielt auch keine Rolle, ob es sich um eine Organisationseinheit, ein Geschäftsobjekt oder ein Sachmittel handelt, welches die Leistung ausführt. Per definitionem sind die Operationen aller Objekte Leistungen. Den im Rahmen der GPM besonders interessanten Fall von Leistungsbeziehungen zwischen Organisationseinheiten erkennt man daran, dass sowohl Initiator wie auch Participant entsprechende Stereotype (`Organization`, `Org-Unit`, `Action Bearer`, `Individual`, `Position`, `Task Bearer`) besitzen, die sie als Aufgabenträger oder Elemente der Aufbauorganisation auszeichnen.

---

[1] Vgl. [Scheer 1998b: S. 10, 13].

[2] [Österle 1995: S. 52]. Vgl. auch [Scheer 1998: S. 93], [Engelmann 1995: S. 45].

[3] Vgl. [Ferstl 1998: S. 17, 41, 60], [Scheer 1998b: S. 23]. „*Der Leistungsbegriff umfasst Sach- und Dienstleistungen, wobei zu den Dienstleistungen auch Informationsdienstleistungen zählen.*“ [Scheer 1998: S. 148].

[4] Vgl. [Scholz 1994: S. 63, 65].

[5] Typische Beispiele sind zum Beispiel der Personenverkehr oder eine Beratungsleistung. Vgl. [Corsten 1990: S. 28], [Schulte-Zurhausen 1999: S. 52].

[6] Durch die Leistung eines Hotels für einen Kunden, eine Übernachtung abzuwickeln, wird für den Kunden kein Output erzeugt.

7.2.8 Input-Output-Beziehungen

Durch Input-Output-Beziehungen[1] wird festgelegt, welcher Input in einen Prozess hineinfließt und welcher Output hinausfließt.[2] Dadurch wird u. a. der Fluss zwischen den Organisationseinheiten sichtbar.[3] Die Aktivitäten von Geschäftsprozessen verwenden den Input und erzeugen einen Output.[4] Jede materielle oder immaterielle Leistung, die sich aus den Aktivitäten ergibt, ist ein Output.[5] Dabei muss der Input den Anforderungen an den Lieferanten entsprechen.[6] Der Abnehmer einer Leistung muss die Anforderungen an den Output des Lieferanten festlegen. Diese Anforderungen müssen zwischen Abnehmer und Lieferanten abgestimmt werden.[7]

Die Input-Output-Beziehungen von Operationen werden in der OBA++ durch die Prädikate `In()` und `Out()` festgelegt. Dabei existieren mehrere Möglichkeiten. Die Input-Schnittstelle des Participants stellt immer das Prädikat `In()` seiner Operation dar. Die Output-Schnittstelle stellt nur dann das Prädikat `Out()` der Operation dar, wenn der Output an den Initiator geliefert wird:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :HiFi-Produzent Stpe(Organization) System(external)	bestellt Ware :Action	:Interaction	:Komponenten-hersteller Stpe(Organization)	liefere Ware :Operation In(:Bestellung) Out(:Lieferung)

Skript 174: Input-Output-Beziehung

In der OBA++ werden solche Input-Output-Beziehungen häufig in zwei Interaktionen zerlegt, wie man am Beispiel des Komponentenherstellers sieht:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Fertigung Stpe(Org-Unit)	ruft Bauteile ab :Action	:Interaction Flow(Information)	:Bauteillager Stpe(Org-Unit)	Bauteile zur Fertigung liefern :Operation

[1] Vgl. [Scholz 1994b: S. 39].

[2] Vgl. [Elgass 1993: S. 43] [Fischermanns 1997: S. 23], [Kueng 1995b], [Schulte-Zurhausen 1999: S. 59].

[3] Vgl. [Rummler 1991: S. 36].

[4] Vgl. [Schulte-Zurhausen 1999: S. 49 f., 60], [Rummler 1991], [Schmelzer 2002: S. 33].

[5] „*Die für einen Kunden erzeugten Produkte bilden den Output eines Prozesses.*“ [Fischermanns 1997: S. 23]. Vgl. auch [Kleinsorge 1994: S. 53]. Dabei sind auch Ausschuss, Abfälle und andere Ergebnisse der Leistungserstellung zu beachten. Vgl. [Mayer S. 38].

[6] Vgl. [Kleinsorge 1994: S. 54].

[7] Vgl. [Kleinsorge 1994: S. 53].

			Content(:Lagerabruf)		In(:Lagerabruf)
	:Bauteillager Stpe(Org-Unit)	liefert Bauteile :Action	:Interaction Flow(Material) Content(:Bauteile)	:Fertigung Stpe(Org-Unit)	fertigt Produkt :Operation In(:Bauteile)

Skript 175: Zerlegte Input-Output-Beziehung

Dies ist insbesondere dann der Fall, wenn zwischen unterschiedlichen Flüssen unterschieden werden soll oder wenn zwischen Input- und Output eine Reihe von Aktivitäten stattfinden, die durch zusätzliche Interaktionen detailliert dargestellt werden.

Wenn ein Output nicht an den Initiator, sondern an ein anderes Objekt geliefert wird, ist die Aufteilung in zwei Interaktionen notwendig. Wie man am Beispiel des Komponentenherstellers erkennen kann, ist dies insbesondere bei der Abfolge von Kunden-Lieferanten-Beziehungen (Abschnitt 7.2.9) der Fall.

Die Prädikate `In()` und `Out()` stellen die formalen Argumente der Operation dar (vgl. Abschnitt 6.6.1). Bei der OBA++ wird außerdem zwischen der Input-Output-Beziehung der Aktivität des Participants und dem aktuellen Fluss zwischen Initiator und Participant unterschieden. Der aktuelle Fluss wird durch die Prädikate `Content()` und `Answer()` abgebildet, wobei `Content()` den Fluss vom Initiator zum Participant und `Answer()` den Fluss in Gegenrichtung darstellt. Input-Output-Beziehungen werden immer nur auf der Participant-Seite festgelegt. Wenn für eine Aktion explizit ein In- und Output festgelegt werden muss, geschieht auch dies durch `Content()` und `Answer()`. Dies wird im folgenden Beispiel erkennbar, in dem der Sachbearbeiter die Leistung anbietet, Angebote zuzusenden, wenn er die Artikeldaten erhält. In diesem aktuellen Fall soll ein Angebot für die Artikel mit der Nummer 04812 und 1607 erstellt werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person	bittet um Angebot :Action	:Interaction Content(04812 :Artikelnummer, 1607 :Artikelnummer) Answer(:Angebot)	:Sachbearbeiter	liefere Angebot :Operation In(:Artikeldaten) Out(:Angebot)

Skript 176: Input-Output-Beziehung des Prozesses

Der Unterschied zwischen der geschäftsprozessorientierten Sichtweise und der OBA++ besteht darin, dass im Geschäftsprozessansatz Prozesse als EVA-Strukturen[1] betrachtet werden. Alles, was

[1] EVA: Eingabe–Verarbeitung–Ausgabe.

in den Prozess hineinfließt, ist seine Eingabe. Alles, was aus dem Prozess hinausfließt, ist seine Ausgabe. Von wem die Eingabe stammt und an wen die Ausgabe geliefert wird, ist dabei (zunächst) von nachrangiger Bedeutung.[1] Die OBA++ – wie auch alle anderen objektorientierten Ansätze – verwendet aber eine andere Sichtweise. Die Ein- und Ausgabe einer Operation bezieht sich immer nur auf den Initiator. Das Prädikat `In()` einer Operation umfasst nur die vom Initiator stammende Eingabe, so wie das Prädikat `Out()` nur die an den Initiator gelieferte Ausgabe enthält.

Die Begriffe *Quelle* und *Senke*, die bei der GPM häufig verwendet werden, sind nicht auf den OA übertragbar. Welches Objekt eine Quelle und welches Objekt eine Senke ist, wird durch die Prädikate `In()` und `Out()` festgelegt. Initiator und Participant können sowohl Quelle wie auch Senke sein. Alle übrigen Ein- und Ausgaben, die zur Ausführung der Operation notwendig sind, aber nicht vom Initiator stammen oder an den Initiator geliefert werden, sind nach dem Geheimnisprinzip ein Internum des Objekts und damit kein Element der Input-Output-Spezifikation der aktuellen Operation. Insbesondere die Verwendung von Objektinterna über `In()` und `Out()` zu modellieren, stellt einen groben Verstoß gegen den OA dar. Wenn ein Objekt zur Ausführung einer Operation neben dem Input des Initiators weitere Objekte benötigt, werden diese durch separate Interaktionen modelliert. Wenn ein Objekt auf seine Interna (Attribute) zugreift, wird dies in der OBA++ durch interne Beziehungen abgebildet. Die OBA++ betrachtet also nicht nur die zwischen Objekten fließenden Ströme, sondern außerdem, von wem die Ströme stammen und an wen die Ströme gehen.[2] Die Anforderungen, die der Participant an den Input und der Initiator an den Output hat, sind Bestandteil des Vertrags (siehe Abschnitt 7.2.10) und werden immer auf der Seite des Participants festgelegt.

Ein weiterer Unterschied zwischen dem OA und der GPM besteht in der Verwendung des Begriffs *Schnittstelle*. Übereinstimmend wird er in beiden Ansätzen verwendet, um die ggf. gedachte Grenze von etwas zu beschreiben. In der GPM sind Schnittstellen in erster Linie die Schnittstellen von Organisationseinheiten. Darüber hinaus ist auch schon ein einzelnes Ausgabe- oder Eingabeargument einer Operation eine Schnittstelle. Im OA wird der Begriff Schnittstelle wesentlich allgemeiner verwendet, da alle Klassen Schnittstellen besitzen. Es handelt sich dabei um eine Ansammlung mehrerer Operationen, die die Klasse anderen Klassen zur Verfügung stellt.[3]

---

[1] In der Informatik wird dies als funktionale Sichtweise betrachtet. Die strukturierten Ansätze verwenden diese Metapher.

[2] Man beachte die in der OBA++ zusätzlich geschaffene Möglichkeit, externe Referenzen durch das Prädikat Ref() anzugeben. Vgl. Abschnitt 6.6.8.

[3] Vgl. [Booch 1999: S. 157 f.]. Es wurde darauf verzichtet, im Metamodell des Konzeptuellen Schemas einen dezidierten Konzeptknoten *INTERFACE* zu definieren, da die von Klassen losgelöste Modellierung von Schnittstellen im Rahmen der GPM als nicht sinnvoll betrachtet wurde. Wie in Abschnitt 4.4.3 ausgeführt wurde, kann das Metamodell aber bei Bedarf entsprechend erweitert werden.

7.2.9 Kunden-Lieferanten-Beziehungen

Die Schnittstellen zwischen Organisationseinheiten werden als *Kunden-Lieferanten-Beziehungen* (KL-Beziehung) aufgefasst.[1] Dies stellt einen der Kernpunkte der Geschäftsprozessorientierung dar.[2] Jeder Prozess und jede Aktivität hat danach mindestens eine Quelle (Sender, Lieferant) und mindestens eine Senke (Empfänger, Kunde).[3] Prozesse sind damit mittelbar oder unmittelbar kundenbezogen.[4] Jeder Empfänger einer Leistung ist ein Kunde.[5] Jeder Ersteller einer Leistung ist ein Lieferant.[6]

Nach dem sog. NOAC-Prinzip (*Next Operation as Customer*) werden nachgelagerte Prozessschritte als Kunden vorgelagerter Prozessschritte betrachtet.[7] Innerhalb des Prozessverlaufs wechselt sich also die Rolle *Kunde* und *Lieferant* ab.[8] Auf diese Weise stellt ein Geschäftsprozess eine Abfolge von Leistungsbeziehungen zwischen Kunden und Lieferanten dar.[9]

Die objektorientierte Interaktionsbeziehung mit der durch die Operation definierten Leistung des Participants stellt im Sinne der GPM implizit immer eine KL-Beziehung dar. Dieser Kernpunkt der

---

[1] Vgl. [Scholz 1994: S. 65], [Rummler 1991], [Völkner 1998: S. 12], [Wassermann 2001: S. 139].

[2] Vgl. [Scholz 1994: S. 166], [Harrington 1991: S. 39, 72 f.], [Koch 1997: S. 66], [Brenner 1995b: S. 22 f.], [Kleinsorge 1994: S. 52], [Scherr 1993: S. 82], [Davenport 1993: S. 84], STÖGER in [Bullinger 1996: S. 379 ff.], [Völkner 1998: S. 12], [Schreyögg 1990: S. 72], [Schmelzer 2002: S. 40 f.].

[3] „*Ein Prozess, aber auch jede einzelne Aktivität, hat immer mindestens einen Lieferanten, von dem die Eingabe kommt, und mindestens einen Kunden, der das um die Bearbeitung angereicherte Arbeitsergebnis, die Ausgabe, erhält. Die Eingabe kann beispielsweise ein Schriftstück sein, die Verarbeitung im Prozess ‚prüfen und unterschreiben' und die Ausgabe ‚das unterschriebene Schriftstück'.*" [Haist 1989: S. 94]. Vgl. auch [Schulte-Zurhausen 1999: S. 50, 59], [Corsten 1997b: S. 17] [Scholz 1994: S. 166].

[4] Vgl. [Schulte-Zurhausen 1999: S. 51].

[5] [Engelmann 1995: S. 45]. Siehe auch die Darstellung in [Bullinger 1996: S. 379 ff., 467 f.], [HeilmannM 1997: S. 100], [Schulte-Zuhausen 1999: S. 50, 59 ff.].

[6] Vgl. [Schulte-Zuhausen 1999: S. 50, 59 ff.].

[7] „*Mit ‚Lieferant' und ‚Kunde' sind sowohl der vorgelagerte bzw. nachgelagerte Prozess (bzw. die Aktivität) gemeint als auch die entsprechende betriebliche Funktion (Abteilung, Bereich).*" [Haist 1989: S. 94]. Vgl. auch [Klepzig 1997: S. 74 f.], [Eversheim 1995: S. 36], [Elgass 1993: S. 43].

[8] Vgl. [Elgass 1996: S. 129], [Striening 1988: S. 64], [Rummler 1991: S. 22].

[9] Zum innovativen Gehalt dieses Konzeptes sei auf die Aussage von NORDSIECK hingewiesen, dass der Betriebsprozess einen geschlossenen Kreis darstellt, der beim Kunden beginnt und beim Kunden endet. Damit wird die Kunden-Lieferanten-Sicht implizit vorweggenommen. Vgl. [Nordsieck 1972: Sp. 12]. Wirklich neu scheint nur der Gedanke zu sein, auch Verwaltungsprozesse als Leistungen für Kunden zu betrachten. Dies findet sich bei NORDSIECK nicht – er trennt den Verwaltungsbereich vom Bereich der Leistungserstellung. Vgl. [Nordsieck 1972: Sp. 30 f.].

Geschäftsprozessorientierung entspricht also dem in Abschnitt 1.4 formulierten *Primat der Interaktionsbeziehung* des Objektansatzes: Prozesse sind – objektorientiert betrachtet – Abfolgen von Interaktionsbeziehungen und außerdem – geschäftsprozessorientiert betrachtet – Abfolgen von KL-Beziehungen.

Da zu einer Interaktion immer ein Initiator und ein Participant gehört, gibt es auch keine Interaktionen ohne Lieferanten und Kunden. Hierbei existiert aber ein wesentlicher Unterschied. Im OA ist die Kunden-Lieferanten-Beziehung genau umgekehrt definiert (vgl. Abschnitt 2.2.8). Danach ist der Participant der Lieferant und der Initiator ist der Kunde. Der Initiator fragt beim Participant eine Leistung nach. Führt der Participant im Rahmen seiner Operation (= Leistung) Folgeaktivitäten aus, wird er zum Initiator und damit zum Kunden eines anderen Lieferanten. Damit stellen auch die Abfolgen von Interaktionsbeziehungen im Skriptmodell Abfolgen von KL-Beziehungen dar. Aber aus der Sicht des Geschäftsprozessansatzes ist die Beziehung „verdreht“. Ob darunter tatsächlich die Verständlichkeit leidet, kann letztendlich weder widerlegt noch bewiesen werden. Nach Meinung des Verfassers ist das NOAC-Prinzip nicht universell anwendbar, wie das folgende Beispiel zeigt, in dem der Lieferant `Sachbearbeiter` zum Kunden der Versandabteilung wird, da er die Kundendaten benötigt. Dabei wurden die Rollen Kunde und Lieferant zur Verdeutlichung explizit modelliert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	/Lieferant :Sachbearbeiter Stpe(Position)	liefere Angebot :Operation In(:Artikeldaten)
/Kunde :Sachbearbeiter Stpe(Position)	Kundendaten ermitteln :Action	:Interaction	/Lieferant :Versand Stpe(Org-Unit)	liefere Kundendaten :Operation In(:Kundenname) Out(:Kundendaten)

Skript 177: Abfolge von KL-Beziehungen; dem OA folgend

Verwendet man dagegen das NOAC-Prinzip, erhält man die im folgenden Skript dokumentierte kontraintuitive Sicht. Auch hier wurden die Rollen Kunde und Lieferant zur Verdeutlichung wieder explizit modelliert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Lieferant :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	/Kunde :Sachbearbeiter Stpe(Position)	liefere Angebot :Operation In(:Artikeldaten)
/Lieferant :Sachbearbeiter Stpe(Position)	Kundendaten ermitteln :Action	:Interaction	/Kunde :Versand Stpe(Org-Unit)	liefere Kundendaten :Operation In(:Kundenname) Out(:Kundendaten)

Skript 178: Abfolge von KL-Beziehungen; dem NOAC-Prinzip folgend

Die Lösung des Problems besteht in der OBA++ darin, den Interaktionsmodus zu dokumentieren.[1] Bittet der Initiator um eine Leistung, erhält die Interaktion den Modus `Request`. Der Kunde ist der Initiator:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction Mode(Request)	:Sachbearbeiter Stpe(Position)	liefere Angebot :Operation In(:Artikeldaten)

Skript 179: Kennzeichnung der KL-Beziehung durch den Interaktionsmodus (1)

Wenn die Initiative vom Lieferanten ausgeht, erhält die Interaktion den Modus `Supply`. In diesem Fall ist der Kunde der Participant:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Sachbearbeiter Stpe(Position)	bietet Produkt an :Action	:Interaction Mode(Supply)	:Person System(external)	kauft :Operation In(:Produkt) Out(:Geld)

Skript 180: Kennzeichnung der KL-Beziehung durch den Interaktionsmodus (2)

Darüber hinaus wird der Begriff der KL-Beziehung im Bereich der Objektorientierung wesentlich allgemeiner verwendet. Eine KL-Beziehung im Sinne der GPM liegt nur dann vor, wenn die beteiligten Objekte Personen, Organisationen oder Organisationseinheiten, also im entferntesten Sinne Aufgabenträger, sind. Die OBA++ dagegen erweitert die KL-Beziehung auf alle Objekttypen. Die folgende Interaktion ist im Sinne der OBA++, nicht aber im Sinne der GPM eine KL-Beziehung, da es sich bei `Artikelstamm` um keine Organisationseinheit handelt:

[1] Vgl. Abschnitt 6.2.10.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Sachbearbeiter Stpe(Position)	suchen :Action	:Interaction Content(:Artikelnummer) Answer(:Artikeldaten)	:Artikelstamm	finden :Operation In(:Artikelnr.) Out(:Artikeldaten)

Skript 181: Eine OBA++-KL-Beziehung

Unabhängig davon, ob man die objektorientierte oder geschäftsprozessorientierte Sichtweise verwendet, hat die durchgängige Interpretation von Interaktionen als Kunden-Lieferanten-Beziehungen unter Umständen überraschende Konsequenzen. Im folgenden Beispiel wird der Geschäftsprozess `liefere Angebot` in der letzten Zeile des Skripts durch die Operation `Person::annehmen Angebot` abgeschlossen (der vollständige Ablauf findet sich in Skript 191), durch die der Kunde das Angebot empfängt. Da das Eintreffen des Angebots ein Ereignis darstellt, auf das der Kunde reagiert, wird er zum Participant der Interaktion. In dieser Situation liegt tatsächlich der Fall vor, dass der Participant der Kunde ist. Als Ergebnis kann festgestellt werden, dass es vom Einzelfall abhängig ist, ob – im Sinne der Geschäftsprozessorientierung – der Participant in einer Interaktion die Rolle des Kunden oder des Lieferanten übernimmt. Eine sinnvolle Lösung besteht in der Verwendung von Rollen, wenn die KL-Beziehungen gemäß der Geschäftsprozessorientierung explizit abgebildet werden sollen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	:Sachbearbeiter Stpe(Position)	liefere Angebot :Operation Goal(Kunde nimmt Angebot an) In(:Artikeldaten) Out(:Angebot)
	...	...	...	...	...
7.	:Angebot	eintreffen :Action	:Interaction	/Kunde :Person Stpe(Individual) System(external)	annehmen Angebot :Operation Goal(Angebot wird angenommen) In(:Angebot) Pre(Angebot entspricht den Erwartungen) Post(Angebot erhalten)

Skript 182: Kennzeichnung des Kunden durch Rollenbezeichnungen

7.2.10 Verträge

Die Leistungsbeziehungen zwischen Kunde und Lieferant mit ihren Anforderungen an den Input und Output werden über Schnittstellen und *Verträge*[1] definiert. Die „*Erkennung und Beschreibung der Schnittstellen ermöglicht die Beurteilung der Input-/Outputbeziehungen und im weiteren Sinne dieser Vorgehensweise auch die Vereinbarung und Festlegung von ‚Kunden/-Lieferanten-beziehungen'. Die Beschreibung dessen, was man von seinem Lieferanten erwartet (was, wann, wie, wo) und dessen Zustimmung, ist gleichbedeutend mit einem Vertrag, der zwischen zwei Parteien geschlossen wird. Eine solche Beschreibung dient nicht nur der Präzisierung der Anforderungen und Verpflichtungen, sondern bietet darüber hinaus auch die Sicherheit der Kontinuität im Falle des Personalwechsels oder der Durchsetzung von Änderungen des Ablaufs.*"[2]

Die Vertragsbedingungen zwischen Initiator und Participant werden durch die Vor- und Nachbedingungen (Prädikate `Pre()` und `Post()`) der Operation des Participants festgelegt (vgl. Abschnitt 6.6.5, 6.6.9). Das Prädikat `Pre()` enthält die durch den Vertrag festgelegten Erwartungen an die Vorleistung des Initiators. Dies enthält auch die Festlegung an die spezifischen Eigenschaften des Inputs des Initiators. Das Prädikat `Post()` legt die Effekte der Operation und damit die Leistung des Participants fest. Dies umfasst die Festlegung an die spezifischen Eigenschaften des vom Participant gelieferten Outputs. Zwei Punkte führen dazu, dass das Vertragsmodell des Objektansatzes über das der GPM hinausgeht.

- Nicht nur die Anforderungen an den Input werden durch die Vorbedingung formuliert, sondern allgemein können alle Einschränkungen dokumentiert werden, die gelten müssen, damit die durch die Nachbedingung dokumentierte Leistung erstellt werden kann. Bei der Nachbedingung verhält es sich ähnlich. Es kann nicht nur angegeben werden, welche spezifischen Eigenschaften der Output der Operation erfüllen muss, sondern auch alle übrigen Effekte der Ausführung der Operation können durch die Nachbedingung dokumentiert werden, unabhängig davon, ob sie für den Initiator eine Rolle spielen.
- Bei der GPM werden nur zwischen Organisationseinheiten Verträge geschlossen. Im OA werden zwischen allen Objekten in Interaktionen Verträge verwendet.

---

[1] Auch als *Leistungsvereinbarung*, *Schnittstellenspezifikation* oder *Geschäftsprozess-Vereinbarung* bezeichnet. Vgl. [Schmelzer 2002: S. 91 f.], [Schulte-Zurhausen 1999: S. 60, 93], [Haist 1989: S. 95]. Auch MORABITO, SACK und BHATE spezifizieren Verträge durch Vor- und Nachbedingungen. Vgl. [Morabito 1999: S. 180 ff.].

[2] [Striening 1988: S. 170 f.].

Ein weiterer zu beachtender Umstand betrifft die Unteraufträge. Bei der GPM werden die durch Delegation entstehenden Unteraufträge (vgl. Abschnitt 2.2.8) völlig anders aufgefasst, als dies in der objektorientierten Softwareentwicklung (ooSe) üblich ist. Obwohl in einem Vertrag nur Anforderungen an die Leistungsbeziehung zwischen Kunde und Lieferant dokumentiert werden sollen, werden in der GPM auch die Anforderungen der Unteraufträge einbezogen. Dies wird als *einbezogener Unterauftrag* bezeichnet und führt zu einem Vertrag wie in folgendem Beispiel:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:AL	veranlasst Kündigungsschreiben an einen Kunden :Action	:Interaction	:Schreibkraft	erstellt Kündigungsschreiben :Operation In(:Kundenname, :Grund) Out(:Kündigungsschreiben) Vis(qualified) Exp(:AL, :Geschäftsführer) Concurrency(sequential) Pre(Schreibkraft ist verfügbar; Kundenname ist vorhanden; Briefpapier ist verfügbar) Post(Kündigungsschreiben ist erstellt)

Skript 183: Ein einbezogener Unterauftrag

Die erste Klausel der Vorbedingung ist korrekt. Das „angesprochene“ Objekt vom Typ Schreibkraft wird den Auftrag, das Kündigungsschreiben zu erstellen, nur dann annehmen, wenn es verfügbar ist. Der Abteilungsleiter kann sich nur dann darauf verlassen, ein Kündigungsschreiben zu erhalten, wenn er sicherstellt, dass die Schreibkraft verfügbar ist. Die zweite Klausel mag dagegen intuitiv gesehen richtig sein, ist aber mit Sicherheit keine Vorbedingung, die üblicherweise der Initiator sicherstellt. Sehr deutlich wird dies bei der dritten Klausel. Es ist kaum anzunehmen, dass der Abteilungsleiter sicherstellen muss, dass Briefpapier vorhanden ist. Tatsächlich handelt es sich um Vertragsklauseln der Unteraufträge:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:AL	veranlasst Kündigungsschreiben an einen Kunden :Action	:Interaction	:Schreibkraft	erstellt Kündigungsschreiben :Operation In(:Kundenname, :Grund) Out(:Kündigungsschreiben) Vis(qualified) Exp(:AL, :Geschäftsführer) Concurrency(sequential) Pre(Schreibkraft ist verfügbar) Post(Kündigungsschreiben ist erstellt)
1.1.	:Schreibkraft	sucht die Kundendaten :Action	:Interaction	:Kundendatei	Kundendaten ermitteln :Operation In(:Kundename) Out(:Kundendaten) Pre(Kundenname ist in Kundendaten vorhanden) Post(Kundendaten sind bekannt)

1.2.	:Schreibkraft	entnimmt Briefpapier :Action	:Interaction	:Briefpapier	benötigte Anzahl Bögen abgeben :Operation In(:Anzahl) Out(:Briefpapierbogen) Pre(Briefpapier ist verfügbar) Post(Briefpapier ist abgegeben)
1.3.	...	...	...	...	...

Skript 184: Ausformulierte Unteraufträge

Einbezogene Unteraufträge sind insbesondere zum Beginn der Untersuchung und Modellierung von Geschäftsprozessen häufig vorzufinden, weil die tatsächlichen Leistungsbeziehungen zwischen Initiator und Participant noch nicht vollständig und präzise erfasst sind. Je detaillierter die Abläufe durch Interaktionen erfasst werden, desto präziser kann zwischen Vertrag und zugehörigen Unteraufträgen unterschieden werden.

Durch den qualifizierten Export (Prädikat `Exp()` in Verbindung mit dem Term `Vis(qualified)`) kann außerdem präziser als in anderen Modellierungsansätzen festgelegt werden, wer eine Leistung bzw. eine Aufgabe von einem Objekt anfordern darf. Dies wird in Skript 183 deutlich: Ein Kündigungsschreiben veranlassen kann nur ein Abteilungsleiter oder ein Geschäftsführer.

7.2.11 Gleichwertigkeit von internen und externen Beziehungen

Die Prozessorientierung unterscheidet sich von der Ablauforganisation der Organisationslehre dadurch, dass Prozesse prinzipiell stellen-, funktions-, abteilungs- und organisationsübergreifend betrachtet werden.[1] *„Bisher orientierte sich die Abgrenzung der Prozesse an der Aufbauorganisation. Prozesse beginnen und enden an den Grenzen einer Abteilung oder eines Unternehmensbereichs, obwohl ein logisch zusammenhängender Ablauf wie zum Beispiel die Auftragsabwicklung eine ganze Reihe von Abteilungen vom Verkauf über die Logistik bis zum*

[1] Nach HARRINGTON „durchfließt“ der Prozess horizontal die Organisation. Vgl. [Harrington 1991: S. 13]. Vgl. auch [Schulte-Zurhausen 1999: S. 45], [Gaitanides 1994b: S. 4], [Rummler 1991: S. 16 f., 22 f.], [Schäl 1996: S. 14], [Elgass 1993: S. 43], [Kleinsorge 1994: S. 51], [Österle 1995: S. 19], [Bullinger 1996: S. 96], [HeilmannM 1997: S. 106], [Alt 2000], [Davenport 1993: S. 84], [Oberweis 1997]. Dies unterscheidet die Prozessorientierung von der klassischen Ablauforganisation. Die Ablauforganisation setzt erst an, wenn die Stellenbildung abgeschlossen ist. Dies wird auch als *Primat der Aufbauorganisation* bezeichnet. Durch die auf KOSIOL beruhende Reihenfolge, zuerst den Aufbau zu schaffen und danach die Abläufe zu gestalten [Kosiol 1962: S. 189], werden aber mögliche Prozessstrukturen übersehen. Vgl. hierzu auch [Wild 1966: S. 126] und die differenzierte Betrachtung des Problems in [Schmelzer 2002: S. 102 ff.].

Rechnungswesen umfasst und damit „quer" zur Aufbauorganisation liegt. [...] Aus diesem Grund sollte sich die Abgrenzung der Prozesse alleine am externen bzw. internen Kunden ausrichten."[1]

Die prozessorientierten Ansätze betrachten also die Schnittstellen zwischen zwei organisatorischen Einheiten als interne KL-Beziehungen.[2] Ein Kunde ist auch jeder unternehmensinterne Nachfrager bzw. Abnehmer einer (Zwischen-)Leistung.[3] Kunden innerhalb des Unternehmens sind interne Kunden, Kunden außerhalb des Unternehmens sind externe Kunden.[4] Mitarbeiter sind Kunden ihres Vorgängers und Lieferanten ihres Nachfolgers.[5] Auf diese Weise wird eine Organisation als ein Geflecht von internen und externen KL-Beziehungen betrachtet.[6] *„Kern der Prozessorientierung ist es gerade, dass interne und externe Beziehungen grundsätzlich als gleichwertig angesehen werden und sowohl die interne Übergabe eines Vorproduktes vom Einkauf in der Fertigung als auch die Abgabe eines Fertigprodukts an den Kunden als Schnittstelle verstanden werden. An jeder Schnittstelle muss der Leistungsaustausch zwischen Kunden und Lieferanten abgestimmt werden. Der Lieferant sollte wissen, was der Kunde erwartet, und der Kunde muss wissen, was der Produzent kann oder will."*[7]

Sowohl interne wie auch externe KL-Beziehungen werden durch Interaktionen abgebildet. Dabei stellen alle KL-Beziehungen zwischen Objekten, die nicht explizit durch das Prädikat `System()` als Umweltobjekte gekennzeichnet sind, interne Beziehungen dar. Im Beispiel des Komponentenherstellers sind die Interaktionen mit dem HiFi-Produzenten, dem Bauteil-Produzenten und den Banken durch das Prädikat `System()` als externe KL-Beziehung gekennzeichnet.[8]

[1] [Hess 1995: S. 481]. Vgl. auch [Schmelzer 2002: S. 35].

[2] Vgl. [Bullinger 1993: S. 106], [Bullinger 1996: S. 91], [Rosemann 1996: S. 196], [Rummler 1991: S. 62], [Völkner 1998: S. 12], GAIROLA in [Bullinger 1996: S. 467 ff.], [Pfitzinger 1997: S. 14].

[3] „*Kunden sind alle Personen oder Organisationseinheiten, die Leistungen (Produkte oder Dienstleistungen) vom betrachteten Prozess empfangen, unabhängig davon, ob sie diese ‚bezahlen' oder nicht.*" [Fischermanns 1997: S. 24]. Vgl. auch [Schulte-Zurhausen 1999: S. 49], [Hinterhuber 1994: S. 63], [Kleinsorge 1994: S. 52], [Hammer 1994: S. 57], [Picot 1996: S. 203].

[4] Vgl [Engelmann 1995: S. 45], [Fischermanns 1997: S. 24], [Schulte-Zurhausen 1999: S. 59].

[5] [Eversheim 1995: S. 36].

[6] „*Somit lässt sich ein Unternehmen als komplexes, offenes System von Prozessen darstellen, die eine Vielfalt von Beziehungen zwischen internen und externen Lieferanten und Kunden enthalten.*" [Haist 1989: S. 94]. Vgl. auch [Corsten 1997b: S. 22, 23].

[7] [Scholz 1994: S. 72].

[8] Vgl. Abschnitt 9.

7.3 Struktur von Geschäftsprozessen

7.3.1 Differenzierung von Prozesstypen

Der Begriff *Geschäftsprozess* wird in der Literatur in der Regel sehr allgemein verwendet. Eine präzise Definition ist bis heute nicht vorhanden.[1] Exemplarisch sei hier die Definition von SCHEER genannt: „*Allgemein ist ein Geschäftsprozess eine zusammengehörende Abfolge von Unternehmensverrichtungen zum Zweck einer Leistungserstellung. Ausgang und Ergebnis des Geschäftsprozesses ist eine Leistung, die von einem internen oder externen ‚Kunden' angefordert und abgenommen wird.*"[2] KRÜGER bezeichnet als *kritische Geschäftsprozesse* jene Prozesse, die bei einem Markpartner beginnen oder enden, mehrere Funktionen oder Teilfunktionen umspannen und sich auf ein gemeinsames Objekt richten. Sie lassen sich durch klar definierte Eingangs- und Ausgangsobjekte voneinander unterscheiden.[3] Diese Definition beschränkt sich auf den Aspekt der Kunden-Lieferanten-Beziehung. Andere Autoren verwenden präzisere Unterscheidungen:

DAVENPORT und SHORT differenzieren zwischen Produktions- und Transportprozessen auf der einen Seite sowie Verwaltungsprozessen auf der anderen Seite.[4] Auch HARRINGTON unterscheidet zwischen physischen Produktionsprozessen und Dienstleistungsprozessen, die er als Geschäftsprozesse bezeichnet: „*Production process. Any process that comes into physical contact with the hardware or software that will be delivered to an external customer, up to the point the product is packaged (e.g., manufacturing computers, food preparation for mass customer consumption, oil refinement, changing iron ore into steel). It does not include the shipping and distribution process. Business process. All service processes (e.g., order process, engineering change process, payroll process, manufacturing process design).*"[5] Geschäftsprozesse sind demnach die Prozesse, durch die ein Ergebnis entsteht, welches kein physisches Produkt darstellt.[6]

---

[1] STAUD spricht auch von der erfolglosen Suche nach der formalen Grundlage des Begriffs. Vgl. [Staud 1999: S. 6]. Zu den unterschiedlichen Definitionen des Begriffs Geschäftsprozess siehe [Wolff 1998].

[2] [Scheer 1998b: S. 3].

[3] Vgl. [Krüger 1994: S. 121]. SCHULTE-ZURHAUSEN bezeichnet diese Prozesse als *primäre Geschäftsprozesse*, von denen er die *Schlüsselprozesse* abgrenzt, die maßgeblich für die Kundenzufriedenheit sind. [Schulte-Zuhausen 1999: S. 83, 86]. Siehe auch die von HARRINGTON genannten Kriterien für kritische Geschäftsprozesse in [Harrington 1991: S. 36 ff.]. Nicht eingegangen wird hier auf die Definition von PICOT, nach der Prozesse Tätigkeitsfolgen sind, die Kundenwerte schaffen. Vgl. [Picot 1995: S. 2].

[4] Vgl. [Davenport 1993], [Engelmann 1995: S. 50].

[5] Vgl. [Harrington 1991: S. 9].

[6] Unklar bleibt dabei, welche Sonderrolle die Versand- und Distributionsprozesse erhalten sollen.

HEINEN unterscheidet Informationsprozesse, Güter- und Geldprozesse.[1] Dabei überlagern die Informationsprozesse die Güter- und Geldprozesse.[2] SCHULTE-ZURHAUSEN differenziert multikriteriell zwischen (1) materiellen und informationellen Prozessen, (2) operativen Prozessen und Managementprozessen sowie zwischen (3) Primär-, Sekundär- und Innovationsprozessen:[3] Materielle Prozesse beinhalten dabei körperliche Vorgänge an physisch real existierenden Objekten. Informationelle Prozesse erstrecken sich auf den Austausch oder die Verarbeitung von Informationen. Auch die Handhabung von materiellen Informationsträgern (Akten, Formulare etc.) wird dieser Kategorie zugerechnet. Operative Prozesse erstellen materielle oder informationelle Leistungen. Managementprozesse planen und kontrollieren Ziele und Maßnahmen. Primärprozesse dienen der eigentlichen Erstellung, Vermarktung und Betreuung von Produkten oder Dienstleistungen. Bei ihnen ist der Bezug zum (externen) Kunden signifikant. Durch Sekundärprozesse wird die Infrastruktur für Primärprozesse bereitgestellt.[4] Dazu gehört die Bereitstellung und Pflege von Potentialfaktoren wie auch die Aus- und Weiterbildung von Mitarbeitern.[5] Sie sind mit den primären Prozessen durch interne Kunden-Lieferanten-Beziehungen verknüpft. Durch Innovationsprozesse werden neue Produkte, Verfahren oder Strukturen entwickelt.[6]

Grundsätzlich wird also (siehe (1)) zwischen materiellen und informationellen Prozessen unterschieden.[7] Informationelle Prozesse sind insbesondere Büro- und Verwaltungstätigkeiten, deren Hauptmerkmal (und Objekt) der Umgang mit Daten und Informationen ist.[8] „*Das Wesen der Büroarbeit ist die Erzeugung und Verwendung von Information.*“[9] Dies beinhaltet das Sammeln, Aufbereiten, Auswerten und Weiterleiten von Daten und Informationen.[10] Diese informationsverarbeitenden Verwaltungsprozesse werden von STRIENING als typische Geschäftsprozesse

---

[1] [Heinen 1985: S. 62].

[2] [Heinen 1985: S. 68].

[3] [Schulte-Zurhausen 1999: S. 51 ff.].

[4] Ähnlich auch in [Schmelzer 2002: S. 47 f., 117 ff.] zu finden.

[5] Das sind zum Beispiel Infrastruktur, Personalwirtschaft, Technologieentwicklung, Beschaffung. Vgl. [Krüger 1994: S. 122].

[6] Vgl. [Schulte-Zurhausen 1999: S. 83].

[7] Vgl. auch [Elgass 1993: S. 43], [Schäl 1996].

[8] Vgl. [Striening 1988: S. 2, 34]. HOYER untersucht die Unterschiede zwischen Büro- und Verwaltungstätigkeiten [Hoyer 1988: S. 36 ff.], auf die hier nicht eingegangen wird. Hier werden die Begriffe als Synonyme verwendet.

[9] Vgl. [Raster 1994: S. 125]. Auch dieser Aspekt, der sehr gern als innovatives Element des Geschäftsprozessansatzes bezeichnet wird, wurde schon in der Betriebswirtschaftslehre untersucht. Schon SZYPERSKI hat 1955 die Bearbeitung *geistiger Objekte*, im weiteren Verlauf als *betriebliche Daten* bezeichnet, als Gegenstand der Büroarbeit bezeichnet. Vgl. [Szyperski 1961: S. 97].

[10] Vgl. [Striening 1988: S. 37].

bezeichnet.[1] Der Vorgang des Informationsaustauschs, die Kommunikation, hat dabei eine besondere Bedeutung, weil 60 % der Büroarbeit Kommunikation ist.[2]

Zunächst ist festzustellen, dass der Begriff Geschäftsprozess völlig unterschiedlich verwendet wird. Der eine Autor verwendet Geschäftsprozess als Sammelbegriff, der andere als speziellen Fall. Da es nicht die Aufgabe dieser Arbeit ist, eine weitere Definition zu entwickeln, deren Verwendbarkeit sich erst in der Praxis bewähren müsste, soll stattdessen untersucht werden, ob das entwickelte Instrumentarium für die unterschiedlichen Sichtweisen geeignet ist.

Folgt man der Definition von SCHEER, ist jede Abfolge von Interaktionen ein Geschäftsprozess, da diese immer aus Beziehungen zwischen internen oder externen Kunden und Lieferanten bestehen. Die von KRÜGER als kritisch bezeichneten Geschäftsprozesse sind daran zu erkennen, dass in einem Skript ein Initiator oder Participant für das Prädikat `System()` einen Term hat, der ihn als Umweltobjekt kennzeichnet. Ist dies der Fall, spielt es keine Rolle, um welche Art von Objekt (Organisation, Einzelperson, Stelle, Organisationseinheit, externes Informationssystem) es sich handelt. Jede Interaktion mit Objekten der Umwelt wird als kritisch angesehen.

Folgt man den Abgrenzungen von DAVENPORT, SHORT, HARRINGTON und HEINEN, lassen sich Prozesse auf der Grundlage des Flusses unterscheiden. Materielle und informationelle Prozesse lassen sich durch die Art der zwischen den Objekten fließenden Ströme (vgl. 7.2.4) unterscheiden. Die folgenden Zeilen enthalten Interaktionen eines materiellen Prozesses:[3]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Bauteil-Produzent	liefert Bauteile :Action	:Interaction Flow(Material)	:Bauteillager	nimmt Bauteile entgegen :Operation
	:Bauteillager	liefert Bauteile :Action	:Interaction Flow(Material)	:Fertigung	fertigt Produkt :Operation
	:Fertigung	liefert Produkt :Action	:Interaction Flow(Material)	:Auslieferungslager	Produkt einlagern :Operation

Skript 185: Ein materieller Prozess

Der zugehörige (informationsverarbeitende) Geschäftsprozess findet sich auszugsweise in folgenden Zeilen:[4]

[1] Vgl. [Striening 1988: S. 37].

[2] Vgl. [Hoyer 1988: S. 37].

[3] Die Interaktionen stammen aus dem Beispiel des Komponentenherstellers (Kapitel 9). Einige Prädikate wurden aus Raumgründen weggelassen.

[4] Die Interaktionen stammen ebenfalls aus dem Beispiel des Komponentenherstellers (Kapitel 9). Einige Prädikate wieder wurden aus Raumgründen weggelassen.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Einkauf	bestellt Bauteile :Action	:Interaction Flow(Information)	:Bauteil-Produzent	erhält Bestellung :Operation
:Bauteil-Produzent	versendet Lieferschein :Action	:Interaction Flow(Information)	:Bauteillager	vergleichen Lieferung mit Lieferschein :Operation
:Fertigung	ruft Bauteile ab :Action	:Interaction Flow(Information)	:Bauteillager	Bauteile zur Fertigung liefern :Operation
:Fertigung	versendet Produktionsmeldung :Action	:Interaction Flow(Information)	:Auslieferungs-lager	erhält Produktionsmeldung :Operation

Skript 186: Auszug aus dem zugehörigen Geschäftsprozess

In beiden Fällen handelt es sich um Interaktionen operativer und primärer Prozesse. Andere Prozesstypen (Management-, Sekundär- und Innovationsprozesse) sind am Namen der Klassen und gegebenenfalls an ihrem Stereotyp zu erkennen. Im Rahmen von Sekundärprozessen werden zum Beispiel Sachmittel gewartet oder Mitarbeiter ausgebildet:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Schulungsabteilung Stpe(Department)	plant Schulung :Action	:Interaction	:Entwickler Stpe(Position)	erhält C++-Schulung :Operation
:Techniker Stpe(Position)	erstellt Datensicherung :Action	:Interaction	:E-Mail-Server	exportiert E-Mail-Verteilerlisten :Operation

Skript 187: Beispielinteraktionen aus Sekundärprozessen

Das Thema dieser Arbeit ist nicht die Modellierung von Innovations- und Managementprozessen, da sie sich auf plan- und strukturierbare, nach Regeln ausgeführte und wiederholbare Prozesse beschränkt.[1] Das bedeutet aber nicht, dass das hier entwickelte Instrumentarium nicht auch bei diesen Prozesstypen verwendet werden kann. Bei Innovationsprozessen werden beispielsweise Verfahren (Stereotyp `Knowledge`) entwickelt oder die Organisationsstruktur (Stereotyp `Organization`) verändert:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Marketing	verändert	:Interaction	:Werbestrategie	Etat Printmedien erhöhen

[1] Siehe die Abgrenzung in der Einleitung dieser Arbeit. STRIENING setzt das Kriterium der Wiederholbarkeit prinzipiell für die Existenz eines Geschäftsprozesses voraus. Vgl. [Striening 1988: S. 60].

		Werbestrategie :Action		Stpe(Knowledge)	:Operation In(:Betrag)
	:Projekt-leitung	Überabe definieren :Action	:Interaction	:Vorgehensmodell Systementwicklung Stpe(Knowledge)	Abnahmekriterien erstellen :Operation In(:Aktivität, :Kriterien)
	:Geschäfts-führung	Neue Abteilung einrichten :Action	:Interaction Content(Produktmgt. :Abteilung)	:Softwarehaus Stpe(Organization)	Abteilung hinzufügen :Operation In(:Abteilung)

Skript 188: Beispielinteraktionen aus Innovationsprozessen

Auch Zusammenhänge im Bereich der Managementprozesse können objektorientiert modelliert werden, da auch Ziele und Probleme Objekte darstellen können, an denen im Verlauf von Prozessen Verrichtungen vorgenommen werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Geschäftsführung	legt Ziel fest :Action	:Interaction	:Erhöhung Wartungsverträge Stpe(Goal)	Monetäres Zielkriterium festlegen :Operation In(:Betrag)
	:Kunde	schließt Wartungsvertrag ab :Action	:Interaction	:Vertrieb Stpe(Org-Unit)	Wartungsvertrag abschließen :Operation In(:Betrag)
	:Vertrieb Stpe(Org-Unit)	trägt zur Zielerreichung bei :Action	:Interaction	:Erhöhung Wartungsverträge Stpe(Goal)	Zu monetärem Ziel beitragen :Operation In(:Betrag, :Mitarbeiter)

Skript 189: Beispielinteraktionen aus dem Bereich der Managementprozesse

7.3.2 Verflechtung von Basissystem und Informationssystem

Die unterschiedlichen Prozessarten stehen nicht isoliert nebeneinander, sondern werden in den geschäftsprozessorientierten Ansätzen zusammen betrachtet.[1] So sind sekundäre Prozesse durch interne Kunden-Lieferanten-Beziehungen mit den primären Prozessen verknüpft. Die Innovationsprozesse können primäre und sekundäre Prozesse auslösen.[2] Schließlich ist die Manipulation physischer Objekte in der Regel mit einer Aktivität im Bereich der Informationsverarbeitung verbunden.[3] Im Informationssystem werden Informationen ausgetauscht, während im Basissystem

[1] KRÜGER spricht hier von *integrativer Strukturierung*. Vgl. [Krüger 1994: S. 127].

[2] Vgl. [Schulte-Zurhausen 1999: S. 83].

[3] Vgl. [Davenport 1993: S. 92], [Striening 1988: S. 59].

auch Nicht-Informationen ausgetauscht werden.[1] Die Informationsflüsse laufen dabei parallel zu Materialflüssen ab.[2] Bezogen auf die Geschäftsprozesse bedeutet das, dass sie die materialflussbezogenen Prozesse steuern und koordinieren.[3] „*Jeder Geschäftsvorfall im Basissystem führt zu einer entsprechenden Transaktion im Informationssystem, die den Geschäftsvorfall mit Hilfe von Transaktionsbelegen begleitet. Transaktionsbelege dienen außerdem der Kommunikation mit der Umwelt des Informationssystems sowie der Synchronisation zwischen Basissystem und Informationssystem.*“[4]

NORDSIECK schreibt, dass *„ein fließender Fertigungsprozess gesteuert und kontrolliert wird durch einen begleitenden Prozess der Datenverarbeitung und Kontrolle“*[5]. Wenn er feststellt, dass Verwaltungsaufgaben „*einen den Betriebsprozess begleitenden Vorgang dar[stellen]“*,[6] entspricht dies in weiten Teilen dem heutigen Prozessverständnis. Die Verwaltungsaufgaben sind dabei aus dem horizontal verlaufenden Betriebsprozess vertikal herausgehoben.[7] Aufgrund dieser Verflechtung sind bei der Prozessmodellierung sowohl die materiellen wie auch die informationellen Flüsse in der Analyse zu berücksichtigen.[8]

Den in Geschäftsprozessen fließenden Informationen[9] steht häufig aber auch kein Materialfluss gegenüber. Dies hat auch schon NORDSIECK erkannt: „*Jeder auf materielle Güter gerichtete Warenprozess wird durch den entsprechenden, auf immaterielle Güter ausgerichteten Informationsfluss gespiegelt; umgekehrt existiert nicht zu jedem Informationsfluss ein entsprechender Warenprozess.*“[10] NORDSIECK relativierte damit schon 1972 die Bedeutung des Materialflusses: „*Die meisten und wichtigsten Betriebsprozesse finden ohne Materialfluss statt.*“[11]

---

[1] Vgl. [Ferstl 1998: S. 5, 28 f.] Die Differenzierung zwischen Basis- und Informationssystem geht auf GROCHLA zurück. Vgl. [Grochla 1975b: S. 12 f.].

[2] Vgl. [Schulte-Zurhausen 1999: S. 51]. Ähnlich auch in [Denert 1991: S. 111].

[3] [HeilmannM 1997: S. 94]. Vgl. auch [Schulte-Zurhausen 1999: S. 287]. PICOT erörtert in [Picot 1999: S. 354 f.] die Vorteile der engen Anbindung von Informations- an Produktionsflüsse am Beispiel des *Kanbansystems*. KRÜGER erwähnt unter dem Schlagwort CIB (*Computer Integrated Business*) die Zusammenführung technischer Prozesse und administrativer Prozesse. [Krüger 1994: S. 127].

[4] [Ferstl 1998: S. 33].

[5] [Nordsieck 1972: Sp. 29].

[6] [Nordsieck 1972: Sp. 32].

[7] Vgl. [Nordsieck 1972: S. 32].

[8] Vgl. [Corsten 1997b: S. 33].

[9] Vgl. [Frank 1995: S. 72].

[10] [Schmidt 1997: S. 11].

[11] [Nordsieck 1972: Sp. 26].

NORDSIECKs Prozessprinzip bezieht sich auf die Erstellung der betrieblichen Kernprodukte und nicht auf den Verwaltungsbereich, den er aus dem Prozessablauf ausgliedert.[1] Hätte er dieses Prinzip auch im indirekten Bereich der Verwaltung verwendet, wäre der heute verwendete Gedanke der Geschäftsprozesse mehr als nur vorgezeichnet gewesen. So hat erst GAITANIDES 1983 das Prinzip der Prozessgliederung auch jenseits des Bereichs der materiellen Leistungserstellung verwendet: Auch *„beispielsweise Beschaffungsvorgänge und zugehörige Finanzierungsprozesse sollen in **einer** Prozessstruktur“* dargestellt werden.[2]

Die Verflechtung der unterschiedlichen Prozesstypen bedeutet für die GPM, dass es nicht ausreicht, nur die Informationsflüsse zu modellieren. Geschäftsprozesse sind häufig so eng mit Materialflüssen verbunden, dass nach der persönlichen Erfahrung des Verfassers keine isolierte Betrachtung sinnvoll ist. Um die Zusammenhänge zwischen Basissystem und Informationssystem zu verdeutlichen, unterstützt deshalb die OBA++ auch die Modellierung des Materialflusses. Skript 185 und Skript 186 enthalten isolierte Teilsichten des Material- und des Informationsflusses. Das Beispiel des Komponentenherstellers (vgl. Kapitel 9) enthält eine Gesamtdarstellung, in der der parallele Verlauf von Informations- und Materialflüssen sichtbar wird. Beispielsweise wird der Materialfluss (`Lieferung`) vom Bauteil-Produzenten zum Bauteillager von einem gleichgerichteten Informationsfluss (`Lieferschein`) begleitet, was am Prädikat `In()` der Operation erkennbar ist. Wie man an der Interaktion zwischen Fertigung und Bauteillager erkennen kann, verlaufen teilweise die Flüsse im Basissystem aber auch entgegengesetzt zu denen im Informationssystem. Die Fertigung veranlasst durch einen Informationsfluss (`Lagerabruf`) einen Materialfluss (`Bauteile`) in entgegengesetzter Richtung.[3] Dass die Masse der Informationsflüsse ohne begleitenden Materialfluss stattfinden, wird ebenfalls sichtbar. `Einkauf`, `Verkauf` und `Produktionssteuerung` tauschen nur Informationen aus. Am Beispiel des Verkaufs und der Produktionssteuerung wird erkennbar, wie die Abläufe im Informationssystem die im Basissystem steuern. Der `Verkauf` reicht den Auftrag an die `Produktionssteuerung` weiter, welche die Prüfung der Materialverfügbarkeit durch das `Bauteillager` und die Produktion durch die `Fertigung` veranlasst.

7.3.3 Prozessabgrenzung

Es lassen sich verschiedene Arten von Prozessen unterscheiden, zwischen denen Beziehungen existieren. Unklar ist dagegen noch, wie man einen Geschäftsprozess gegen einen anderen Geschäftsprozess abgrenzt. Nach ÖSTERLE stellt ein Prozess eine Folge von Aufgaben zwischen

---

[1] Vgl. [Nordsieck 1972: Sp. 32], [Nordsieck 1955: S. 88, 101].

[2] [Gaitanides 1983: S. 65]. (Hervorhebung im Original.)

[3] Hier wurde – nur zur Veranschaulichung – das Prädikat `Content()` verwendet.

der Entstehung und der Beseitigung eines Geschäftsobjekts dar.[1] Nach SCHULTE-ZURHAUSEN sind Prozesse Folgen von logisch beziehungsweise funktional zusammenhängenden Aktivitäten zur Erstellung eines Ergebnisses, einer Leistung oder zur Veränderung eines Arbeitsobjekts.[2] Auch nach BECKER und VOSSEN wird eine inhaltlich abgeschlossene und sachlogische Abfolge von Funktionen, die zur Bearbeitung eines betriebswirtschaftlichen Objekts notwendig ist, als Prozess betrachtet.[3] Die zugehörigen Aktivitäten eines Geschäftsprozesses sorgen für die

- Erstellung und Vermarktung eines Produkts oder einer spezifischen Dienstleistung,
- Planung, Steuerung und Verwaltung der Ressourcen sowie für die
- Beeinflussung der Umwelt (Kunden, Lieferanten, Öffentlichkeit).[4]

SCHULTE-ZURHAUSEN nennt als Beispiele für typische Geschäftsprozesse eines Industrieunternehmens:[5]

- Systematische Untersuchung von Teilmärkten
- Auftragsbearbeitung vom Angebot bis zur Auslieferung
- Beschaffung von Roh-, Hilfs- und Betriebsstoffen
- Produktentwicklung von der Produktidee bis zum Produktionsbeginn
- Produktion von der Vorfertigung bis zur Endmontage
- Physischer Materialfluss vom Lieferanten bis zum Kunden
- Geschäftsfeldplanung einschließlich Budgetierung
- Organisationsgestaltung von der Vorstudie bis zur Einführung

Schon in Abschnitt 7.3.1 und 7.3.2 wurde festgestellt, dass Informationsprozesse und Prozesse im Basissystem nicht isoliert betrachtet werden, sondern zusammen einen Geschäftsprozess bilden. Ein weiteres (eher allgemeines) Kriterium für die Abgrenzung von Geschäftsprozessen ist ihre Selbständigkeit. Das bedeutet, dass man sie vergleichsweise isoliert von anderen Geschäfts-

[1] Vgl. [Österle 1995: S. 87].

[2] Vgl. [Schulte-Zurhausen 1999: S. 54, 107].

[3] Vgl. [Becker 1996: S. 11].

[4] [Schulte-Zurhausen 1999: S. 54, 83].

[5] [Schulte-Zurhausen 1999: S. 54]. Vgl. auch [Wassermann 2001: S. 43 ff.] und die umfangreiche Darstellung unterschiedlicher Geschäftsprozesse in [Schmelzer 2002: S. 117 ff.].

prozessen betrachten und gestalten kann. Dadurch soll der Koordinationsaufwand zwischen Geschäftsprozessen reduziert werden.[1]

Aber abgesehen von diesen sehr allgemeinen Richtlinien kann man insgesamt feststellen, dass kein generelles Kriterium für den Beginn und das Ende von Geschäftsprozessen bestimmt werden kann.[2] Das bestätigt die Position von GAITANIDES, nach der Anfang und Ende eines Prozesses eher einer subjektiven Problemsicht entspringen. Ein Prozess gilt demnach als in sich abgeschlossen, wenn der Organisator ihn isoliert betrachten will.[3]

Inhaltlich existieren keine sinnvollen Kriterien für die Länge von Geschäftsprozessen. Dies ist jedoch unproblematisch, da mit der OBA++ Prozesse in beliebiger Granularität und beliebigem Umfang modelliert werden können. Dabei werden nur elementare Prozesse mit einer einfachen operativen Prozessstruktur oder High-Level- und Blackbox-Ansichten (vgl. Abschnitt 6.12.2) in einem einzigen Skript abgebildet. Bei komplexeren Abläufen verwendet man Subskripte (vgl. Abschnitt 6.12.2), um die Realisierung oder Verfeinerung einzelner Operationen zu dokumentieren. Die in High-Level-Ansichten definierten Schnittstellen von Prozessen werden in Subskripten als Blackbox- oder Whitebox-Ansichten verfeinert. Sehr lange Abläufe werden zerlegt in Abfolgen einzelner Skripte (vgl. Abschnitt 6.12.3).

7.3.4 Ereignisse

Da Organisationen als offene Systeme verstanden werden, müssen sie sich den aus ihrer Offenheit ergebenden Veränderungen anpassen. Sie benötigen einen Mechanismus, um Veränderungen festzustellen, diese zu verarbeiten bzw. entsprechend zu reagieren.[4] Dieser Mechanismus wird als Ereignis bezeichnet. Ein *Ereignis* ist das Eintreten eines definierten Zustandes[5], einer neuen Situation[6], das Auftreten eines Objektes oder die Veränderung einer Attributausprägung.[7] „*Ereignisse beschreiben eine Zustandsveränderung und kennzeichnen z. B. den Eintritt des Ergebnisses eines Vorgangs, das dann den folgenden Vorgang auslöst.*“[8]

---

[1] Vgl. [Schulte-Zurhausen 1999: S. 84].

[2] Vgl. [Staud 1999: S. 10 f.], [Staud 2001: S. 6, 22 f.], [Corsten 1997b: S. 25], [Rauschecker 2000: S. 153], [Schulte-Zurhausen 1999: S. 84], [Engelmann 1995: S. 46].

[3] Vgl. [Gaitanides 1983: S. 65]. Eine andere Möglichkeit ist, Beginn und Ende von Geschäftsprozessen durch den Kontakt mit externen Kunden abzugrenzen. Vgl. [Schmelzer 2002: S. 36 ff.].

[4] Vgl. [Kappler 1991: S. 78].

[5] [REFA 1991/6: S. 12].

[6] Vgl. [Schulte-Zurhausen 1999: S. 50].

[7] Vgl. [Scheer 1995: S. 49].

[8] [Scheer 1998b: S. 18].

Prozesse werden durch Ereignisse begonnen.[1] Sie legen damit die Bedingungen für den Anstoß eines Prozesses fest.[2] Solche Ereignisse werden in der GPM auch als *Geschäftsereignisse* bezeichnet.

Ereignisse entstehen in der OBA++ als Folge der Aktion des Initiators. Jede Interaktion stellt implizit ein Ereignis für den Participant dar. Auf dieses Ereignis folgt die Ausführung der Operation durch den Participant. Auf die unterschiedlichen Ereignistypen und ihre Verwendung wurde in Abschnitt 6.8 eingegangen. Der hier verwendete Ereignisbegriff unterscheidet sich wie folgt vom eher umgangssprachlichen Gebrauch in der GPM:

Ereignisse finden nicht nur außerhalb, sondern auch innerhalb der Organisation statt. Extern ausgelöste Geschäftsereignisse sind daran zu erkennen, dass der Initiator ein Umweltobjekt (Prädikat `System(external)`) ist. Während beispielsweise die Ereignisse in ARIS „*irgendwo*" eintreten, ist in der OBA++ immer ein Objekt die Ursache für das Eintreten des Ereignisses. Da Ereignisse keine zeitliche Ausdehnung besitzen, können nicht mehrere Ereignisse Auslöser für einen Prozess sein. Wird ein Prozess durch mehrere zusammengesetzte Ereignisse (wie in ARIS) gestartet, werden in der OBA++ die Einzelereignisse durch separate Interaktionen abgebildet und in einen zeitlichen Bezug gesetzt (vgl. Abschnitt 6.11.6). In der OBA++ hat man zusätzlich die Möglichkeit, Ereignisse verfallen zu lassen (vgl. Abschnitt 6.2.7). Das bedeutet, dass der Participant innerhalb eines bestimmten Zeitraums reagieren muss.

In der objektorientierten Ereignisgesteuerten Prozesskette (oEPK)[3] werden auch die Vor- und Nachbedingungen von Aktivitäten durch Ereignisse abgebildet.[4] Die Nachbedingungen der Operationen sind gleichzeitig die auslösenden Ereignisse der folgenden Aktivität. Dies entspricht den Aktionszuständen in den UML-Aktionsdiagrammen, die ebenfalls direkt die Folgeaktion auslösen.[5] Eine Interaktion ist nicht notwendig. Damit sind Prozesse aber keine Abfolge von Interaktionen zwischen Objekten. Es bleibt offen, wie gegebenenfalls ein anderes Objekt davon erfährt, dass seine Operation ausgeführt werden soll. Aus diesem Grund wird hier zwischen Ereignissen und Vor- bzw. Nachbedingungen unterschieden. Die Nachbedingung einer Operation stellt nicht den Auslöser für eine andere Operation dar. Hierfür bedarf es noch einer Interaktion.

---

[1] Vgl. [Bullinger 1996: S. 96], [HeilmannM 1997: S. 90], [Schulte-Zurhausen 1999: S. 45, 50, 59], [Bertram 1996: S. 86], [Völkner 1998: S. 12], [Fischermanns 1997: S. 23], [ZimmermannV 1999: S. 8], [Corsten 1997b: S. 16], [Kruse 1996: S. 21], [Ferstl 1993: S. 590], [Elgass 1993: S. 43], [Kruse 1992: S. 4], [Kruse 1996: S. 123 ff.], [Fischermanns 1997: S. 23].

[2] Vgl. [Schulte-Zurhausen 1999: S. 86].

[3] Vgl. [ZimmermannV 1999], [Scheer 1997], [Nüttgens 1998].

[4] Vgl. [ZimmermannV 1999: S. 106 ff.].

[5] Vgl. [OMG 2001: S. 3-163].

7.3.5 Ziele

In Anlehnung an ÖSTERLE lassen sich im Prozessmanagement[1] drei Betrachtungsebenen identifizieren:

- Auf der Ebene der *Strategieebene*[2] werden Entscheidungen über Ziele, Märkte, Geschäftsfelder, kritische Erfolgsfaktoren und die Unternehmensstrategie oder -politik getroffen.
- Die *Prozessebene* betrachtet betriebliche Abläufe, durch die organisatorische Einheiten mittels ihrer Aufgaben Leistungen erbringen.
- Die *Informationssystemebene* spezifiziert Informationssysteme, die die automatisierten Teile des Geschäftsprozesses unterstützen.

Ziele werden auf der strategischen Ebene definiert. Sie lassen sich hierarchisch beziehungsweise horizontal in Ober- und Unterziele und vertikal in Abfolgen von Zwischenzielen strukturieren.[3] Durch die hierarchische Strukturierung entstehen Mittel-Zweck-Beziehungen.[4] Das untergeordnete Ziel stellt das Mittel zur Erreichung des übergeordneten Ziels dar. Durch die vertikale Strukturierung werden Zwischenschritte zur Erreichung eines Endziels abgegrenzt. Auf diese Weise entsteht sowohl eine horizontale wie auch eine vertikale *Zielstruktur*. Die Verbindung zwischen der strategischen Ebene und der Prozessebene wird dadurch hergestellt, dass die Prozesse ausgeführt werden, um diese Ziele zu erreichen.[5]

NORDSIECK verbindet die Begriffe Aufgabe, Ziel und Leistung, indem er das Ziel als Inhalt der Aufgabe bezeichnet, welches durch schrittweise Verwirklichung erreicht wird und das eine Leistung erfordert. In der Regel ist noch eine Ausgangslage gegeben.[6]

Bei FERSTL und SINZ werden die Ziele von Aufgaben durch deren Nachbedingungen ausgedrückt.[7] *„Die Durchführung einer Aufgabe manipuliert das Aufgabenobjekt und überführt es von einem Vorzustand in einen Nachzustand. Das Sachziel der Aufgabe gibt die gewünschten*

---

[1] ÖSTERLE verwendet den Begriff *Business Engineering*. Inhaltlich entspricht er dem häufiger verwendeten Begriff *Prozessmanagement*. Vgl. [Österle 1995: S. 16].

[2] Auch als *strategische Geschäftsprozessanalyse und Sollkonzeption* bezeichnet.

[3] Vgl. [Schulte-Zurhausen 1999: S. 355].

[4] Vgl. [Ferstl 1998].

[5] Vgl. [Bullinger1996: S. 96], [Kueng 1995b], [Kueng 1996]. Zur Differenzierung der Begriffe Aufgabe und Ziel vgl. [Schulte-Zurhausen 1999: S. 70 ff.], [Bleicher 1991: S. 35].

[6] [Nordsieck 1972: Sp. 16].

[7] Vgl. [Ferstl 1994c: S. 5], [Ferstl 1993b: S. 145, 150 f.], [Ferstl 1998: S. 189], [Ferstl 1991: S. 484 ff.], [Rüffer 1999].

Nachzustände an und beschreibt damit, was die Aufgabe tun soll."[1] Das Sachziel gibt an, welche Endsituation erreicht werden soll, während sich das Formalziel auf die Form oder die Art und Weise der Zielerreichung bezieht.[2] Da die Sachziele zur Bestimmung einzelner Aktivitäten und der gesamten Organisation dienen,[3] sind sie von primärem Interesse für die GPM.

Die Modellierung der Strategieebene war nicht das Thema dieser Arbeit.[4] Es ist nicht die Aufgabe der OBA++, mit Objekten Zielstrukturen zu modellieren. Dafür kann der von ERIKSSON und PENKER entwickelte Ansatz verwendet werden, in dem Ziele als Objekte modelliert werden und die Zielstrukturen als Beziehungen zwischen den Zielobjekten.[5] Die Verbindung zwischen Strategie- und Prozessebene erfolgt nach dem Verfahren von KUENG, KAWALEK, BICHLER und SCHREFL.[6] Danach muss für jedes Ziel im Zielsystem in den Geschäftsprozessen eine entsprechende Aufgabe bzw. Operation vorhanden sein. Der von NORDSIECK definierte Zusammenhang zwischen Aufgabe, Ziel und Leistung stellt sich in der OBA++ in der Form dar, dass Aufgabe und Leistung durch Operationen von Klassen abgebildet werden. Die Ziele werden jedoch – im Gegensatz zu FERSTL und SINZ – nicht durch die Nachbedingungen der Operationen dokumentiert, sondern durch das Prädikat `Goal()` der Operation festgehalten. Dass sich die horizontale und vertikale Zielstruktur in den Skripten wiederfinden lässt, soll an folgendem Skript demonstriert werden:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	:Sachbearbeiter Stpe(Position)	liefere Angebot :Operation Goal(Kunde nimmt Angebot an) In(:Artikeldaten) Out(:Angebot)

Skript 190: Ausgangspunkt der Zielstrukturierung

Diese Interaktion wird durch das folgende Unterskript operativ verfeinert. Ober- und Unterziele bzw. Zwischenziele werden in der OBA++ durch die Prädikate `Goal()` der einzelnen Operationen

[1] [Ferstl 1998: S. 184].

[2] Vgl. [Wild 1966: S. 90]. *„Die Aufgabenziele werden implizit durch Benennung der Aufgabenverrichtung spezifiziert. ... Eine Transformationsaufgabe realisiert ausschließlich Sachziele (AS), eine Entscheidungsaufgabe nimmt zusätzlich auf Formalziele (AX) Bezug."* [Ferstl 1998: S. 90]. Transformations- und Entscheidungsaufgaben werden in Abschnitt 7.4.6 dargestellt.

[3] Vgl. [Wild 1966: S. 90].

[4] Vgl. Abschnitt 1.7.

[5] Vgl. [Eriksson 2000: S. 78 f., 283 ff.]

[6] Vgl. [Kueng 1995b], [Kueng 1996], [Kueng 1996b].

gebildet. Die Aufgabe `liefere Angebot` mit dem Oberziel `Kunde nimmt Angebot an` wird durch ein Unterskript verfeinert. Das Oberziel wird dabei in die Unterziele `Angebot ist eingetroffen`, `Kundendaten sind ermittelt`, `Artikeldaten sind bekannt`, `Artikelpreis ist berechnet`, und `Angebotsposition ist erstellt`, `Angebot ist versendet`, `Angebot ist angenommen`. zerlegt. Gleichzeitig wird durch die Sequenzausdrücke eine Abfolge bestimmt und damit Zwischenziele festgelegt:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	:Sachbearbeiter Stpe(Position)	erhält Anfrage :Operation Goal(Angebot ist eingetroffen) In(:Artikeldaten)
2.	:Sachbearbeiter Stpe(Position)	ermitteln :Action	:Interaction	:Versand Stpe(Org-Unit)	liefere Kundendaten :Operation Goal(Kundendaten sind ermittelt) In(:Kundenname) Out(:Kundendaten)
3.	:Sachbearbeiter Stpe(Position)	suchen :Action	:Interaction	:Artikelstamm	finden :Operation Goal(Artikeldaten sind bekannt)
4.	:Sachbearbeiter Stpe(Position)	berechnen :Action	:Interaction	:Angebots- position	berechne Abschlag :Operation Goal(Artikelpreis ist berechnet)
5.	:Sachbearbeiter Stpe(Position)	erstellen :Action	:Interaction	:Angebot	einfügen Angebotsposition :Operation Goal(Angebotsposition ist erstellt)
6.	:Sachbearbeiter Stpe(Position)	senden :Action	:Interaction	:Angebot	wird versendet :Operation Goal(Angebot ist versendet)
7.	:Angebot	eintreffen :Action	:Interaction	/Kunde :Person Stpe(Individual) System(external)	Angebot akzeptieren :Operation Goal(Angebot ist angenommen) In(:Angebot)

Skript 191: Beispiel zur Zielstrukturierung

7.3.6 Operative Prozessstruktur

Innerhalb einer Betrachtungsebene sind Prozesse ablauforganisatorische Zusammenfassungen von Aktivitäten bzw. Aufgaben.[1] Sie bestehen aus einer Menge von Aktivitäten,[2] die in einem logischen Zusammenhang[3] beziehungsweise in Reihenfolgebeziehungen stehen.[1] Eine solche Abfolge

[1] „*Eine ablauforganisatorische Zusammenfassung von Aufgaben bildet einen Prozess.*“ [Eversheim 1995: S. 15].

[2] Vgl. [Hammer 1993: S. 27].

[3] Vgl. [Becker 1996: S. 18], [Becker 2000: S. 4], [Davenport 1993: S. 84], [HeilmannM 1997: S. 89], [Schulte-Zurhausen 1999: S. 49], [Scholz 1994b: S. 42], [Völkner 1998: S. 11], [Striening 1988: S. 57],

funktional zusammenhängender Aktivitäten wird als *Prozesskette* bezeichnet.[2] Durch diese *operative Prozessstruktur* wird die logische Folge von Aktivitäten abgebildet. Sie kann Verzweigungen und Vereinigungen sowie Aufspaltungen und Synchronisationen enthalten.[3]

Zur Darstellung der operativen Prozessstruktur werden meistens Darstellungen verwendet, die Flussdiagrammen ähneln: „*Die logische Abfolge der einzelnen Aktivitäten wird zweckmäßigerweise mit Hilfe von Folgeplänen dokumentiert. Hierbei wird die Abfolge der im Gesamt- oder in einem Teilprozess durchzuführenden Schritte sowie deren Abhängigkeiten und Beziehungen untereinander beschrieben.*“[4]

Die OBA++ verwendet keine Flussdiagramme, sondern Sequenzausdrücke. Durch Sequenzausdrücke wird die operative Prozessstruktur festgelegt, während die Vor- und Nachbedingungen als Anwendbarkeitsbedingungen festlegen, unter welchen Bedingungen prinzipiell Operationen ausgeführt werden können.[5] Dieser Unterschied soll an folgendem Beispiel untersucht werden, welches eine Erweiterung von Skript 191 ist.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction	:Sachbearbeiter Stpe(Position)	erhält Anfrage :Operation Goal(Angebot ist eingetroffen) In(:Artikeldaten) Pre(True) Post(Kundenname vorhanden)
2.	:Sachbearbeiter Stpe(Position)	ermitteln :Action	:Interaction	:Versand Stpe(Org-Unit)	liefere Kundendaten :Operation Goal(Kundendaten sind ermittelt) Pre(Kundenname vorhanden)

[Kruse 1996: S. 24], [Krickl 1994b: S. 19], [Elgass 1993: S. 43], [Grob 1995], [Johansson 1994: S. 57], [Staud 1999: S. 6], [Fischermanns 1997: S. 42], [Haist 1989: S. 93], [Krüger 1994: S. 124].

[1] Vgl. [Scheer 1998b: S. 11], [Fischermanns 1997: S. 24], [Bertram 1996: S. 86], [Österle 1995: S. 19], [Rummler 1991: S. 45], [Harrington 1991: S. 9], [Corsten 1997b: S. 16]. Dies wird auch als *temporale Zuordnung* bezeichnet. [Krüger 1994: S. 15].

[2] Vgl. [Schulte-Zurhausen 1999: S. 53 f.], [Eversheim 1995: S. 15].

[3] „*Dabei können sequentielle, parallele, alternative und zusammenführende Wege mit logischen Verknüpfungen bestehen.*“ [Scheer 1998b: S. 18]. Vgl auch [Popp 1994: S. 50], [Haist 1989: S. 93], [Scheer 1998b: S. 18], [Harrington 1991: S. 86 ff.], [Fischermanns 1997: S. 26, 44 ff.].

[4] Vgl. [Schulte-Zurhausen 1999: S. 100]. Vgl. auch die Darstellungen der unterschiedlichen Werkzeuge zur GPM in [Bullinger 2001b].

[5] Wie auch im Ansatz von MORABITO, SACK und BHATE werden hier Operationen durch Vor- und Nachbedingungen spezifiziert. Vgl. [Morabito 1999: S. 180 ff.]. Dies ist ähnlich auch in SOM zu finden. Vgl. [Ferstl 1998: S. 184]. Dort entsteht eine Sequenz, wenn die Vorbedingung einer Operation der Nachbedingung einer anderen Operation entspricht. Dieses Prinzip wurde hier nicht übernommen.

					Post(Kundendaten vorhanden)
3.	:Sachbearbeiter Stpe(Position)	suchen :Action	:Interaction	:Artikelstamm	finden :Operation Goal(Artikeldaten sind bekannt) Pre(Artikelnummer vorhanden) Post(Artikeldaten vorhanden)
4.	:Sachbearbeiter Stpe(Position)	berechnen :Action	:Interaction	:Angebotsposition	berechne Abschlag :Operation Goal(Artikelpreis ist berechnet) Pre(Artikeldaten vorhanden; Kundendaten vorhanden) Post(Angebotsposition berechnet)
5.	:Sachbearbeiter Stpe(Position)	erstellen :Action	:Interaction	:Angebot	einfügen Angebotsposition :Operation Goal(Angebotsposition ist erstellt) Pre(Angebotsposition berechnet) Post(Angebotsposition eingefügt)
6.	:Sachbearbeiter Stpe(Position)	senden :Action	:Interaction	:Angebot	wird versendet :Operation Goal(Angebot ist versendet) Pre(Angebotsposition eingefügt) Post(Angebot versendet)
7.	:Angebot	eintreffen :Action	:Interaction	/Kunde :Person Stpe(Individual) System(external)	annehmen Angebot :Operation Goal(Angebot ist angenommen) In(:Angebot) Pre(True) Post(Angebot erhalten)

Skript 192: Operative Prozessstruktur mit Vor- und Nachbedingungen

Die durch Sequenzausdrücke ausgedrückte operative Prozessstruktur ergibt sich in diesem Beispiel aus der definierten Abfolge der Zwischenziele. Würde man die Zwischenziele anders strukturieren, ergäbe sich auch eine andere Prozessstruktur.

Durch Sequenzausdrücke kann auch erkennbar gemacht werden, wenn Aktivitäten nicht an eine bestimmte Sequenz gebunden sind oder die Reihenfolge unerheblich ist, wie dies SCHOLZ fordert.[1] Wie in Abschnitt 6.11.7 demonstriert wurde, lassen sich durch Sequenzausdrücke auch Kontrollstrukturen ausdrücken, die insbesondere bei der GPM nützlich sind. Dies ist bei der Abbildung unstrukturierter Abläufe ein Vorteil gegenüber Flussdiagrammen.[2] Darüber hinaus kann erkennbar gemacht werden, wenn Aktivitäten parallel verlaufende Folgeaktivitäten besitzen.[3] Auch dies ist ausführlich im letzten Kapitel dargestellt worden.

Im letzten Skript ist an den jeweiligen Vor- und Nachbedingungen zu erkennen, dass der Abschlag erst berechnet werden kann, wenn die Kunden- und die Artikeldaten bekannt sind.

[1] Vgl. [Scholz 1994b: S. 42].

[2] Viele Prozesse enthalten Aktivitäten, deren Reihenfolge nicht festgelegt ist, wie z. B. im Bereich der Konstruktion. In diesem Fall kann der Geschäftsprozess nur über Ziele und Meilensteine definiert werden. Vgl. [Siebert 2001], [Jablonski 2001].

[3] Vgl. [Scholz 1994b: S. 42].

Dagegen spielt es keine Rolle, ob zuerst die Kundendaten oder die Artikeldaten ermittelt werden. In der letzten Zeile des Skripts wird schließlich auch deutlich, dass Vor- und Nachbedingungen nur beschränkt zur Spezifikation von Reihenfolgebeziehungen geeignet sind. Die einzige Bedingung, die erfüllt sein muss, damit ein Kunde ein Angebot erhält, ist wohl die, dass es abgeschickt wurde. Dies ist aber schon durch die Interaktion selbst ausgedrückt. Daher kann die Vorbedingung nur logisch wahr (`True`) sein. Welche Aktivitäten vorher passiert sein müssen, kann der Kunde nicht wissen, und es spielt auch für seine Leistung, das Angebot anzunehmen, keine Rolle. Daraus ergibt sich, dass die Vor- und Nachbedingungen und die Sequenzausdrücke zwar häufig ähnliche Informationen beinhalten, einander aber nicht vollständig ersetzen können. Wenn sich dagegen in Prozessen keine operative Prozessstruktur feststellen lässt, sondern für jede Operation immer nur bestimmte Bedingungen vorher erfüllt sein müssen, verlagert sich der Schwerpunkt von der Festlegung einer operativen Prozessstruktur weg und hin zur Definition von Vor- und Nachbedingungen. Dies ist gerade bei schwach strukturierten informationsverarbeitenden Prozessen häufig der Fall.[1]

7.3.7 Variantenbildung

Mit zunehmender Detaillierung von Prozessen zeigen sich unterschiedliche Varianten der Ausführung.[2] HAMMER und CHAMPY bezeichnen mit *Triage* die Definition solcher Varianten für unterschiedliche Anforderungen an einen Prozess.[3] Dadurch erfolgt eine Differenzierung von Standard- und Sonderfällen mit dem Ziel, komplexe Prozesse zu vereinfachen.[4] Man dokumentiert einen einfachen Standardverlauf und separat die abweichenden Varianten. Dadurch soll auch die höhere Komplexität der Varianten deutlich werden.[5] Der Punkt, bis zu dem unterschiedliche

[1] Diese Aussage basiert auf persönlichen Erfahrungen des Verfassers.

[2] Vgl. [Kieser 1996: S. 237], SPECK und SCHNETGÖTGE in [Becker 2000: S. 173], [HeilmannM 1997: S. 103 ff.], [Scholz 1994b: S. 49], [Bertram 1996: S. 86], [Schulte-Zurhausen 1999: S. 89 f.], [Fischermanns 1997: S. 144 ff.].

[3] Vgl. [Hammer 1994: S. 77 f.].

[4] So sind bei der Differenzierung von Geschäftsprozessen nach SEMPF einfache von komplexen Prozessen zu unterscheiden, flexibel zu haltende Prozesse von Standardabläufen und zeitkritische Prozesse von solchen mit hohen Qualitätsanforderungen. Vgl. [Sempf 1993: S. 371].

[5] Vgl. [Schulte-Zurhausen 1999: S. 90]. Dadurch soll auch das sog. „*Dilemma der Prozessdarstellung*“ vermieden werden, nach dem einerseits Abläufe möglichst einfach und übersichtlich darzustellen sind, andererseits die tatsächlichen Bearbeitungsvorgänge so exakt wie notwendig wiederzugeben sind. Vgl. [Scholz 1994b: S. 49].

Prozesse einen gemeinsamen Ablauf besitzen, wird als *Konfigurations-* oder *Varianten-bestimmungspunkt* bezeichnet.[1]

Die Differenzierung von Standard- und Sonderfällen erfolgt in der OBA++ mit Hilfe von Überwachungsbedingungen. Die Operationen `prüft Kreditwürdigkeit` und `genehmigt Kredit` sind die Schnittstellen von Prozessen, die als Sonderfälle in Abhängigkeit von der Kreditsumme ausgeführt werden. Die durch die Interaktion `Kreditsumme` ausgelöste Operation `ermitteln der Summe` ist der Variantenbestimmungspunkt. Die im Rahmen der Sonderfälle stattfindenden Interaktionen können entweder als geschachtelte Interaktionen mit prozeduralem Kontrollfluss oder als Subskripte dokumentiert werden. Ob die Sonderfälle parallel oder sequentiell ausgeführt werden, wird durch die Sequenzausdrücke dokumentiert. In diesem Fall ist festgelegt worden, dass die Bearbeitung der Sonderfälle parallel verlaufende Prozesse darstellen. Der Kreditvertrag wird entweder direkt nach der Prüfung der Kreditsumme erstellt oder wenn – nach der Prüfung der Kreditsumme – die Auskunft und/oder die Genehmigung vorliegen.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Antragsteller Stpe(Individual)	Kreditantrag stellen :Action	:Interaction	:SB Stpe(Position)	erhält Kreditantrag :Operation
2.	:SB Stpe(Position)	formal prüfen :Action	:Interaction	:Kreditantrag	notwendige Daten vorhanden :Operation
3.	:SB Stpe(Position)	Kreditsumme prüfen :Action	Kreditsumme :Interaction	:Kreditantrag	ermitteln der Summe :Operation
3. / A5.	:SB Stpe(Position)	Auskunft einholen :Action	Auskunft :Interaction Guard(10.000 GE < Kreditsumme)	:Schufa System(external) Stpe(Organization)	prüft Kreditwürdigkeit :Operation
3. / B5.	:SB Stpe(Position)	genehmigen lassen :Action	Genehmigung :Interaction Guard(Kreditsumme > 40.000 GE)	:Gruppenleiter Stpe(Position)	genehmigt Kredit :Operation
3.^ (A5.|B5.) / 7.	:SB Stpe(Position)	erstellen :Action	Erstellung :Interaction	:Kreditvertrag	wird erzeugt :Operation

Skript 193: Modellierung einer Triage

Die Modellierung von Varianten ist mit der OBA++ aber nicht auf Überwachungsbedingungen beschränkt. Da die Integrität des Modells durch die Abbildung auf das Konzeptuelle Schema sichergestellt wird, werden einfach die unterschiedlichen Abläufe, die zu einem Geschäftsprozess gehören, in Form von Skripten abgebildet. Dies sind in der Regel:

- ein einfacher Standardverlauf

[1] Vgl. [Schulte-Zurhausen 1999: S. 91].

- eine größere Anzahl von Sonderfällen
- Abläufe, in denen die Bearbeitung von Fehlersituationen dokumentiert wird (s. u.)

7.3.8 Ausnahme- und Fehlersituationen

In Prozessen können immer Ausnahmesituationen und Fehler auftreten, die entsprechend zu behandeln sind.[1] Bestimmte Prozesse bestehen sogar fast nur aus Festlegungen, wie auf Ausnahmen zu reagieren ist.[2] Zum Modell eines Geschäftsprozesses gehört folglich auch die Behandlung von Ausnahme- bzw. Fehlersituationen.[3] Eine solche explizite Differenzierung zwischen Normal- und Ausnahmeverhalten führt zu einem „schlanken" Modell des Normalverlaufs von Geschäftsprozessen,[4] da die Bearbeitungsschritte der Ausnahmesituationen separat dokumentiert werden. GALLER unterscheidet bei der Modellierung von Ausnahmen folgende Situationen (siehe Skript 195): [5]

Zwischen den Aktivitäten können unterschiedliche Arten der *Ausnahmebeziehung* existieren:

- Der Vorgang wird (vor oder nach der Bearbeitung) an einen beliebigen vor- oder nachgelagerten Aktionsträger weitergeleitet. (= Situation c)
- Der Vorgang wird (vor oder nach der Bearbeitung) an den Aktionsträger der vorherigen Aktivitäten zurückübergeben. (Sonderfall von Situation c)
- Der Vorgang wird (ohne Bearbeitung) an den Aktionsträger der nächsten Aktivität weitergeleitet. (= Situation e)

Durch die *Ausnahmezuordnung* werden zusätzlich andere Organisationseinheiten in die Bearbeitung einbezogen:

- Zur Bearbeitung des Vorgangs wird eine weitere Organisationseinheit einbezogen. (= Situation b)
- Der Aktionsträger, der die Ausnahme erzeugt hat, kann eine Rückfrage an eine andere Organisationseinheit stellen. (wird wie Situation b modelliert)

[1] Vgl. [Mayer S. 38], [Harrington 1991: S. 149].

[2] [Jablonski 2001].

[3] Vgl. [Ferstl 1996: S. 60], [Krüger 1994: S. 132], [Bertram 1996: S. 86], [Klepzig 1997: S. 85], [Eversheim 1995: S. 42].

[4] Vgl. [Ferstl 1996: S. 61].

[5] Vgl. [Galler 1997: S. 78 ff.].

- Der Vorgang wird vollständig an eine andere Organisationseinheit übergeben. (= Situation c)

Zur Bearbeitung von Ausnahmen sind bestimmte *Befugnisse* notwendig:

- Die Vorgangsbearbeitung muss unterbrochen werden können. (= Situation d)
- Die Vorgangsbearbeitung muss zurückgesetzt werden können. Dadurch werden die Effekte der Bearbeitung kompensiert. (= Situation a)
- Ist der Vorgang schon an eine andere Organisationseinheit weitergeleitet worden, muss er ggf. zurückgeholt werden.[1]

Im Gegensatz zur OBA++ beinhalten die meisten Methoden im Bereich der GPM keine Konzepte für Ausnahmesituationen. In den folgenden beiden Beispielen sind einige der von GALLER genannten Situationen exemplarisch dargestellt. Im nächsten Skript ist der Standardverlauf eines Geschäftsprozesses dokumentiert. Die für die Operationen definierten Ausnahmen finden sich als Prädikat `Err()`. Die Beschreibung der Ausnahme geschieht durch das Prädikat `when()` (vgl. Abschnitt 6.8.2). Auf die verschiedenen Verfahren, eingetretene Fehler zu korrigieren, wird hier nicht eingegangen, sie finden sich in Abschnitt 6.6.10. Die Ausnahmebehandlung geschieht durch separate Interaktionen, die durch die Interaktionsart `Exception` gekennzeichnet werden. Sie sind im darauf folgenden Skript dokumentiert worden. Dort findet sich auch ein Beispiel für die Modellierung von *Befugnisse*n. Durch den Ausdruck

```
unterbrochen :Operation
Vis(qualified)
Exp( :Einkauf, :Lager, Produktion, :Vertrieb)
Post( State = unterbrochen)
```

wurde festgelegt, dass nur der Einkauf, dass Lager, die Produktion oder der Vertrieb die Befugnis besitzen, einen Auftrag zu unterbrechen.[2]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Kunde	bestellen :Action	:Interaction	:Aufan.	annehmen Bestellung :Operation In(:Auftrag) Pre(Auftrag.State = initial) Post(Auftrag.State = geprüft)
1.1.	:Aufan.	übernimmt	Situation	:Kundendatei	liefere Kundendaten :Operation

[1] Die Behandlung dieser Ausnahmesituation würde – analog zur Situation d – durch einen Ablauf modelliert werden, innerhalb dessen der Initiator zuerst durch einen Operationsaufruf den Vorgang zurückholt und dann an eine andere Organisationseinheit vergibt.

[2] Aufan = Auftragsannahme.

		Kundendaten :Action	a und b :Interaction		Err(schlechter Kunde) Err(Kunde nicht vorhanden)
1.2.	:Aufan.	prüfen :Action	:Interaction	:Auftrag	prüfen :Operation Pre(State = initial) Post(State = geprüft)
1.3.	:Aufan.	erstellen :Action	:Interaction	:Auftrag	anlegen :Operation Post(State = aktiv)
2.	:Aufan.	übergeben :Action	Situation d :Interaction	:Lager	kommissionieren :Operation In(:Auftrag) Pre(Auftrag.State = geprüft) Post(Auftrag.State = kommissioniert) Err(Ware fehlt: Ware nicht auf Lager -> Auftrag.State = nicht kommissioniert)
2.1.	:Lager	komissionieren :Action	:Interaction	:Auftrag	kommissioniert :Operation Pre(State = geprüft) Post(State = kommissioniert)
3.	:Lager	anfertigen lassen :Action	:Interaction	:Produktion	anfertigen :Operation In(:Auftrag) Pre(Auftrag.State = kommissioniert) Post(Auftrag.State = gefertigt)
3.1.	:Produk-tion	fertigen :Action	Situation c und d :Interaction	:Auftrag	fertigen :Operation Pre(Auftrag.State = kommissioniert) Post(State = gefertigt) Err(KeineFertigung -> Auftrag.State = kommissioniert)
4.	:Produ-ktion	ausliefern lassen :Action	:Interaction	:Versand	ausliefern :Operation In(:Auftrag) Pre(Auftrag.State = gefertigt) Post(Auftrag.State = geliefert)
4.1.	:Versand	liefert :Action	:Interaction	:Kunde	erhalten :Operation In(:Auftrag)

Skript 194: Standardverlauf eines Geschäftsprozesses

Eine zusätzliche Möglichkeit findet sich im folgenden Skript in der Interaktion `Stornierung` . Im Gegensatz zu den anderen Ausnahmen wird diese nicht durch eine Operation ausgelöst, sondern stellt ein Ereignis dar, auf das hier die Auftragsannahme beispielsweise dadurch reagiert, dass sie in der folgenden Interaktion die Fertigung storniert.[1] Mit Hilfe dieser Technik werden insbesondere solche Geschäftsprozesse modelliert, für die sich keine operative Prozessstruktur feststellen lässt, da sie tendenziell fast nur aus Ausnahmebehandlungen bestehen.

[1] In einem realistischeren Modell würde die Auftragsbearbeitung den Status des Auftrags erfragen und in Abhängigkeit davon die entsprechende Abteilung informieren.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Kundendatei	Kunde nicht beliefern :Action when(Schlechter Kunde)	Situation a :Interaction Kind (Exception)	:Aufan.	ablehnen Bestellung :Operation In(:Auftrag) Pre(Auftrag.State = any) Post(Auftrag.State = gelöscht)
	:Kundendatei	Kundendaten fehlen :Action when(Kunde nicht vorhanden)	Situation b :Interaction Kind (Exception)	:Vertriebs-system	liefere Personendaten :Operation
	:Auftrag	wird nicht gefertigt :Action when(Keine Fertigung)	Situation c :Interaction Content(self :Auftrag) Kind (Exception)	:Versand	ausliefern2 :Operation In(:Auftrag) Pre(Auftrag.State = kommissioniert) Post(Auftrag.State = geliefert)
1.	:Lager	Ware fehlt :Action when(Ware fehlt)	Situation d :Interaction Kind (Exception)	:Einkauf	Ware bestellen :Operation In(:Auftrag)
1.1.	:Einkauf	unterbrechen :Action	:Interaction	:Auftrag	unterbrochen :Operation Vis(qualified) Exp(:Einkauf, :Lager, :Produktion, :Vertrieb) Pre(State = any) Post(State = unterbrochen)
1.2.	:Einkauf	bestellt Ware :Action	:Interaction	:Lieferant	liefert Ware :Operation
1.3.	:Lieferant	liefert Ware :Action	:Interaction	:Einkauf	erhält bestellte Ware :Operation
1.4.	:Einkauf	reaktivieren :Action	:Interaction	:Auftrag	reaktivieren :Operation Pre(State = unterbrochen) Post(State = aktiv)
2.	:Einkauf	übergibt eingetroffene Ware :Action	:Interaction	:Lager	kommissionieren :Operation In(:Auftrag) Pre(Auftrag.State = geprüft) Post(Auftrag.State = kommissioniert) Err(Ware fehlt: Ware nicht auf Lager -> Auftrag.State = nicht kommissioniert)
4.	:Kunde	storniert Auftrag :Action when(Auftrag wird storniert)	Stornierung :Interaction Kind (Exception)	:Aufan.	stornieren Bestellung :Operation In(:Auftrag)
4.1.	:Aufan.	storniert Auftrag :Action	:Interaction	:Produktion	stornieren :Operation In(:Auftrag) Pre(Auftrag.State = any) Post(Auftrag.State = gelöscht)

Skript 195: Ausnahmebehandlung im Geschäftsprozess

7.3.9 Prozessebenen

MILLING unterscheidet bei der Modellierung von Prozessen die vertikale und die horizontale Problemdimension. Die vertikale Auflösung legt den Detaillierungsgrad des Modells fest. Die horizontale Auflösung legt die einzubeziehenden Elemente durch die Definition einer Systemgrenze fest.[1] Bei der vertikalen Auflösung unterscheidet HARRINGTON auf der obersten Ebene Prozesse, die auf der nächsten Ebene in Aktivitäten zerfallen und die auf der dritten Ebene aus Aufgaben bestehen.[2] DAVENPORT unterscheidet Interorganisationsprozesse, welche die Grenzen von Unternehmen überschreiten, Interfunktionsprozesse, die innerhalb einer Organisation ausgeführt werden, und Interpersonenprozesse, an denen einige wenige Personen oder eine einzelne Abteilung beteiligt ist. Auf der untersten Ebene finden sich stellenbezogene Prozesse, die vollständig von einer Person ausgeführt werden.[3] RUMMLER und BRACHE unterscheiden drei Ebenen der Prozessbetrachtung:[4]

- Auf oberster Ebene werden die Beziehungen eines Unternehmens mit seiner Umgebung betrachtet.[5]
- Auf der Prozessebene werden die Abläufe innerhalb des Unternehmens betrachtet.[6]
- Auf der Ausführungsebene werden die einzelnen Aktivitäten betrachtet.[7]

Dies wird von CORSTEN auch als Makro-, Meso- und Mikroebene bezeichnet.[8] Die *Makrostruktur* ist die Unternehmensebene, die *Mesostruktur* ist die Ebene der Organisationseinheiten, die *Mikrostruktur* ist die Ebene der Stellen.[9] Die Modellierung von Prozessen auf der Makroebene (vgl. Abschnitt 7.3.9) wird insbesondere verwendet, um den Kontext darzustellen. Der *Kontext* zeigt den Leistungsfluss zwischen Prozessen[10] oder Aufgabenträgern.[11] Die verschiedenen

---

[1] Vgl. [Milling 1981: S. 104]. Siehe zu diesem Problem auch [Gaitanides 1983: S. 79].

[2] Vgl. [Harrington 1991: S. 30].

[3] Vgl. [Davenport 1993: S. 90 f.]. Vgl. auch [Scholz 1994: S. 83], [Engelmann 1995: S. 48].

[4] Vgl. [Rummler 1991: S. 15 ff.].

[5] Vgl. [Rummler 1991: S. 31 ff.].

[6] Vgl. [Rummler 1991: S. 44 ff.].

[7] Vgl. [Rummler 1991: S. 64 ff.].

[8] Vgl. [Corsten 1997b: S. 16]. In ähnlicher Weise unterscheidet KRÜGER die Makro- und die Mikrostruktur. Für die Untersuchung der Makrostruktur schlägt er Wertschöpfungsketten vor. Vgl. [Krüger 1994: S. 122 ff.].

[9] Vgl. [Scholz 1994: S. 27].

[10] Vgl. [Österle 1995: S. 79].

[11] Vgl. [Scheer 1998b: S. 10].

Detaillierungsgrade von Prozessen führen zu einer *Prozesshierarchie*.[1] Die Ebenen dieser Struktur heißen *Prozessebenen*.[2] Die Prozesse setzen sich aus Teilprozessen zusammen.[3] Diese Teilprozesse weisen wieder alle Merkmale von Prozessen auf.[4]

Unabhängig davon, ob es sich um Aktivitäten der Makro-, Meso- oder Mikroebene handelt, werden in der OBA++ alle Aktivitäten durch Aktionen und Operationen der beteiligten Objekte abgebildet. Dieses Prinzip wurde gewählt, um auf allen Ebenen immer die gleichen Primitive verwenden zu können. Tatsächlich zeigt nämlich die praktische Erfahrung, dass beispielsweise eine exakte Abgrenzung, was als Prozess, Subprozess, Funktion oder Aktivität betrachtet werden soll, kaum gelingt bzw. von subjektiven Faktoren abhängig ist, wie auch STAUD schreibt.[5] In der OBA++ werden keine vordefinierten Ebenen bei der GPM verwendet, weil diese kaum voneinander abzugrenzen sind. Deshalb wird auch im Metamodell des Konzeptuellen Schemas nicht zwischen Geschäftsprozessen, Subprozessen, Aufgaben etc. unterschieden: In der Praxis sind diese Begriffe kaum zu unterscheiden. Ihnen ist aber in jedem Fall gemeinsam, dass sie eine Aktivität darstellen. Um allen Abgrenzungsproblemen zu entgehen, stellen sie im vorliegenden Metamodell Aktivitäten (Metatyp *ACTIVITY*) dar.

Die vertikale Modellauflösung erfolgt also durch Subskripte, in denen Makrosichten durch Meso- und schließlich durch Mikrosichten verfeinert werden (vgl. Abschnitt 6.12.2). Dies soll an folgendem Beispiel demonstriert werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Gast Stpe(Individual) System(external)	wünscht Beherbung :Action	:Interaction Flow(Service)	:Hotel Stpe(Organization)	wickelt Beherbung ab :Operation
	:Hotel Stpe(Organization)	erzeugt Abfälle :Action	:Interaction Flow(Material)	:Entsorgungsbetrieb Stpe(Organization) System(external)	entsorgt Abfälle :Operation
	:Hotel Stpe(Organization)	benötigt Schutz :Action	:Interaction Flow(Service)	:Wachdienst Stpe(Organization) System(external)	liefert Objektschutz :Operation
	:Hotel Stpe(Organization)	lässt Schmutzwäsche reinigen :Action	:Interaction Flow(Material)	:Wäscherei Stpe(Organization) System(external)	reinigt Wäsche :Operation
	:Hotel Stpe(Organization)	richtet Zimmer ein :Action	:Interaction Flow(Material)	:Möbelhaus Stpe(Organization)	liefert Mobiliar :Operation

[1] Vgl. [Fischermanns 1997: S. 27], [Schulte-Zurhausen 1999: S. 87], [Gaitanides 1983: S. 75 ff.], [Haist 1989: S. 96], [Scholz 1994b: S. 39], [Schmelzer 2002: S. 79].

[2] Vgl. [Scholz 1994b: S. 45].

[3] Vgl. [Bullinger 1996: S. 96].

[4] Vgl. [Schulte-Zurhausen 1999: S. 87].

[5] Vgl. [Staud 2001: S. 6].

				System(external)	
	:Hotel Stpe(Organization)	beauftragt Werbeagentur :Action	:Interaction Flow(Service)	:Werbeagentur Stpe(Organization) System(external)	wirbt für Hotel :Operation
	:Hotel Stpe(Organization)	erhält Gäste vermittelt :Action	:Interaction Flow(Service)	:Fremdenverkehrs- verein Stpe(Organization) System(external)	vermittelt Gäste :Operation

Skript 196: Makrostruktur eines Hotels

Skript 196 stellt die Makrostruktur – den Kontext – der Geschäftsprozesse eines Hotels dar. Wendet man das Prinzip der objektorientierten Verfeinerung (vgl. Abschnitt 6.12.2) an, erhält man für die erste Interaktion folgendes Skript, welches im o. g. Sinne die Mesostruktur eines Prozesses darstellt. Dabei wird das Hotel in einzelne Organisationseinheiten zerlegt. Im Gegensatz zur Methode SOM kann hier die Zerlegung der Organisation `Hotel` in die einzelnen Organisationseinheiten durch Aggregationsbeziehungen dokumentiert werden, wie dies schon in Skript 18 demonstriert wurde.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Gast Stpe(Individual) System(external)	wünscht Information :Action	:Interaction	:Marketing Stpe(Org-Unit)	versendet Prospekt :Operation
	:Gast Stpe(Individual) System(external)	reserviert Zimmer :Action	:Interaction	:Rezeption Stpe(Org-Unit)	Zimmer reservieren :Operation
	:Gast Stpe(Individual) System(external)	übernachtet :Action	:Interaction	:Zimmer Stpe(Org-Unit)	werden zur Übernachtung genutzt :Operation
	:Gast Stpe(Individual) System(external)	frühstücken :Action	:Interaction	:Restaurant Stpe(Org-Unit)	verpflegt Gast :Operation
	:Gast Stpe(Individual) System(external)	abreisen :Action	:Interaction	:Rezeption Stpe(Org-Unit)	Gast auschecken :Operation

Skript 197: Mesostruktur einer Beherbergung

Im Rahmen der Zimmerreservierung wird die Rezeption eine Reihe von Aktivitäten ausführen. Die zweite Zeile des letzten Skripts erhält nun ein weiteres Unterskript. Hier ist die Ebene interner Kunden-Lieferanten-Beziehungen erreicht:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Rezeption Stpe(Org-Unit)	reserviert Zimmer :Action	:Interaction	:Zimmerpool	freies Zimmer reservieren :Operation

	:Rezeption Stpe(Org-Unit)	informiert Reinigungspersonal :Action	:Interaction	:Reinigungspersonal Stpe(Group)	reinigt Zimmer :Operation
	:Rezeption Stpe(Org-Unit)	Anmeldung vorbereiten :Action	:Interaction	:Rezeptionist Stpe(Task Bearer)	Anmeldeunterlagen erstellen :Operation
	:Rezeption Stpe(Org-Unit)	Zimmerstand melden :Action	:Interaction	:Fremdenverkehrs-verein Stpe(Organization)	wird über verfügbare Zimmer informiert :Operation

Skript 198: Vorgang einer Zimmerreservierung

Durch Ebenenbildung werden sowohl Objekte, als auch Operationen globaler Sichten in elementareren Sichten zerlegt. Geschieht diese Zerlegung durch Subskripte, ist durch den verwendeten Abbildungsprozess über das Konzeptuelle Schema immer festzustellen, ob eine Operation in einer feingranulareren Sicht „verfeinert" wird. Ist dies nicht der Fall, geht von der Operation keine Abstraktionsbeziehung mehr aus. Die simultane Modellierung der dynamischen und statischen Sicht ermöglicht dabei, die Zerlegung der beteiligten Objekte explizit darzustellen.

Durch Abstraktionsbeziehungen (vgl. 4.3.6.3, 6.12) können Skripte beliebig verfeinert bzw. präzisiert werden. Ob dabei Vorgänge bis auf die Ebene der Mikrostruktur[1] zerlegt werden, ist vom Einzelfall abhängig, aber prinzipiell möglich. Wenn es sich als notwendig erweist, können durchaus einzelne Handgriffe modelliert werden, auch wenn dies im Rahmen der GPM nicht üblich ist. Festzuhalten ist aber ein wesentlicher Unterschied zwischen der GPM und dem OA: Der OA basiert auf der Zerlegung eines Systems in dessen konstituierende Objekte, wogegen die GPM auf der funktionalen Zerlegung von Prozessen beruht.[2]

Wie schon mehrfach erwähnt, findet die Definition der Systemgrenze – also die horizontale Auflösung – durch das Prädikat `System()` statt. Auch für externe (Umwelt-)Objekte können die unterschiedlichen Ebenen verwendet werden.

An den vorangegangenen Beispielen dieses Abschnitts ist aber auch deutlich geworden, dass das in Abschnitt 7.2.3 erwähnte Zuordnungsproblem auf den zweiten Blick ein geringes Problem ist. Eine Zuordnung ist immer nur auf der Betrachtungsebene notwendig. Wenn also Prozesse auf der Makroebene betrachtet werden, ist die personale Zuordnung auch nur auf dieser Ebene notwendig.

---

[1] Werden die Teilaufgaben niedrigster Ordnung (die *Elementaraufgaben*) weiter zerlegt, spricht man von Arbeitsgängen, Arbeitsstufen, Griffen und Griffelementen. Die Mikrostruktur entspräche den Gang- oder Griffelementen der betriebswirtschaftlichen Organisationslehre und hier insbesondere der Ablauf-organisation. Vgl. [Kosiol 1962: S. 189, 200 f.], [Schulte-Zurhausen 1999: S. 43]. Man kann aber feststellen, dass in der Literatur zur GPM solche Elementaraufgaben in der Regel nicht weiter zerlegt werden.

[2] Vgl. Abschnitt 2.3.2.

Wie man in Skript 196 sieht, wurden die Aufgaben dem Aufgabenträger Hotel zugeordnet. Mit zunehmendem Detaillierungsgrad wird auch die Aufgabenzuordnung präzisiert.

7.3.10 Externe Prozessverkettung

Geschäftsprozesse werden auch unternehmens-[1] bzw. organisationsübergreifend gestaltet,[2] was als *externe Prozessverkettung*[3], *externe Prozessvernetzung*[4] oder *unternehmensübergreifende Supply Chain*[5] bezeichnet wird.[6] In diesem Fall überschreitet der Geschäftsprozess die Grenze des betrachteten Systems. Einige dieser Verbindungen repräsentieren den physischen Austausch von Gütern, aber bei allen Verbindungen werden Informationen ausgetauscht.[7] Eine automatisierte zwischenbetriebliche Kommunikation ist für eine Vielzahl von Prozessen möglich und bietet eine Reihe von Vorteilen, weswegen ein Modellierungsansatz die Darstellung externer Prozessverkettungen unterstützen sollte. SCHULTE-ZURHAUSEN nennt folgende Vorteile einer externen Prozessverkettung:[8]

- Die Kommunikation erfolgt sehr schnell, da Daten ohne manuelle Eingriffe übermittelt werden.
- Fehler werden durch die fehlende manuelle Bearbeitung vermieden.
- Durch die Vermeidung der wiederholten Erfassung, Eingabe und Ablage der gleichen Daten werden Prozesskosten reduziert.
- Die Daten können beim Empfänger direkt weiterverarbeitet werden; dies reduziert die Durchlaufzeiten.
- Die schnelleren Reaktionsmöglichkeiten sorgen für Wettbewerbsvorteile.

In der OBA++ ist eine externe Prozessverkettung am Prädikat `System(external)` entweder von Initiator oder Participant zu erkennen (vgl. Abschnitt 6.1.3). Wie man am Beispiel des Komponentenherstellers sieht, gehören auch die Interaktionen mit der Bank zum Geschäftsprozess.

---

[1] Vgl. [Johann 2000], [Alt 2000], [Staud 2001: S. 16 f.], [Wassermann 2001: S. 249 ff.].

[2] Vgl. [Scholz 1994: S. 83], [Alt 2000], [Davenport 1993]. Dies wird von GEHRING als *interorganisatorische Prozessintegration* bezeichnet. Vgl. [Gehring 1998: KE 1, S. 50].

[3] Vgl. [Schulte-Zurhausen 1999: S. 101].

[4] [Krüger 1994: S. 127].

[5] Vgl. [Wassermann 2001: S. 250].

[6] Vgl. auch folgendes Zitat: *„Der Prozessentwurf betrachtet Abläufe ohne Rücksicht auf organisatorische Grenzen, weder innerbetriebliche noch zwischenbetriebliche.“* [Österle 1995: S. 38].

[7] Vgl. [Schulte-Zurhausen 1999: S. 101 f.].

[8] Vgl. [Schulte-Zurhausen 1999: S. 102 f.].

7.4 Analyse von Geschäftsprozessen

7.4.1 Zeitliche Struktur

Prozesse vollziehen sich in einem zeitlichen Rahmen, der als Durchlaufzeit bezeichnet wird.[1] „*Eine genaue Kenntnis der Durchlaufzeit ist sowohl für materielle als auch für informationelle Prozesse von Bedeutung.*“[2] Nach HARRINGTON ist die Reduzierung von Prozessdurchlaufzeiten eine wichtige Maßnahme der Prozessverbesserung.[3] Die Bedeutung der Analyse zeitlicher Abhängigkeiten wird dadurch unterstrichen, dass nur 20 % bis 40 % der Auftragsdurchlaufzeiten durch die direkten Bereiche determiniert werden und 90 % der Gesamtdurchlaufzeit von Prozessen nicht wertschöpfend sind, also zum Beispiel Wartezeiten darstellen.[4]

Geringe Durchlaufzeiten haben folglich eine große Wirkung auf die Kundenzufriedenheit: „*Durchlaufzeiten, die externe Kunden betreffen, haben einen entscheidenden Einfluss auf die Kundenzufriedenheit und stehen darum häufig im Mittelpunkt von Optimierungsanstrengungen, die im Rahmen einer prozessorientierten Umgestaltung der Organisation eines Unternehmens unternommen werden.*“[5] Insbesondere bei ähnlichen Produkten bringen Zeitvorteile Wettbewerbsvorteile.[6] Bei der GPM spielt das Phänomen Zeit (Zeitpunkte, Zeiträume und Zeitdauern) folglich eine wichtige Rolle.[7] Dazu gehören Durchlaufzeiten, Bearbeitungszeiten, Transportzeiten, Liegezeiten und Wartezeiten.[8]

Im Gegensatz zu anderen Methoden im Bereich der GPM[9] kann in der OBA++ durch das Prädikat `TimingConstraint()`, welches für alle Aktivitäten und alle dynamischen Beziehungen

---

[1] Vgl. [Schulte-Zurhausen 1999: S. 51].

[2] Vgl. [Schulte-Zurhausen 1999: S. 72 f.]. Vgl. auch [Österle 1995: S. 161 ff.], [Kueng 2000: S. 831], [Schmelzer 2002: S. 156 ff.].

[3] Vgl. [Eversheim 1995: S. 52 ff.], [Harrington 1991: S. 146 ff.].

[4] [Eversheim 1995: S. 29]. Ähnliche Werte finden sich in [Schwetz 1993: S. 298], [Krüger 1994: S. 125] und [Wassermann 2001: S. 8].

[5] [Turowski 1996: S. 210].

[6] [Bullinger 2001].

[7] Vgl. [Hoheisel 1999: S. 108], [Klepzig 1997: S. 85], [Gehring 1998: KE1, S. 69].

[8] Vgl. [Schulte-Zurhausen 1999: S. 51, 72 ff.], [Krüger 1994: S. 126], [Eversheim 1995: S. 47 ff.], [Gaitanides 1983: S. 91], [Fischermanns 1997: S. 51 f.], [Oberweis 1997]. Erstaunlich ist, wie schwach in der verbreitetsten Methode zur Modellierung von Geschäftsprozessen die Modellierung zeitlicher Abhängigkeiten unterstützt wird, wie man an der autoritativen Monographie der Methode ARIS feststellen kann. Vgl. [Scheer 1998]. Wie weit hier schon frühere Ansätze waren, ist an [Corsten 1986] festzustellen.

[9] Vgl. [Staud 1999: S. 49], [Staud 2001: S. 135].

definiert ist, sowohl der maximale wie auch der minimale Zeitverbrauch dokumentiert werden. Außerdem können zulässige Zeitabstände zwischen den Aktivitäten einer Interaktion und zwischen den Zeilen eines Skripts definiert werden (vgl. Abschnitte 6.2.14, 6.6.11, 6.8, 6.11.6). Im einfachsten Fall bedeutet dies, dass für einen Geschäftsprozess die maximal notwendige Ausführungsdauer festgelegt wird:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :Person Stpe(Individual) System(external)	bittet um Angebot	:Interaction	:Sachbearbeiter Stpe(Position)	liefere Angebot :Operation Goal(Kunde hat Angebot) In(:Artikeldaten) Out(:Angebot) TimingConstraint (executionTime() < 2 d)

Skript 199: Ausführungsdauer eines Geschäftsprozesses

Versieht man die Aktionen, Operationen und Interaktionen der Realisierung dieser Leistung in einem Subskript oder in geschachtelten Interaktionen (siehe Skript 192) ebenfalls mit zeitlichen Restriktionen, kann man feststellen, ob diese zeitliche Restriktion eingehalten werden kann und wer und wofür die meiste Zeit „verbraucht“. Allerdings ist die Berechnung der Durchlaufzeit eines Prozesses nicht einfach durch Addition der Ausführungszeiten aller Aktivitäten und Interaktionen möglich. Hierbei sind die Synchronisation und Ausbreitungsart der Interaktionen, die Fristen in zeitlich begrenzten Interaktionen, das Ausnahmeverhalten und die operative Prozessstruktur zu berücksichtigen. Dieses Problem sei exemplarisch an folgender Interaktion erläutert:

1.3	:Sachbearbeiter Stpe(Position)	berechnen :Action	:Interaction Prop(Multicast, Anzahl der Positionen, seq)	:Angebotsposition	berechne Abschlag :Operation Pre(Artikeldaten vorhanden; Kundendaten vorhanden) Post(Angebotsposition berechnet) TimingConstraint (executionTime() < 5 s)

Unter der Bedingung, dass die Artikel- und Kundendaten bekannt sind, kann eine Angebotsposition in 5 Sekunden berechnet werden. Die Gesamtdauer der Interaktion hängt aber von der Anzahl der Angebotspositionen ab, die berechnet werden. Hier empfiehlt sich, die Gesamtzeit einer Interaktion inklusive Fristen und der Ausbreitungsart als Ausführungszeit der Aktionen des Initiators festzuhalten.

Zusätzlich hat man in der OBA++ die Möglichkeit, durch die Kombination der Prädikate `Concurrency()` und `Suspend()` die Synchronisation von Initiator und Participant, abwartende Aktivitäten, ungültig werdende und nach Fristablauf abbrechende Interaktionen abzubilden (vgl. Abschnitte 6.2.6, 6.2.7, und 6.2.8). Diese Möglichkeiten sind aus keiner anderen Methode zur GPM bekannt. Durch das Verknüpfen von Interaktionen durch Abhängigkeitsbeziehungen können außerdem zeitliche Abstände zwischen Ereignissen festgelegt werden.

Allerdings ist ein wesentlicher Unterschied zwischen der GPM und der ooSe festzustellen. Bei der Modellierung von Softwaresystemen wird auch heute noch in der Regel der Faktor Zeit ignoriert, was bei der GPM besser nicht der Fall sein sollte. Diese Tatsache hat aber weitreichende Konsequenzen, wenn zum Beispiel auf der Seite des Participants sowohl das Prädikat `In()` als auch das Prädikat `Out()` verwendet wird, also ein bidirektionaler Fluss zwischen Initiator und Participant abgebildet wird. In diesem Fall muss – bei genauer Betrachtung – eine synchrone oder sequentielle Synchronisation vorliegen. Die Konsequenz sei noch einmal am Beispiel von Skript 174 erläutert. Wenn der Kunde eine Bestellung tätigt und im Rahmen dieser Interaktion die Ware erhält (= bidirektionaler Fluss), hat das zur Folge, dass er von der Ausführung weiterer Aktivitäten blockiert ist. Das ist mit Sicherheit eine etwas realitätsferne Annahme. Tatsache ist, dass man bei der GPM besser nur unidirektionale Interaktionen verwendet, außer man will tatsächlich ausdrücken, dass der Initiator von weiteren Aktivitäten blockiert ist.

Dass in der praktischen Anwendung (auch hier) trotzdem die Version aus Skript 174 verwendet wird, hat pragmatische Erwägungen. In dieser (weniger präzisen) Fassung ist der Sinn der Operation – also die Leistung des Geschäftsprozesses – klarer. Meistens beschäftigt man sich mit Fragen der Synchronisation gar nicht erst. In dem Moment aber, in dem bei der GPM Zeitaspekte untersucht werden, spielen auch Synchronisationsgesichtspunkte eine Rolle. Dann ist die präzisere Formulierung zu verwenden.

7.4.2 Entlinearisierung und Synchronisationsaufwand

Auf der einen Seite können durch die Parallelisierung bisher sequentieller Abläufe Durchlaufzeiten reduziert werden.[1] Unter *Entlinearisierung* versteht HAMMER die parallele Ausführung von Aktivitäten, wenn eine sequentielle Ausführung nicht notwendig ist.[2] Auf der anderen Seite ist in Geschäftsprozessen insbesondere die notwendige Synchronisation paralleler Abläufe kritisch.[3] Der Parallelisierung sequentieller Abläufe steht der erforderliche Zeitaufwand für die Aufteilung der

[1] Vgl. [Fischermanns 1997: S. 216 ff.], [Eversheim 1995: S. 31, 133], [Harrison 1993: S. 7], [Harrington 1991: S. 147], [Schulte-Zurhausen 1999: S. 105].

[2] Vgl. [Hammer 1994: S. 75 ff.].

[3] Vgl. [Ferstl 1996: S. 58].

Arbeit, die Koordination der Arbeitsteilung und die Synchronisation der Teilergebnisse zu einem Ganzen gegenüber.[1] Dieser zusätzliche Zeitaufwand für die Aufteilung der Arbeit sowie die Koordination der Arbeitsteilung und die Zusammenführung/Synchronisation der Teilergebnisse zu einem Ganzen muss aus dem Modell ersichtlich werden. Dabei ist auch darauf zu achten, dass die Zeitdauer der parallel laufenden Prozessketten möglichst gleich ist.[2]

Hierzu sei folgendes Beispiel konstruiert: Zwei Aufgabenträger üben sequentiell vier Verrichtungen an einem Geschäftsobjekt aus. Der dafür notwendige Zeitaufwand ist durch das Prädikat `TimingConstraint()` festgehalten. Die Gesamtprozesszykluszeit beträgt 8 Stunden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:A	w durchführen :Action	:Interaction	:GO 1	w durchgeführt :Operation TimingConstraint(executionTime() = 1 h)
2.	:A	x durchführen :Action	:Interaction	:GO 1	x durchgeführt :Operation TimingConstraint(executionTime() = 2 h)
3.	:B	y durchführen :Action	:Interaction	:GO 1	y durchgeführt :Operation TimingConstraint(executionTime() = 3 h)
4.	:B	z durchführen :Action	:Interaction	:GO 1	z durchgeführt :Operation TimingConstraint(executionTime() = 2 h)

Skript 200: Ausgangspunkt der Entlinearisierung

Im Rahmen einer Entlinearisierung werden die Interaktionen 1. und 2. sowie 3. und 4. parallelisiert. Zusätzlich muss das Geschäftsobjekt `GO 1` in die Geschäftsobjekte `GO 1a` und `GO 1b` zerlegt und hinterher wieder zusammengefügt werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:A	zerlegt GO 1 in Teile :Action	zerlegen :Interaction	:GO 1	wird in GO1a und GO 1b zerlegt :Operation TimingConstraint (executionTime() = 0,5 h)
2.	:A	übergibt :Action	übergeben :Interaction Concurrency(synchronous)	:B	erhält GO 1b :Operation TimingConstraint (executionTime() = 0,01 h)
2. / B3.	:A	w durchführen :Action	:Interaction	:GO 1a	w durchgeführt :Operation TimingConstraint(executionTime() = 1 h)
B4.	:A	x durchführen :Action	:Interaction	:GO 1a	x durchgeführt :Operation TimingConstraint(executionTime() = 2 h)
2. / A3.	:B	y durchführen :Action	:Interaction	:GO 1b	y durchgeführt :Operation TimingConstraint(executionTime() = 3 h)
A4.	:B	z durchführen	:Interaction	:GO 1b	z durchgeführt :Operation

[1] Vgl. [Fischermanns 1997: S. 218].

[2] Vgl. [Fischermanns 1997: S. 219].

		:Action			TimingConstraint(executionTime() = 2 h)
B4. & A4. / 5.	:B	übergibt :Action	montieren :Interaction Concurrency(synchronous)	:A	fügt GO1a und GO 1b zusammen :Operation TimingConstraint (executionTime() = 0,5 h)

Skript 201: Entlinearisierter Prozessverlauf

Die Synchronisation der Teilergebnisse ist durch die drei zusätzlichen Interaktionen `zerlegen`, `übergeben` und `montieren` abgebildet worden. Aus dem Sequenzausdruck der Interaktion `montieren` wird deutlich, dass diese Interaktion zur Synchronisation der parallel laufenden Prozessketten notwendig ist, weil die Objekte `GO 1a` und `GO 1b` erst montiert werden können, wenn die Interaktionen mit den Sequenzausdrücken `B4.` und `A4.` abgeschlossen sind. Die Gesamtprozesszykluszeit beträgt in diesem Fall 6,01 (= 0,5 +0,01 +3 +2 +0,5) Stunden, wobei die Interaktionen mit den Sequenzausdrücken `2./A3.` und `A4.` maßgeblich für diesen Wert ist. Der zusätzliche Zeitaufwand für die Aufteilung der Arbeit und die Synchronisation der Teilergebnisse wird deutlich, aber auch die Zeitersparnis durch die Entlinearisierung.

7.4.3 Kosten

Da z. B. die steigenden Kosten der Informationsverarbeitung sowie planende und steuernde Funktionen über Gemeinkostenzuschläge verteilt werden, mangelt es an der Kostentransparenz in Unternehmen.[1] Ein weiteres Ziel der GPM ist deshalb die Abbildung von Kosten.[2] „*Die Kostenanalyse untersucht die Kosten der Prozessleistung und die Anteile der einzelnen Aufgaben.*“[3] Dies umfasst die direkte Zuordnung von Kosten zu Aktivitäten, die Zuordnung von Kosten zu Ressourcen, Personalkosten, Kommunikationskosten, Lagerhaltungskosten etc.[4]

Allen von Objekten ausgeführten Aktivitäten kann man in der OBA++ Kosten zuordnen. Dadurch wird sichtbar, wer im Prozessverlauf für welche Aktivität welche Kosten verursacht. Wie in Abschnitt 6.6.12 geschildert, können dabei Aktivitätentreiber und Aktivitätenkostensätze

---

[1] [Eversheim 1995: S. 33].

[2] Vgl. [Raster 1994b], [HeilmannM 1997: S. 91], [Ferstl 1998], [Gehring 1998: KE 2, S. 51], [Scheer 1998b: S. 66 ff.], [Schulte-Zurhausen 1999: S. 74]. Hier ist festzustellen, dass – abgesehen von den Arbeiten von EVERSHEIM und HEILMANN – die aktuellen Methoden die Notwendigkeit der Modellierung von Kosten zwar erwähnen, aber im Rahmen ihrer Methoden nicht unterstützen. Vgl. [Eversheim 1995: S. 31, 80 ff.], [HeilmannM 1997].

[3] [Österle 1995: S. 163].

[4] Vgl. [Oberweis 1997].

verwendet werden. Als Alternative bietet sich an, Kostenstellen und Kostenarten als Objekte zu modellieren und die Kostenträger und Kostenstellen mit den Kosten zu belasten, wie dies in Skript 111 demonstriert wurde.

Die Anschaffungskosten einer Ressource kann modelliert werden, indem man ihr beispielsweise eine Operation `anschaffen` zuordnet:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Einkauf Stpe(Department)	anschaffen :Action	:Interaction	:Notebook	anschaffen :Operation Cost(12.998 GE)

Sollen Kommunikationskosten erfasst werden, muss das Übertragungsmedium explizit als Objekt modelliert werden, wie dies in Abschnitt 6.2.14 und in Skript 148 geschehen ist. Für die Berechnung der Gesamtkosten gelten allerdings die in Abschnitt 7.4.1 gemachten Anmerkungen analog.

Auch Transaktionskosten[1], also die Kosten für die Anbahnungs-, Vereinbarungs- und Leistungs- und Akzeptanzphase von Prozessen, lassen sich in der OBA++ unterscheiden. Sie sind durch das Prädikat `Coord()` der Interaktion erkennbar:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	...	...	...	...	...
	:Verkauf	bietet Leistung an :Action Cost(800 GE)	B :Interaction Coord(request)	/Kunde :Person	akzeptiert Angebot :Operation
	...	...	...	...	...
	/Kunde :Person	bezahlt Ware :Action	E :Interaction Coord(evaluation)	:Auftrags- bearbeitung	gleicht Offenen Posten aus :Operation Cost (48 GE)

Skript 202: Abbildung von Transaktionskosten

7.4.4 Mengengerüste

Mengengerüste quantifizieren nach FISCHERMANNS Aufgaben.[2] Dies sind unter anderem die Anzahl der Aufgabenträger, die Anzahl der zur Verfügung stehenden Sachmittel, die Anzahl der Aufgaben (Fallzahlen) oder die Kommunikationshäufigkeit zwischen bestimmten Aufgabenträgern.

[1] Vgl. [Picot 1982], [Picot 1990], [Picot 1999], [HeilmannM 1997: S. 180 ff.].

[2] Vgl. [Fischermanns 1997: S. 54 f.].

Wird die Anzahl der Objekte einer Klasse prinzipiell beschränkt, geschieht dies durch das Prädikat `Card()` einer Klasse. Auf diese Weise kann die Anzahl von Aufgabenträgern und Sachmitteln, aber auch von jedem anderen Objekttyp eingeschränkt werden. Soll dagegen einem Objekt eine bestimmte Anzahl anderer Objekte zugeordnet werden, stellt dies die Eigenschaft einer referentiellen Beziehung dar. Im folgenden Skript wird zum Beispiel festgelegt, dass die Werbeabteilung genau 3 Farbdrucker besitzt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Werbung Stpe(Department)		verfügt über :Association DomCard(1) RngCard(3)	Farbdrucker	

Skript 203: Beispiel einer Assoziation auf Typebene

Dagegen wird im Folgenden (nur zum Teil ausgefüllten Skript) festgelegt, dass es im betrachteten System nur genau einen Geschäftsführer gibt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Geschäftsführer Stpe(Task Bearer) System(internal) Card(1)				

Skript 204: Beispiel zur Anzahl eines Aufgabenträgers

Die Fallzahlen und die Kommunikationshäufigkeit zwischen Aufgabenträgern werden in der OBA++ durch sich periodisch wiederholende Zeitereignisse mit Hilfe des Prädikats `periodic()` abgebildet. Da jede Interaktion die Ausführung einer Operation zu Folge hat, ist auf diese Weise sowohl die Häufigkeit der Interaktion wie auch die Häufigkeit der auszuführenden Aufgabe festgelegt:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Personal Stpe(Department)	Reisekostenabrechnung einfordern :Action periodic(t0 = freitags, 12:00 h, In = True)	:Interaction	:Mitarbeiter Stpe(Individual)	Reisekosten-abrechnung erstellen :Operation

Skript 205: Beispiel zur Kommunikationshäufigkeit

7.4.5 Koordinationsprotokolle

Das *Action-Workflow-Protokoll* wird in der Literatur verwendet, um die Interaktionen zwischen Kunde und Lieferant zu strukturieren.[1] Das Protokoll definiert die Phasen (Anbahnung, Vereinbarung, Leistung, Akzeptanz), die der Kunde im Verlauf des Geschäftsprozesses durchläuft.[2] Danach sollen Geschäftsprozesse bei einem Kunden beginnen und bei einem Kunden enden.[3] Durch die letzte Phase des Action-Workflow-Protokolls wird dem Lieferanten einer Leistung mitgeteilt, ob der Kunde die Leistung akzeptiert bzw. ob er Verbesserungswünsche hat.[4] Solche Rückmeldungen helfen dem Lieferanten, die Qualität seiner Produkte oder Dienstleistungen aufrechtzuerhalten oder zu verbessern.[5] Durch die Rückmeldung wird ein geschlossener Kreis beschrieben, der im Idealfall so lange durchlaufen wird, bis der Kunde seine Zufriedenheit erklärt.[6] Solche Kreise sollen für eine schnelle Erfassung von Soll/Ist-Abweichungen und die Durchführung zugehöriger Gegenmaßnahmen sorgen.[7] Bei der Analyse von Geschäftsprozessen wird deshalb insbesondere untersucht, ob alle Phasen des Action-Workflow-Protokolls im Geschäftsprozess vorhanden sind.[8] Im SOM wird dieses Konzept erweitert, da zwei unterschiedliche Koordinationsprotokolle verwendet werden. Neben dem Action-Workflow-Protokoll werden die an einem Geschäftsprozess beteiligten Objekte auch durch Regelung miteinander koordiniert.[9]

Nicht nur das Action-Workflow-Protokoll aus dem Bereich der Workflow-Modellierung ist in der OBA++ durch das Prädikat `Coord()` realisiert worden, sondern auch das Regelungsprotokoll der Methode SOM (vgl. Abschnitt 6.2.4). Das Muster eines (nach dem Action-Workflow-Protokoll) wohlgeformten Geschäftsprozesses lässt sich in der OBA++ daran erkennen, dass im

---

[1] Vgl. [Schäl 1996: S. 36], [Scherr 1993], [Medina-Mora 1992].

[2] Vgl. [Klepzig 1997: S. 83]. WINOGRAD und FLORES haben für dieses Protokoll ein Modell der möglichen Zustandsübergänge entwickelt. [Winograd 1986: S. 65], [Winograd 1987: S. 8], [Schäl 1996: S. 31 ff.]

[3] Vgl. [Schäl 1996: S. 14], [HeilmannM 1997: S. 91]. Man beachte den Hinweis von PICOT (vgl. [Picot 1999: S. 356]), dass die Verwendung von Regelkreisen nur dann erfolgreich ist, wenn die Abnehmerstufe auch in Zukunft dieselben Arten und Mengen verbrauchen wird wie in der Vergangenheit.

[4] Vgl. [Scherr 1993: S. 87].

[5] Vgl. [Haist 1989: S. 159].

[6] Vgl. [Schäl 1996: S. 37], [Denning 1992].

[7] Vgl. [Klepzig 1997: S. 94], J. BRAUN in [Bullinger 1996: S. 118]. Dieses Konzept taucht auch bei [Morabito 1999] auf.

[8] Vgl. [Schäl 1996].

[9] Vgl. [Ferstl 1998: S. 185]. Dies findet sich ähnlich auch bei RUMMLER und BRACHE: Unternehmen sollen wie adaptive Systeme gestaltet werden, die auf die Konsequenzen ihrer Aktivitäten achten. [Rummler 1991: S. 9 ff.].

einfachsten Fall der Geschäftsprozess durch einen Kunden als Initiator begonnen und durch den Kunden als Participant beendet wird. Darüber hinaus müssen alle Phasen (`request`, `commitment`, `performance` und `evaluation`) als Terme des Prädikats `Coord()` verwendet worden sein. Das folgende Skript stellt diesen Fall als Muster dar. Beispiele für die Verwendung des Koordinationsprotokolls finden sich in Abschnitt 5.3.2.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Kunde	...	:Interaction Coord(request)	:Lieferant	...
	...	...	:Interaction Coord(commitment)	...	...
	...	...	:Interaction Coord(performance)	...	...
	:Lieferant	...	:Interaction Coord(evaluation)	:Kunde	...

Skript 206: Muster eines Geschäftsprozesses nach dem Action-Workflow-Protokoll

Ein Vorteil der OBA++ besteht darin, dass weitere Protokolle durch zusätzliche Terme für das Prädikat `Coord()` definiert werden können. Hierfür ist nur der Aufzählungsdatentyp `Coord Identifier` (vgl. Kapitel 11) zu erweitern.

7.4.6 Transformations- und Entscheidungsaufgaben

Da ein Ziel des Prozessansatzes ist, die Trennung von Leistung und Lenkung aufzuheben,[1] muss abgebildet werden können, welche Objekte Transformations- und welche Objekte Entscheidungsaufgaben[2] ausführen. Auch FERSTL und SINZ unterscheiden bei der GPM zwischen dem Leistungs- und Lenkungssystem.[3] Die Durchführungsaufgaben gehören zum Leistungssystem. Die Planungs-, Steuerungs- und Kontrollaufgaben gehören zum Lenkungssystem.[4]

---

[1] HAMMER bringt dies auf die kurze Formel: *Wer die Arbeit macht, soll die Entscheidungen treffen.* [Hammer 1993b: S. 81].

[2] Vgl. [Ferstl 1998: S. 30 f.].

[3] Vgl. [Ferstl 1998: S. 5].

[4] Vgl. [Ferstl 1995: S. 211 f.]. Dieser Begriff lehnt sich an das Analysekriterium der *Phase* von KOSIOL an. Vgl. [Kosiol 1962: S. 53]. Schon WILD unterschied im sog. *SOS-Konzept* Steuerungsaufgaben, operative Aufgaben und Serviceaufgaben, die in allen sozioökonomischen Systemen vorhanden sein müssen. Sie stellen die funktionalen Subsysteme von Organisationen dar. Vgl. [Krüger 1994: S. 37 f.]. FISCHERMANNS bezeichnet diese Subsysteme als Führungsprozesse, Ausführungsprozesse, Unterstützungsprozesse. Vgl. [Fischermanns 1997: S. 36]. Siehe auch die unterschiedlichen Aspekte der Zentralisation bei BLEICHER. Vgl. [Bleicher 1991: S. 49 ff.].

Unter dem Schlagwort *Zusammenführung von Hand- und Kopfarbeit*[1] sollen Aufgaben prozessorientiert zusammengefasst werden. Dies beinhaltet die Integration vorgelagerter Planungs- und nachgelagerter Kontrollaufgaben sowie unterstützende Instandhaltungsaufgaben.[2] Diese Reintegration dispositiver und administrativer Aufgaben bedeutet, dass die Entscheidungskompetenz und Ergebnisverantwortung in der Hierarchie der Leitungsebenen[3] so niedrig wie möglich gelagert sein soll.[4] Die Entscheidungen sollen von den Prozessbeteiligten selbst getroffen werden. HAMMER bezeichnet dies als *vertikale Komprimierung*.[5] Alternativ unterscheidet man auch die horizontale und vertikale Arbeitsintegration.[6] Die *vertikale Arbeitsintegration* bedeutet das Zusammenführen von Arbeiten ungleicher Ebene, also die Integration von Entscheidung und Ausführung. Die *horizontale Arbeitsintegration* bedeutet nur die Integration aufeinander folgender Arbeitsschritte zu einem ganzheitlichen Prozess. Die Trennung von Entscheidung und Ausführung wird jedoch beibehalten.[7]

Die Unterscheidung von Transformations- und Entscheidungsaufgabe, also die Differenzierung des Leistungs- und Lenkungssystems, basiert in der OBA++ auf den Prinzipien, die für die Methode SOM entwickelt wurden. Maßgeblich ist die Verwendung der Terme `control` und `feedback` für das Prädikat `Coord()` der Interaktion. Erhält der Participant eine Nachricht mit dem Term `control`, wird er vom Initiator gelenkt. Erhält der Participant eine Nachricht mit dem Term `feedback`, handelt es sich bei seiner Operation um eine Entscheidungs- bzw. Lenkungsaktivität. Im Beispiel des Komponentenherstellers gehört die Produktionssteuerung zum Lenkungssystem, während die Fertigung zum Leistungssystem gehört. Im Gegensatz zum Ansatz von FERSTL und SINZ, bei dem für die Objekte global festgelegt wird, ob sie zum Lenkungs- oder zum Leistungssystem gehören,[8] wird hier durch die Interaktionen, an denen sie teilnehmen, entschieden, ob sie Lenkungs- oder Leistungsaktivitäten ausüben.

Vertikale Komprimierung bzw. Arbeitsintegration modelliert man, indem ein Objekt sowohl Lenkungs- wie auch Leistungsaktivitäten ausführt. Findet die Leistung und Lenkung eines Prozesses integriert durch ein Objekt statt, interagiert das Objekt mit sich selbst, um seine

---

[1] [Bullinger 1996: S. 95].

[2] Vgl. [Picot 1999: S. 345 ff.]. Vgl. dort auch die ausführliche Diskussion über die Grenzen dieses Konzepts, auf die hier nicht eingegangen wird.

[3] Vgl. [Schulte-Zurhausen 1999: S. 223].

[4] Vgl. [Picot 1996: S. 205].

[5] Vgl. [Hammer 1994: S. 74 f.], [Hammer 1993b: S. 81].

[6] Vgl. [Scholz 1994: S. 14].

[7] Vgl. [Schreyögg 1990: S. 72].

[8] Vgl. [Ferstl 1998: S. 43].

Leistungs- und Lenkungsaufgaben zu initiieren. Das folgende Skript beinhaltet ein einfaches Muster dieses Prinzips:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Kunde	sendet Bestellung :Action	:Interaction	:Bestellabwicklung Stpe(Department)	Bestellung abwickeln :Operation
	:Bestell-abwicklung Stpe(Department)	Auslieferung einplanen :Action	:Interaction Coord(control) Prop(Unicast, self)	:Bestellabwicklung Stpe(Department)	Bestellung ausliefern :Operation
	:Bestell-abwicklung Stpe(Department)	liefern :Action	:Interaction	:Kunde	erhält Bestellung :Operation
	:Bestell-abwicklung Stpe(Department)	Bearbeitungsergebnis dokumentieren :Action	:Interaction Coord(feedback) Prop(Unicast, self)	:Bestellabwicklung Stpe(Department)	Abwicklung dokumentieren :Operation

Skript 207: Beispiel eines vertikal komprimierten Prozesses

Dieses Beispiel basiert auf [Ferstl 1994c] und wurde hier entsprechend modifiziert. Zu erkennen ist, dass die Abteilung `Bestellabwicklung` (gekennzeichnet durch das Stereotyp `Department`) sowohl die Auslieferung einplant wie auch durchführt. Auch die abschließende Kontrollmeldung führt die Abteilung selbst aus. Durch das Prädikat `Prop(Unicast, self)` ist eindeutig festgelegt, dass das Objekt Bestellabwicklung nicht eventuell mit irgendeinem anderen Objekt der Klasse Bestellabwicklung interagiert, sondern mit sich selbst. Auch hier besteht ein Vorteil der OBA++ darin, dass ggf. andere Differenzierungen definiert werden können, indem man weitere Terme für den Aufzählungsdatentyp `Coord Identifier` (vgl. Kapitel 11) definiert.

7.4.7 Stellenbildung

Geschäftsprozesse sind unter anderem aufgrund ihrer Komplexität in der Regel arbeitsteilig organisiert. *Arbeitsteilung* bedeutet die Zuteilung von Aufgaben nach Art und Menge auf mehrere Personen. Man unterscheidet zwei Grundformen der Arbeitsteilung. *Mengenteilung* liegt vor, wenn die Aktionsträger aus einem umfangreichen Arbeitskomplex ein gleichartiges Arbeitspensum zur Erledigung erhalten. *Artenteilung* liegt vor, wenn Arbeitspensen unterschiedlicher Art auf spezialisierte Personen verteilt werden.[1]

[1] Vgl. [Schulte-Zurhausen 1999: S. 130 f.], [Corsten 1997b: S. 45].

Im Rahmen der Analyse von Geschäftsprozessen wird untersucht, ob eine funktions- oder prozessorientierte Stellenbildung vorliegt.[1] Bei der funktionsorientierten Stellenbildung werden vom Stelleninhaber gleichartige Verrichtungen ausgeführt. Dies entspricht einer Artenteilung. Bei der prozessorientierten Stellenbildung soll das Durchschneiden von Geschäftsprozessen reduziert werden: Stelleninhaber sollen möglichst umfangreiche Abschnitte von Geschäftsprozessen ausführen.[2] Dies entspricht einer Mengenteilung.[3]

Funktionsorientierte Stellenbildung wird in Skripten dadurch sichtbar, dass Aufgabenträger gleiche Verrichtungen an unterschiedlichen Geschäftsobjekten durchführen. Die Verrichtungen haben immer die gleichen Zustandveränderungen zu Folge. Es spielt dabei keine Rolle, ob dies in einem Skript oder in unterschiedlichen Skripten dokumentiert ist.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Schreibkraft Stpe(Position)	ausfüllen :Action	:Interaction	:Reisekostenantrag	wird ausgefüllt :Operation Post(State = ausgefüllt)
	:Schreibkraft Stpe(Position)	ausfüllen :Action	:Interaction	:Ausgabenbericht	wird ausgefüllt :Operation Post(State = ausgefüllt)
	:Schreibkraft Stpe(Position)	ausfüllen :Action	:Interaction	:Projektstatusbericht	wird ausgefüllt :Operation Post(State = ausgefüllt)
	:Schreibkraft Stpe(Position)	ausfüllen :Action	:Interaction	:Umsatzübersicht	wird ausgefüllt :Operation Post(State = ausgefüllt)

Skript 208: Funktionsorientierte Stellenbildung

Prozessorientierte Stellenbildung (dies entspricht der horizontalen Arbeitsintegration) lässt sich dagegen erkennen, wenn ein Aufgabenträger unterschiedliche Verrichtungen am gleichen Geschäftsobjekt ausführt.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Entwicker	analysiert :Action	:Interaction	:Reisekostenabrechnung	wird analysiert :Operation

[1] Vgl. [Schulte-Zurhausen 1999: S. 144]. Auf die Stellenbildung *ad rem* (aufgrund der technischen Gegebenheiten eines Sachmittels), *ad personam* (aufgrund der Fähigkeiten, Fertigkeiten und Interessen einer konkreten Person) und aufgrund rechtlicher Normen wird hier nicht eingegangen. Vgl. [Schulte-Zurhausen 1999: S. 148 f.], [Scholz 1994: S. 167]. Siehe hierzu auch KOSIOLS Darstellung der Zentralisation von Teilaufgaben hinsichtlich eines Merkmals. Vgl. [Kosiol 1962: S. 81 ff.].

[2] Vgl. [Hammer 1993b: S. 78]. Eine solche Integration von Teilbearbeitungsschritten zu einem abgeschlossenen Teilbearbeitungsprozess führt nach PICOT zu deutlichen Zeit- und Kosteneinsparungen. Vgl. [Picot 1999: S. 272]. HAMMER fordert sogar, ganze Geschäftsprozesse von sog. *Caseworkern* abwickeln zu lassen, komplexe Prozesse von sog. *Caseteams*. Vgl. [Hammer 1994: S. 73].

[3] Vgl. [Schulte-Zurhausen 1999: S. 145].

	Stpe(Position)				Post(State = analysiert)
	:Entwicker Stpe(Position)	entwirft :Action	:Interaction	:Reisekostenabrechnung	wird entworfen :Operation Post(State = entworfen)
	:Entwicker Stpe(Position)	implementiert :Action	:Interaction	:Reisekostenabrechnung	wird implementiert :Operation Post(State = implementiert)
	:Entwicker Stpe(Position)	testen :Action	:Interaction	:Reisekostenabrechnung	wird gestestet :Operation Post(State = getestet)

Skript 209: Prozessorientierte Stellenbildung

7.4.8 Strukturierung nach dem Prozessprinzip

Der Prozess ist nach NORDSIECK der Ausgangspunkt der Betrachtung einer Organisation.[1] Danach ist der „*Betrieb in Wirklichkeit ein fortwährender Prozess, eine ununterbrochene Leistungskette [...]. Die wirkliche Struktur des Betriebes ist die eines Stromes. Immerfort schafft und verteilt er im Durchlauf neue Produkte und Dienstleistungen auf Grund der gleichen oder nur wenig sich wandelnden Aufgaben [...]. Wie kann man angesichts solcher durchgängigen Vorstellungen die Aufgaben eines Betriebes prinzipiell anders gliedern als nach den natürlich-technischen Prozessabschnitten?*“[2] *„Das Prinzip der Aufgabengliederung ist demnach das der Prozessgliederung.“*[3] *“Nur eine Gliederung des Betriebes nach dem Prozessprinzip kann auf Dauer zu der Integration unserer Betriebe führen, ...“*[4]

Verwendet man das Prozessprinzip, entstehen zwischen den organisatorischen Einheiten Schnittstellen, an denen Geschäftsobjekte mit definierten Zuständen ausgetauscht werden: „*Typisch für einen Geschäftsprozess ist der Durchlauf durch organisatorische Einheiten (Abteilungen, Bereiche) eines Unternehmens. An den Grenzen zwischen den Einheiten treten im Prozess Schnittstellen auf.*“[5] Durch die Schnittstellen werden Prozesse in einzelne Phasen gegliedert: „*Bemerkenswert sind die Schnittstellen, welche den Betriebsprozess in Abschnitte zerlegen. Handelt es sich um eine echte Prozessgliederung, so sind diese Schnittstellen gekennzeichnet durch ein fortgeschrittenes Planziel. Am Ende des Abschnittes, der Phase, steht jeweils als Ergebnis ein*

[1] Vgl. [Nordsieck 1972: Sp. 105].

[2] [Nordsieck 1972: S. 9 ff.].

[3] [Nordsieck 1972: Sp. 12].

[4] [Nordsieck 1972: Sp. 12]. Dieser Gedanke der Dynamik findet sich auch bei WILD, der Unternehmen als *Aktionsgefüge* bezeichnet. Vgl. [Wild 1966].

[5] [Haist 1989: S. 94].

festes Konzept, ein spezifizierter Plan, oftmals auch ein Modell [...]. Alles dies ist als eine Art „Auftrag" (Motiv) Unterlage für die weiterführende Arbeit der nächsten Prozessphase."[1]

Die Strukturierung nach dem Prozessprinzip modelliert man in der OBA++, indem man die beteiligten organisatorischen Einheiten und ihre Aufgaben abbildet. Diese tauschen über `In()` und `Out()` Geschäftsobjekte aus. Im Rahmen ihrer Verrichtungen wird der Zustand des Geschäftsobjekts verändert. Aus dem Zustand des Geschäftsobjekts ist ersichtlich, in welchem Zustand sich der so genannte *Geschäftsvorfall* gerade befindet.[2] Das folgende Skript enthält ein Muster eines einfachen Prozesses, wobei die Zustandsveränderung des Geschäftsobjekts explizit modelliert wird. Die drei beteiligten Abteilungen übergeben den Geschäftsvorfall immer in einem definierten Zustand:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Kunde	...	:Interaction	:Auftragsannahme	annehmen Bestellung :Operation In(:Auftrag) Goal(Auftrag.State = erstellt)
1.1.	:Auftragsannahme	erstellen :Action	:Interaction	:Auftrag	anlegen :Operation Goal(State = erstellt)
1.1.1.	:Auftrag	Initial :State	:Statechange Act(anlegen :Operation)	:Auftrag	erstellt :State
2.	:Auftragsannahme	anfertigen lassen :Action	:Interaction	:Produktion	anfertigen :Operation In(:Auftrag) Goal(Auftrag.State = gefertigt)
2.1.	:Produktion	fertigen :Action	:Interaction	:Auftrag	fertigen :Operation Goal(State = gefertigt)
2.1.1.	:Auftrag	erstellt :State	:Statechange Act (fertigen :Operation)	:Auftrag	gefertigt :State
3.	:Produktion	ausliefern lassen :Action	:Interaction	:Versand	ausliefern :Operation In(:Auftrag) Goal(Auftrag.State = geliefert)
3.1.	:Versand	liefert :Action	:Interaction	:Kunde	erhalten :Operation In(:Auftrag) Goal(Autrag.State = geliefert)
3.1.1.	:Auftrag	gefertigt :State	:Statechange Act(erhalten :Operation)	:Auftrag	geliefert :State

Skript 210: Strukturierung nach dem Prozessprinzip

[1] [Nordsieck 1972: S. 11].

[2] Vgl. [Kueng 1995: S. 81, 85].

7.4.9 Objekt- und Verrichtungszerlegung

In der Methode SOM wird die Objekt- und die Verrichtungszerlegung unterschieden. *Verrichtungszerlegung* bedeutet, dass eine Verrichtung in mehrere Teilverrichtungen zerlegt wird. Zwischen den Verrichtungen werden zusätzliche Interaktionen – im SOM Transaktionen genannt – eingefügt. *Objektzerlegung* bedeutet, dass ein Objekt in mehrere Objekte zerlegt wird. Zwischen diesen Objekten werden ebenfalls wieder zusätzliche Interaktionen eingefügt.[1]

Tatsächlich stellen beide Zerlegungen Auflösungen von Abstraktionen dar. Bei der Objektzerlegung liegt der Fall vor, dass im Rahmen der Analyse ein Objekt einer höheren Prozessebene in mehrere Objekte einer niedrigeren Prozessebene zerlegt wird. Bei der Verrichtungszerlegung wird eine Operation in einzelne Teiloperationen zerlegt. Beide Fälle können – nach dem im letzten Kapitel entwickelten Verfahren – durch Subskripte modelliert werden. Eine Objektzerlegung findet sich in Skript 197, in welchem das Objekt `Hotel` aus Skript 196 zerlegt wird.

Durch die Verrichtungszerlegung kann man unter anderem auch die Situation modellieren, dass eine Ressource durch den Beginn einer Aktivität nicht mehr zur Verfügung steht und nach Abschluss der Aktivität wieder freigegeben wird. Dies soll an Skript 183 erläutert werden. Durch das Prädikat `Concurrency(sequential)` wird festgelegt, dass eine Schreibkraft die Operation zu einem Zeitpunkt immer nur einmal ausführen kann und die Aufforderungen, sie auszuführen, nicht puffert. Verwendet man eine Verrichtungszerlegung, kann durch die Nachbedingungen modelliert werden, dass die Schreibkraft durch die Erstellung des Kündigungsschreibens blockiert ist. Im folgenden Skript wird eine Verrichtungszerlegung der Interaktion aus Skript 183 vorgenommen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:AL	veranlasst Kündigungsschreiben an einen Kunden :Action	:Interaction	:Schreibkraft	beginnt Erstellung Kündigungsschreiben :Operation In(:Kundenname, :Grund) Vis(qualified) Exp(:AL, :Geschäftsführer) Concurrency(sequential) Pre(Schreibkraft ist verfügbar) Post(Schreibkraft ist nicht verfügbar)
2.	:Schreibkraft	schreibt Kündigungsschreiben :Action	:Interaction	:Kündigungsschreiben	wird erstellt :Operation In(:Kundenname, :Grund) Out(:Kündigungsschreiben) Post(Kündigungsschreiben ist erstellt)
3.	:Schreibkraft	stellt Kündigungsschreiben	:Interaction	:Schreibkraft	beendet Erstellung Kündigungsschreiben :Operation

[1] Vgl. [Ferstl 1995b].

		fertig :Action			Pre(Schreibkraft ist nicht verfügbar) Post(Schreibkraft ist verfügbar)
4.	:Schreibkraft	übergibt Kündigungsschreiben :Action	:Interaction	:AL	erhält Kündigungsschreiben :Operation In(:Kündigungsschreiben) Pre(True) Post(Kündigungsschreiben erhalten)

Skript 211: Beispiel für Verrichtungszerlegung

Durch Verrichtungszerlegung kann ein Problem des OA bei der GPM gelöst werden. Die Vorbedingung einer Operation dokumentiert einen Zustand vor der Ausführung der Operation. Die Nachbedingung einer Operation dokumentiert den Zustand nach Abschluss der Ausführung. Sowohl vor als auch nach der Erstellung des Kündigungsschreibens ist die Schreibkraft verfügbar. Dass sie während der Erstellung nicht verfügbar ist, kann durch Vor- und Nachbedingungen nicht ausgedrückt werden, da diese nicht den Zustand während der Ausführung dokumentieren. Das Problem, dass ein Objekt blockiert und wieder freigegeben wird, kann am besten durch dezidierte Beginn- und Abschlussoperationen modelliert werden, wie dies im letzten Skript der Fall ist.

7.4.10 Kundendifferenzierung

In der Literatur unterscheidet man primäre und sekundäre Kunden, indirekte und direkte Kunden.[1] *Primäre Kunden* oder *direkte Kunden* sind die Empfänger der eigentlichen Leistung eines Prozesses. *Sekundäre Kunden* erhalten andere Ergebnisse des Prozesses, die nicht zur eigentlichen Leistung gehören. Dies können Berichte an die Unternehmensleitung oder an ein Entsorgungsunternehmen gelieferte Wertstoffe sein. *Indirekte Kunden* sind interne Kunden. Sie erhalten nicht direkt eine Leistung, werden aber beeinflusst, wenn Leistungen zu spät oder falsch erfolgen. *Externe Kunden* sind außerhalb des Unternehmens und erhalten die fertiggestellte Leistung des Prozesses; sie sind direkte Kunden.

Rollenbezeichnungen werden in der OBA++ auch verwendet, um zwischen unterschiedlichen Kundenarten differenzieren zu können. Sekundäre und indirekte Kunden erhalten die Rollenbezeichnungen `Sekundärer Kunde` bzw. `Indirekter Kunde`. Primäre und direkte Kunden erhalten die Rolle `Kunde`. Dabei werden externe Kunden zusätzlich durch das Prädikat `System(external)` gekennzeichnet.

[1] Vgl. [Harrington 1991: S. 72 f.], [Engelmann 1995: S. 45], [Fischermanns 1997: S. 24].

7.4.11 Automatisierungsgrad von Aktivitäten und Kommunikation

Die Aktivitäten eines Geschäftsprozesses werden teilweise von Menschen, teilweise von CIS ausgeführt.[1] Durch die Modellierung des *Automatisierungsgrades* soll u. a. die fehlende Computerunterstützung automatisierbarer Prozesse bzw. Aktivitäten aufgedeckt werden.[2] Die Zielsetzung ist, dass „*typische Geschäftsvorfälle [...] automatisch erkannt und so weit wie möglich durch standardisierte Prozesselemente elektronisch bearbeitet werden.*“[3] Manuell durchgeführte, repetitive Aktivitäten sollen möglichst automatisiert werden.[4]

Der *Automatisierungsgrad* wird durch die Zuordnung von Aufgaben zu Aufgabenträgern erkennbar.[5] Dabei wird unterschieden zwischen der *vollständigen*, *teilweisen* oder *fehlenden* Automatisierung von Aufgaben und Interaktionen.[6] Vollständig bzw. vollautomatisiert bedeutet, dass eine Aktivität vollständig von einem Arbeitsträger ausgeführt wird. Fehlende Automatisierung bzw. nicht-automatisiert bedeutet, dass eine Aktivität vollständig von einer Person ausgeführt wird. Teilautomatisierung bzw. teil-automatisiert bedeutet, dass eine Aktivität kooperativ von einer Person und einem Arbeitsträger ausgeführt wird.[7] Bei der Teilautomatisierung unterscheidet man außerdem:[8]

1. Personale Vorgangsauslösung. Der Vorgang ist bis auf die Auslösung automatisiert.

---

[1] Vgl. [Ferstl 1998: S. 3, 5], [Oberweis 1997]. Eine möglichst umfassende Unterstützung der aufbau- und ablauforganisatorischen Strukturen durch adäquate Informations- und Kommunikationssysteme ist heute ein wichtiger Erfolgsfaktor für Unternehmen. Vgl. [Davenport 1993b: S. 37 ff.], [Huber 1997: S. 83]. Sie wird nicht nur als Automatisierung bestehender Prozesse verstanden, sondern als eine Möglichkeit, das Geschäft grundlegend zu verändern. Vgl. [Davenport 1993: S. 84], [Davenport 1993b]. Die Bedeutung betrieblicher Anwendungssysteme ergibt sich daraus, dass das eigentliche Wirtschaften im Betrieb durch Informationsprozesse stattfindet. Vgl. [Wild 1970: S. 50]. Auch im Transaktionskostenansatz wird auf die besondere Bedeutung der Informationstechnik zur Senkung der Transaktionskosten hingewiesen. Vgl. [Picot 1982: S. 272 f.]. Vgl. auch die Darstellung der Stellung der Informationswirtschaft im Industriebetrieb in [Picot 1991].

[2] Vgl. [Gehring 1998: KE 1, S. 69].

[3] Vgl. [Gaitanides 1994b: S. 4].

[4] Vgl. [Schulte-Zurhausen 1999: S. 105].

[5] Vgl. [Ferstl 1998: S. 47]. Vgl. Abschnitt 7.2.3.

[6] Vgl. [Ferstl 1998: S. 196].

[7] Vgl. [Ferstl 1998: S. 196].

[8] Vgl. [Ferstl 1998: S. 100 f.]. Eine ähnliche Differenzierung findet sich schon bei GUTENBERG. Vgl. [Gutenberg 1973: S: 93 ff.]. Siehe hierzu auch die Weiterentwicklung von SZYPERSKI in [Szyperski 1961: S. 110 ff.].

2. Automatisierung der Aktionen. Der Vorgang ist automatisiert, die Steuerung des Ablaufs erfolgt durch einen Menschen.
3. Automatisierung der Steuerung. Der Vorgang wird von Menschen ausgeführt, die Steuerung übernimmt ein Rechner.

Bei der Untersuchung des Automatisierungsgrades spielt auch die Automatisierung der Interaktion eine Rolle.[1] Eine Interaktion wird in Anlehnung an FERSTL und SINZ als automatisiert bezeichnet, wenn sie zwischen Menschen und Maschinen oder zwischen Maschinen stattfindet. Sie wird als nicht-automatisiert bezeichnet, wenn sie zwischen Menschen stattfindet.[2]

Aus dem Modell muss festzustellen sein, ob die Kommunikation zwischen Computern, Computern und Menschen oder zwischen Menschen stattfindet.[3] „*Mensch-Maschine-Systeme sind in der Regel leistungsfähiger als rein personelle oder rein maschinelle Aufgabenträger, da sie menschliche Kreativität und Assoziationsfähigkeit mit maschineller Verarbeitungskapazität und Verarbeitungsgeschwindigkeit vereinigen.*“[4]

Auf der einen Seite müssen im Rahmen der Prozessmodellierung die Abläufe unabhängig davon abzubilden sein, ob sie manuell oder unter Verwendung von Anwendungssystemen abgewickelt werden.[5] Auf der anderen Seite muss durch die GPM der erreichte bzw. erreichbare Automatisierungsgrad von Geschäftsprozessen erkennbar sein.[6]

Sowohl der Automatisierungsgrad von Aufgaben wie auch der von Interaktionen wird in der OBA++ durch die Verwendung von Stereotypen erkennbar. Die Automatisierung einer Aktivität modelliert man, in dem man das Objekt als Arbeitsträger stereotypisiert. Fehlende Automatisierung eines Prozesses wird durch das Stereotyp Aufgabenträger gekennzeichnet, welches im Prozessverlauf keinen Arbeitsträger verwendet. Teilautomatisierung erkennt man daran, dass ein Aufgabenträger im Prozessverlauf bei einigen Aktionen einen Arbeitsträger verwendet. In diesem Fall fordert entweder der Aufgabeträger als Initiator den Arbeitsträger auf, bestimmte Aktivitäten auszuführen, oder der Arbeitsträger fordert als Initiator den Aufgabenträger auf, bestimmte Eingaben vorzunehmen.

---

[1] Vgl. [Ferstl 1998: S. 196]. Siehe auch [Krüger 1994: S. 143] für die Differenzierung der technischen Unterstützung von I.u.K.-Prozessen, die hier nicht berücksichtigt wurde.

[2] Vgl. [Ferstl 1998: S. 196].

[3] Vgl. [Ferstl 1998: S. 3].

[4] [Ferstl 1998: S. 51]. Schon GUTENBERG argumentiert ähnlich. Vgl. [Gutenberg 1973: S. 19].

[5] Vgl. [Kueng 1995b], [Scholz 1994b: S. 39].

[6] Vgl. [Ferstl 1996: S. 57], [Scheer 1998].

Kann oder soll nicht entschieden werden, ob eine Aufgabe automatisiert oder nicht-automatisiert ausgeführt wird, bildet man das Objekt als Aktionsträger ab.[1] Bei der Verwendung des Stereotyps für Aktionseinheiten kann nicht entschieden werden, ob die Aufgabe teil-automatisiert ausgeführt wird, da es sich bei dem Sachmittel sowohl um einen Arbeitsträger als auch um ein Arbeitsmittel handeln könnte.

Die personale Vorgangsauslösung ist daran zu erkennen, dass der Initiator eine Person, eine Organisationseinheit oder ein Aufgabenträger ist. Alle übrigen Operationen werden von Arbeitsträgern ausgeführt. Ein Arbeitsträger löst die Operation des nächsten Arbeitsträgers aus. Automatisierte Aktionen erkennt man daran, dass die Operationen von Arbeitsträgern ausgeführt werden, aber eine Person, eine Organisationseinheit oder ein Aufgabenträger jeweils die Operation des nächsten Arbeitsträgers auslöst. Die automatisierte Steuerung ist daran zu erkennen, dass die Operationen von Aufgabenträgern, Personen oder Organisationseinheiten ausgeführt werden, der Initiator dieser Aufgaben aber ein Arbeitsträger ist.

Die automatisierte Interaktion ist in Skripten daran zu erkennen, dass mindestens der Initiator oder der Participant ein Arbeitsträger ist. Nicht-automatisierte Interaktion erkennt man daran, dass sowohl Initiator als auch Participant Personen, Aufgabenträger oder Organisationseinheiten sind.[2]

Soll ein Prozess unabhängig vom Automatisierungsgrad modelliert werden, verwendet man Organisationseinheiten oder Aktionsträger, die Geschäftsobjekte verändern.

7.4.12 Integrationsart

Nach KRÜGER sind für nahezu alle Funktionen eines Industriebetriebs in den letzten Jahren Softwaresysteme entwickelt worden.[3] Organisatorische Analysen haben sich bei der Entwicklung dieser Softwaresysteme jedoch in der Regel in Grenzen gehalten. Die dabei entstehenden CIS sind häufig nicht miteinander verbunden und technisch auch gar nicht verbindbar. Sie stellen sog. *Insellösungen* dar.[4] Im Ergebnis kommt es zu einer „*Elektrifizierung des Ist-Zustands*“.[5] Solche Integrationsdefizite zeigen sich häufig durch

- die mehrfache Erfassung gleicher Daten durch unterschiedliche Systeme,

---

[1] Vgl. Abschnitt 7.2.1.

[2] Dabei muss allerdings das Prädikat `Form()`, durch welches die physische Repräsentation der Interaktion abgebildet wird, beachtet werden.

[3] Vgl. [Krüger 1994: S. 148].

[4] [Krüger 1994: S. 148].

[5] [Krüger 1994: S. 148]. Dies wird auch als „*Elektrifizierung der Abläufe*“ bezeichnet. Vgl. [Gaitanides 1994b: S. 4].

- einen aufwändigen Austausch von Daten über Schnittstellen,
- die redundante Datenhaltung mit der Gefahr inkonsistenter Datenbestände und
- die Beschränkung der Anwendungssysteme auf einzelne betriebswirtschaftliche Funktionen und die mangelnde Unterstützung funktionsübergreifender Arbeitsabläufe.[1]

Daraus hat sich die Forderung nach der Verknüpfung der isolierten Systeme ergeben: *„ Um den Zugriff auf erforderliche Daten und Informationen und deren individuelle dezentrale Verarbeitung zu gewährleisten, müssen alle betrieblichen Informationssysteme durchgängig integriert und vernetzt sein.“*[2] Das Ziel dieser Integration ist die Verbindung von Daten und Arbeitsabläufen bzw. Prozessen.[3] Ein Aufgabenkomplex wird als maschinell integriert bezeichnet, wenn er von einem maschinellen Aufgabenträger (hier: Arbeitsträger) durchgeführt wird.[4] Bei der maschinellen Integration werden folgende Alternativen unterscheiden: [5]

- Die *Funktionsintegration* stellt die Zusammenfassung unterschiedlicher Aktivitäten in einem Anwendungssystem dar.
- Die *Datenintegration* stellt die Verwendung eines von mehreren Anwendungssystemen gemeinsam genutzten Datenbestands oder eines Datenaustauschformats dar, welches von mehreren Anwendungssystemen verwendet werden kann.
- Die *vertikale Integration* stellt die Verflechtung von Anwendungssystemen entlang einer Ebene des Leistungserstellungsprozesses dar.
- Die *horizontale Integration* stellt die Verknüpfung von Anwendungssystemen von der operativen bis zur strategischen Ebene dar.
- Die *Prozessintegration* stellt die durchgängige Unterstützung von Geschäftsprozessen durch ein oder mehrere Anwendungssysteme über die Grenzen betrieblicher Funktionalbereiche hinweg dar. Das bedeutet, dass nicht nur die Aufgaben einer bestimmten Organisationseinheit (in der Regel einer Abteilung) unterstützt werden, sondern die Aufgaben, die entlang eines Geschäftsprozesses (oder mindestens eines Teils eines Geschäftsprozesses) anfallen. Dabei ist es unerheblich, welche oder wie viele Organisationseinheiten involviert sind.

---

[1] Vgl. [Gehring 1998: KE 1, S. 40].

[2] Vgl. [Picot 1996: S. 247]. Ähnlich auch in [Österle 1995: S. 6]. Hier sind mit betrieblichen Informationssystemen die CIS gemeint.

[3] Vgl. [Gehring 1998: KE 1, S. 40].

[4] Vgl. [Ferstl 1998: S. 53].

[5] Vgl. [Gehring 1998: KE 1, S. 40 ff.]. Vgl. auch [Staud 2001: S. 14 f.].

Die Funktionsintegration ist der einfachste Fall der maschinellen Integration. Sie liegt vor, wenn ein Anwendungssystem – welches meistens als Arbeitsträger (`Stpe(Work Bearer)`) stereotypisiert ist – mehrere Operationen ausführt. Datenintegration liegt vor, wenn mehrere Arbeitsträger mit den gleichen Datenbeständen – die in den Skripten in Form von Geschäftsobjekten auftreten – interagieren oder Geschäftsobjekte direkt von einem Arbeitsträger über eine Kopplungsoperation zu einem anderen Arbeitsträger bewegt werden.[1]

Die vertikale Integration wird anhand eines kurzen Beispiels erläutert. Bei der vertikalen Integration durchläuft das Auftragsobjekt nacheinander die Abteilungen Auftragsbearbeitung, Konstruktion, Technik, Disposition und Finanzen. In den einzelnen Abteilungen werden unterschiedliche Anwendungssysteme verwendet.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Vertrieb Stpe(Department)	erstellen :Action	:Interaction	:Auftrag	wird angelegt :Operation In(: KundenNr, :AuftragsNr, :Komponente, :Kostenstelle, :Lieferdatum, :Preis)
:Vertrieb Stpe(Department)		verwendet :Association	Auftragsbearbeitung Stpe(Work Bearer)	
:Vertrieb Stpe(Department)	übertragen :Action	:Interaction	:Technik Stpe(Department)	erhält :Operation In(:Auftrag)
Technik Stpe(Department)		verwendet :Association	CAD-System Stpe(Work Bearer)	
:Technik Stpe(Department)	konstruieren :Action	:Interaction	:Auftrag	wird konstruiert :Operation In(:AuftragsNr, :Komponente; :Lieferdatum, :ZeichnungsNr)
:Technik Stpe(Department)	übertragen :Action	:Interaction	:Disposition Stpe(Department)	erhält :Operation In(:Auftrag)
:Disposition Stpe(Department)		verwendet :Association	Planungssoftware Stpe(Work Bearer)	
:Disposition Stpe(Department)	disponieren :Action	:Interaction	:Auftrag	wird disponiert :Operation In(:AuftragsNr, :Komponente, :Kostenstelle, :Liefertermin, :ZeichnungsNr, :FertAuftragsNr, :FertTermin)
:Disposition Stpe(Department)	übertragen :Action	:Interaction	:Finanzen Stpe(Department)	erhält :Operation In(:Auftrag)
:Finanzen Stpe(Department)		verwendet :Association	Auftragsbearbeitung Stpe(Work Bearer)	
:Finanzen Stpe(Department)	fakturieren :Action	:Interaction	:Auftrag	wird fakturiert :Operation In(:KundenNr, :AuftragsNr, :Lieferdatum, :Preis, :RechnDatum)

[1] Auf eine Gegenüberstellung dieser beiden Konzepte wird hier verzichtet. Sie ist bei [Gehring 1998: KE 1, S. 42] zu finden.

	:Finanzen Stpe(Department)	übertragen :Action	:Interaction	:Finanzen Stpe(Department)	erhält :Operation In(:Auftrag)
	:Finanzen Stpe(Department)		verwendet :Association	Debitorenbuchhaltung Stpe(Work Bearer)	
	:Finanzen Stpe(Department)	buchen :Action	:Interaction	:Auftrag	wird gebucht :Operation In(:KundenNr, :AuftragsNr, :Lieferdatum, :Preis, :RechnDatum)

Skript 212: Beispiel einer vertikalen Integration[1]

Die Modellierung der horizontalen Integration erfolgt ähnlich. Organisationseinheiten der operativen, mittleren und strategischen Ebene tauschen Daten aus und verwenden hierfür Anwendungssysteme. Von unten nach oben werden Daten verdichtet, von oben nach unten werden Daten aufgelöst. Ein Beispiel horizontaler Integration liegt beispielsweise vor, wenn die Entwicklungszeiten von Software-Ingenieuren in einem entsprechenden System erfasst werden.[2] Wöchentlich wird eine Zusammenfassung an die Projektleitung übermittelt, die für jedes Entwicklungsteam Summen bildet und eine Gesamtschau des Projekts erstellt. Schließlich werden jeden Monat von der Geschäftsführung die Gesamtdarstellungen aller Projektleiter zu einem Statusbericht des Unternehmens zusammengefügt. Die horizontale Integration ist also daran zu erkennen, dass die beteiligten Organisationseinheiten (zu finden in den Tabellenspalten Initiator und Participant) der operativen, mittleren und strategischen Ebene angehören und außerdem zwischen den von ihnen verwendeten CIS eine Datenintegration realisiert ist.

Die Prozessintegration ist eine Kombination der bisher genannten Integrationsprinzipien. Prozessorientierte Anwendungssysteme basieren sowohl auf der Daten- wie auf der Funktionsintegration. Sie unterstützen Abläufe sowohl in horizontaler wie auch in vertikaler Sicht. Dabei ist es unerheblich, ob es sich um ein oder mehrere Anwendungssysteme handelt. Wesentlich dagegen ist, dass die Anwendungssysteme die Abläufe auch über organisatorische Grenzen hinweg unterstützen.

Wenn interorganisatorische Daten- oder Prozessintegration[3] (vgl. Abschnitt 7.3.10) modelliert werden soll, ändern sich die vorgestellten Techniken nicht. Der einzige Unterschied besteht darin, dass externe Entitäten durch das Prädikat `System(external)` entsprechend gekennzeichnet werden.

[1] Das Beispiel ist formuliert in Anlehnung an [Gehring 1998: KE 1, S. 42].

[2] Zum Beispiel mit dem Produkt CLEARQUEST der Fa. Rational.

[3] Vgl. [Gehring 1998: KE 1, S. 47 ff.].

7.4.13 Schnittstellenanalyse

Die Aufgaben und Leistungen der am Geschäftsprozess Beteiligten werden durch Schnittstellen definiert.[1] Auch Anfang und Ende von Prozessen werden durch Schnittstellen definiert.[2] Durch die Fragestellung, wer eine Tätigkeit ausführt (vgl. Abschnitt 7.2.3), werden prozessuale und organisatorische Schnittstellen aufgefunden.[3] Gerade diese Schnittstellen erweisen sich häufig als die potenziellen Schwachstellen im Ablauf von Prozessen.[4] Eine Gemeinsamkeit moderner Organisationskonzepte ist daher die Vermeidung oder Reduzierung von Schnittstellen.[5] Die Prozessorganisation versteht sich somit zu einem erheblichen Teil als Schnittstellenorganisation.[6]

Nach SCHOLZ lassen sich vertikale und horizontale Schnittstellen unterscheiden. *Vertikale Schnittstellen* sind Schnittstellen zwischen aufeinander aufbauenden Stufen der Leistungserstellung. *Horizontale Schnittstellen* sind Beziehungen mit anderen Leistungsbereichen.[7] Außerdem können Schnittstellen *intraprozessual* (innerhalb des Prozesses) oder *interprozessual* (prozessübergreifend) sein. Dadurch entstehen theoretisch vier unterschiedliche Schnittstellenarten:

- Vertikal-interprozessuale Schnittstellen existieren zwischen zwei Prozessen, von denen einer eine Leistung für den anderen erbringt.
- Horizontal-interprozessuale Schnittstellen sind Schnittstellen zwischen zwei Prozessen, die unterschiedlichen Leistungsbereichen angehören und u. U. unterschiedliche Ziele verfolgen.
- Horizontal-intraprozessuale Schnittstellen sind die Schnittstellen zwischen Organisationseinheiten, die am gleichen Prozess beteiligt sind.
- Vertikal-intraprozessuale Schnittstellen sind die Schnittstellen zwischen den Elementen strategische Spitze, mittleres Linienmanagement, operativer Kern, Technostruktur und Serviceeinheiten nach MINTZBERG.[8]

Die Minimierung von Schnittstellen soll für die notwendige Teilautonomie von dezentralen Geschäftseinheiten sorgen. Diese zeichnen sich durch geschlossene Aufgabengebiete, flache

---

[1] Vgl. [Kleinsorge 1994], [Haist 1989: S. 94], [Schulte-Zurhausen 1999: S. 60].

[2] Vgl. [Striening 1988: S: 57].

[3] Vgl. [Scholz 1994b: S. 42].

[4] Vgl. [Striening 1988: S. 170], [Krüger 1994: S. 127], [Schulte-Zurhausen 1999: S. 204 f.].

[5] „*Was inhaltlich zusammengehört, sollte organisatorisch nicht getrennt sein.*“ [Huber 1997: S. 82]. Vgl. auch [Schulte-Zurhausen 1999: S. 46], [Rosemann 1996: S. 196], [Davenport 1993b: S. 8 f.].

[6] Vgl. [Fischermanns 1997: S. 127], [Krüger 1994: S. 40, 123].

[7] Vgl. [Scholz 1994: S. 78 ff.].

[8] Vgl. [Krüger 1994: S. 40 f.], [Bleicher 1991: S. 130 f.], [Scholz 1994: S. 56 f., 81].

Hierarchien, geringe Arbeitsteiligkeit und hohe Kundennähe aus.[1] Organisationseinheiten sollen möglichst so gebildet werden, dass sie vorrangig einen ganzen Prozess oder möglichst große Abschnitte davon abwickeln können.[2] Dadurch können Übergangszeiten, geistige Rüstzeiten und Informationsverluste vermieden werden.[3] Weitere Vorteile der Integration von Prozessen ist, dass emotionale und motivationale Barrieren vermieden werden.[4]

Eine Maßnahme zur Reduzierung von Schnittstellen ist nach SCHREYÖGG die Verwendung der Mengenteilung an Stelle der Artenteilung (Abschnitt 7.4.6). Dies reduziert den Koordinationsbedarf zwischen den Aufgabenträgern.[5] Ein Schnittstellenabbau lässt sich auch dadurch erreichen, dass Planungs-, Kontroll-, Entscheidungs- und Ausführungsaufgaben durch eine Organisationseinheit ausgeführt werden.[6]

Eine Schnittstellenanalyse hat zum Ziel, die unterschiedlichen Schnittstellen in einem Prozess und die sich gegebenenfalls daraus ergebenen Friktionen deutlich zu machen. Unabhängig davon, ob es sich um *vertikale* oder *horizontale Schnittstellen* oder um *intraprozessuale* oder *interprozessuale Schnittstellen* handelt, sind dabei die zu verwendenden Prinzipien der Modellierung identisch. Dies wird am folgenden Beispiel demonstriert:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Abt A Stpe(Department)	bearbeitet :Action	:Interaction	:Geschäftsobjekt	Bearbeitung erste Stufe :Operation
	:Abt B Stpe(Department)	bearbeitet :Action	:Interaction	:Geschäftsobjekt	Bearbeitung zweite Stufe :Operation

Skript 213: Implizit vorhandene organisatorische Schnittstelle

Aus diesem Beispiel lässt sich erkennen, dass zwei Abteilungen nacheinander einen Geschäftsobjekttyp bearbeiten. Damit ist implizit auch eine organisatorische Schnittstelle zwischen den Abteilungen vorhanden. Nach Abschnitt 7.2.10 werden solche Schnittstellen durch Verträge dokumentiert, in denen festgelegt ist, welche Anforderungen gelten sollen. Die hier vorliegende organisatorische Schnittstelle wird durch eine zusätzlich eingefügte Interaktion deutlich, in der die Übergabe des Geschäftsobjekts von der Abteilung A an die Abteilung B und die dafür geltenden Vor- und Nachbedingungen dokumentiert sind:

---

[1] Vgl. [Bullinger 1996: S. 92].

[2] Vgl. KUGELER und VIETING in [Becker 2000: S. 189].

[3] Vgl. [Eversheim 1995: S. 31].

[4] Vgl. [Krüger 1994: S. 127].

[5] Vgl. [Schreyögg 1990: S. 72].

[6] Vgl. [Fischermanns 1997: S. 227]. Vgl. auch Abschnitt 7.4.6.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Abt A	bearbeitet :Action	:Interaction	:Geschäfts-objekt	Bearbeitung erste Stufe :Operation
	:Abt A	übergeben :Action	:Interaction TimingConstraint (executionTime < 1 s) Concurrency(timeout) Suspend(1 d) Reply(Ack)	:Abt. B	Erhält Geschäftsobjekt :Operation In(:Geschäftsobjekt) Pre(Anforderungen an die Vorleistung von Abteilung A) Post(von B zugesichterte Leistung)
	:Abt B	bearbeitet :Action	:Interaction	:Geschäfts-objekt	Bearbeitung zweite Stufe :Operation

Skript 214: Explizit modellierte organisatorische Schnittstelle

Durch diese zusätzliche Interaktion ist die organisatorische Schnittstelle zwischen Abteilung A und B „offengelegt“ worden. Durch die Operation `Erhält Geschäftsobjekt()` kann der Vertrag zwischen A und B dokumentiert werden. Außerdem kann unter Verwendung der Prädikate `TimingConstraint()`, `Concurrency()`, `Suspend()` und `Reply()` die Interaktion zwischen den beiden Abteilungen präzise erfasst werden. Im hier formulierten Beispiel wurde (willkürlich) festgelegt, dass die Weitergabe des Geschäftsobjekts abgebrochen wird, wenn Abteilung B nicht innerhalb von einem Tag mit der Weiterbearbeitung beginnt. Schließlich soll die eigentliche Weitergabe nicht länger als eine Sekunde dauern. Abschließend wurde außerdem festgelegt, dass Abteilung A eine Benachrichtigung erhält, wenn Abteilung B mit der Weiterbearbeitung beginnt.

Wenn zwischen zwei parallel verlaufenden Prozessen interprozessuale Schnittstellen modelliert werden sollen, werden hierfür die Sequenznummern der Skriptzeilen mit einem Präfix versehen. Die Schnittstelle zwischen Prozessen ist daran zu erkennen, dass eine Interaktion mit beiden Prozessen synchronisiert ist. In dem folgenden (konstruierten) Beispiel ist das in der Interaktion mit der Sequenznummer `A1.&B3./5.` der Fall. In dieser Interaktion müssen die beteiligten Objekte der beiden parallel verlaufenden Prozesse interagieren.[1]

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
A1.	:Dreher Stpe(Position)	drehen :Action	:Interaction	:Werkstück 1	wird gedreht :Operation
B1.	:Fräser Stpe(Position)	zuschneiden :Action	:Interaction	:Werkstück 2	wird zugeschnitten :Operation
B2.	:Fräser Stpe(Position)	kanten :Action	:Interaction	:Werkstück 2	wird gekantet :Operation
B3.	:Fräser Stpe(Position)	entgraten :Action	:Interaction	:Werkstück 2	wird entgratet :Operation

[1] Im anderen Fall würde keine Schnittstelle existieren.

A2.	:Dreher Stpe(Position)	Gewinde schneiden :Action	:Interaction	:Werkstück 1	Gewinde wird geschnitten :Operation
A1.&B3. / 5.	:Dreher Stpe(Position)	übergibt Werkstück 1 :Action	:Interaction	:Fräser Stpe(Position)	nimmt entgegen :Operation In(:Werkstück 1)
5. / A4.	:Dreher Stpe(Position)	drehen :Action	:Interaction	:Werkstück 3	wird gedreht :Operation
5. / B5.	:Fräser Stpe(Position)	baut Werkstück 1 ein :Action	:Interaction	:Werkstück 2	einfügen :Operation In(:Werkstück 1)

Skript 215: Modellierung einer interprozessualen Schnittstelle

7.4.14 Kopplung

FERSTL und SINZ unterscheiden lose und eng gekoppelte Aufgaben.[1] Lose gekoppelte Aufgaben besitzen unterschiedliche Geschäftsobjekte und tauschen Informationen über das Geschäftsobjekt aus. Eng gekoppelte Aufgaben besitzen gemeinsame Geschäftsobjekte und tauschen diese aus.

Der Aspekt der Kopplung hängt eng mit der Schnittstellenproblematik des letzten Abschnitts zusammen. In Skript 214 liegt eine enge Kopplung der Aufgaben der beteiligten Abteilungen vor, da das zu bearbeitende Geschäftsobjekt direkt ausgetauscht wird. Liegt dagegen eine lose Kopplung vor, werden nur bestimmte Informationen über ein Geschäftsobjekt ausgetauscht. Im folgenden Beispiel speichert die Entwicklung die von ihr erstellten Programmquellen in einem *Repository*[2] ab und informiert die Testabteilung über diesen Umstand. Die Testabteilung entnimmt dem Repository die Programmquellen und testet sie:

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Entwicklung	erstellen :Action	:Interaction	:Programmquellen	erstellt werden :Operation
:Entwicklung	speichern :Action	:Interaction	:Repository	einchecken :Operation In(:Programmquellen) Out(:Beleg)
:Entwicklung	Testfreigrabe melden :Action	:Interaction	:Test	Freigabemeldung erhalten :Operation In(:Beleg)
:Test	entnimmt Testling :Action	:Interaction	:Repository	auschecken :Operation In (:Beleg) Out(:Programmquellen)
:Test	testen :Action	:Interaction	:Programmquellen	getestet werden :Operation

Skript 216: Beispiel loser gekoppelter Aufgaben

[1] Vgl. [Ferstl 1998: S. 91].

[2] Als *Repository* werden in der Softwareentwicklung spezielle Datenbanksysteme bezeichnet, die Entwicklungsartefakte verwalten.

7.4.15 Medienbrüche

Nach ZIMMERMANN ist es die Aufgabe einer Geschäftsprozessanalyse, insbesondere folgende organisatorische Schwachstellen zu erkennen:[1]

- Organisationsbrüche,
- Medienbrüche und unzureichende DV-Integration, wodurch Datenredundanzen und Doppelerfassungen von Informationen entstehen,
- die mangelnde automatische Abwicklung von Funktionen.[2]

Medienbrüche sind im Rahmen der Untersuchung des Informationsflusses von Interesse.[3] Sie führen zu Mehrfacherfassungen (Zeit- und Kostennachteile) und Redundanz- und Konsistenzproblemen (Qualitätsnachteile).[4]

Organisationsbrüche werden durch die explizite Modellierung von organisatorischen Schnittstellen, die mangelnde IT-Unterstützung wird durch die Modellierung des Automatisierungsgrades deutlich. Durch die explizite Modellierung organisatorischer Schnittstellen werden in der OBA++ aber auch Medienbrüche sichtbar, weil durch das Prädikat `Form(Form Identifier)` das Medium dokumentiert wird, welches bei einer Interaktion verwendet wird. Dadurch wird – selbst bei der Beachtung des Information Hidings – sichtbar, dass die Abteilung B der Abteilung C Lieferanweisungen als Papierformular übermittelt, obwohl sie selbst dieses Geschäftsobjekt zuvor als elektronisch übermittelte Aufgabe erhalten hat. Für Abteilung B entsteht damit die unnötige Aktivität, im Rahmen der Weiterbearbeitung der dritten Stufe die Lieferanweisung ggf. wieder elektronisch zu erfassen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Abt A	übergeben :Action	:Interaction Form(Outlook-Aufgabe)	:Abt. B	Weiterbearbeitung zweite Stufe :Operation In(:Lieferanweisung)
2.	:Abt B	übergeben :Action	:Interaction Form(Papierformular)	:Abt. C	Weiterbearbeitung dritte Stufe :Operation In(:Lieferanweisung)

Skript 217: Abbildung eines Medienbruchs über organisatorische Schnittstellen

[1] Vgl. [ZimmermannV 1999: S. 130].

[2] Vgl. auch M. ROSEMANN in [Becker 2000: S. 61].

[3] Vgl. [Rosemann 1996: S. 196], [Gehring 1998: KE 1, S. 69].

[4] Vgl. [Ferstl 1996: S. 58].

7.4.16 Fallmanagement

Eine der wirksamsten Maßnahmen zum Schnittstellenabbau ist die Zusammenfassung mehrerer Aufgaben oder eines ganzen Prozesses auf eine Stelle oder Organisationseinheit.[1] Als *Case*- bzw. *Fallmanagement* wird bezeichnet, wenn eine einzelne Stelle einen kompletten Geschäftsvorfall verantwortlich ausführt.[2] Der Kunde erhält für den kompletten Ablauf einen festen Ansprechpartner, der alle Informationen über die Abwicklung des Geschäftsvorfalls besitzt. Der Fallmanager führt aber tatsächlich überhaupt nicht alle Prozessaktivitäten aus, sondern verwendet die Dienste anderer Organisationseinheiten. Der Fallmanager verbirgt auf diese Weise gegenüber dem Kunden den tatsächlichen Projektverlauf, indem er den gesamten Ablauf von der Kundenanfrage über die Lieferung, Rechnungsstellung und Bezahlung abwickelt.[3] Solche *Casemanager* sind nach HAMMER und CHAMPY dafür zuständig, als Puffer zwischen komplexen Prozessen und Kunden zu fungieren. Sie verhalten sich so, als ob sie den gesamten Prozess abwickeln, auch wenn dies nicht der Fall ist.[4]

Fallmanagement wird in der OBA++ modelliert, indem alle Interaktionen eines Kunden mit der gleichen Organisationseinheit oder dem gleichen Aufgabenträger stattfinden.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Kunde	sendet Bestellung :Action	:Interaction	:Fallmanager	erhält Bestellung :Operation
:Fallmanager	beauftragen :Action	:Interaction	:Lager	Lieferung zusammenstellen :Operation
:Fallmanager	sendet Ware :Action	:Interaction	:Kunde	erhält Ware :Operation
:Fallmanager	Rechnungstellung veranlassen :Action	:Interaction	:Auftrags-bearbeitung	erstellt Rechnung :Operation
:Auftragsbe-arbeitung	sendet Rechnung :Action	:Interaction	:Fallmanager	erhält Rechnung :Operation
:Fallmanager	sendet Rechnung :Action	:Interaction	:Kunde	erhält Rechnung :Operation
:Kunde	bezahlt Rechnung :Action	:Interaction	:Fallmanager	erhält Bezahlung :Operation
:Fallmanager	informiert :Action	:Interaction	:Auftrags-bearbeitung	gleicht offenen Posten aus :Operation

Skript 218: Modellierung von Fallmanagement

[1] Vgl. [Fischermanns 1997: S. 224].

[2] Vgl. [Schulte-Zurhausen 1999: S. 295], [Striening 1988: S. 164], [Engelmann 1995: S. 83 ff.].

[3] Vgl. [Schulte-Zurhausen 1999: S. 295].

[4] Vgl. [Hammer 1994: S. 86], [Hammer 1993b: S. 16]. Eine kompatible Sichtweise wird auch im SOM verwendet, wenn FERSTL und SINZ auf abstrakter Ebene die gesamte Lenkung einer Leistung auf ein Objekt zusammenfassen. Vgl. [Ferstl 1996: S. 59]. Solche Kunden-Kontaktpersonen müssen nach KLEPZIG daraufhin untersucht werden, ob sie über ausreichende Kompetenz verfügen und alle notwendige Informationen besitzen. Vgl. [Klepzig 1997: S. 83].

7.4.17 Anwendungsfälle

Anwendungsfälle (*Use Cases*) sind in der ooSe das meistverwendete Instrument zur Dokumentation von Geschäftsprozessen.[1] Das Anwendungsfalldiagramm hat die Aufgabe, jene Abschnitte von Geschäftsprozessen zu benennen und zu strukturieren, die von einem (Software-)System unterstützt werden sollen. Das Anwendungsfalldiagramm beschränkt sich dabei auf die Außensicht des Systems.[2]

Die Akteure des Anwendungsfalldiagramms sind in der Notation der OBA++ die Initiatoren einer Interaktion.[3] Sie erhalten das Prädikat `System(external)`, wenn sie sich außerhalb des betrachteten Systems befinden. Ihre Absicht ist die Action. Das System, mit dem sie interagieren, ist der Participant, der Anwendungsfall ist dessen Operation. Die im Rahmen eines Anwendungsfalls stattfindenden Interaktionen zwischen Akteur und System (also die Beibehaltung der Blackbox-Sicht) werden durch ein Subskript abgebildet. Auch die Realisierung des Anwendungsfalls innerhalb des Systems durch eine Kollaboration mehrerer Objekte (also die Veränderung des Betrachtungsstandpunkts zu einer Whitebox-Sicht) wird durch ein Subskript abgebildet.[4]

In Anwendungsfalldiagrammen der UML werden drei Beziehungen zwischen Anwendungsfällen verwendet. Die Erweitert-Beziehung (UML-Stereotyp «extend») bedeutet, dass ein Anwendungsfall A optional – wenn also bestimmte Bedingungen gegeben sind – die Funktionalität des Anwendungsfalls B einbezieht. Die Benutzt-Beziehung (UML-Stereotyp «include») bedeutet, dass ein Anwendungsfall A immer die Funktionalität des Anwendungsfalls B einbezieht. Diese Beziehungen werden in den Skripten durch Interaktionen abgebildet, weil in beiden Fällen im Rahmen der Ausführung eines Anwendungsfalls das Verhalten eines anderen Anwendungsfalls einbezogen wird. Im Fall der Erweitert-Beziehung wird die Bedingung, unter der der zweite Anwendungsfall einbezogen wird, als Prädikat `Guard()` dokumentiert. Unterstützt ein Vertriebssystem neben dem Anwendungsfall `Verkaufen` auch den Anwendungsfall `Sonderanfertigung verkaufen`, würden folgende Interaktion entstehen:

---

[1] Vgl. [Balzert 1998], [Jacobson 1995], [Jacobson 2000], [OMG 2001: S. 4-12], [Graham 1998], [Linssen 1999b], [Schneider 1998], [Anderson 1995], [Henderson-Sellers 1998], [Leffingwell 2000: S. 49 ff.].

[2] Vgl. Abschnitt 2.4.3.

[3] Nur die externen Klassen, die auch als Initiator auftreten, sind Akteure. Klassen, die nur als Participant auftreten, sind keine Akteure, weil sie keine Ereignisse auslösen.

[4] Vgl. Abschnitt 6.12.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Anwender System(external)	verwendet System zum Verkauf :Action	:Interaction	:Vertriebssystem	verkaufen :Operation
:Vertriebssystem	verwendet :Action	:Interaction Guard(es wird kein Standardprodukt verkauft)	:Vertriebssystem	Sonderanfertigung verkaufen :Operation

Skript 219: Ein Anwendungsfall

Der dritte Fall, die Generalisierungsbeziehung zwischen Anwendungsfällen (in der UML als Vererbungsbeziehung dargestellt), wird dagegen nicht unterstützt, da ihre Semantik nicht klar definiert ist.[1] Im Bezug auf Anwendungsfälle unterscheidet sich die GPM von der Modellierung von Softwaresystemen dadurch, dass bei der letztgenannten die Anwender des Systems Akteure sind. Bei der GPM gehören die Anwender zum betrachteten System. Akteure sind in diesem Fall die Entitäten außerhalb des Systems, also zum Beispiel externe Kunden und Lieferanten.

7.4.18 Geschäftsregeln

In einer Reihe von Ansätzen werden sog. *Geschäftsregeln* (engl. *Business Rule*; abgekürzt als GR) in Form von ECA-Strukturen[2] verwendet, um Abhängigkeiten der Form *wenn ... dann* zu formulieren.[3] Geschäftsregeln bestehen aus einem Ereignis, einer Bedingung und einer Aktion.

Die OBA++ ist kein Ansatz zur Modellierung von Geschäftsregeln auf der Grundlage der ECA-Struktur. Trotzdem lassen sich mit Hilfe von Interaktionen auch Geschäftsregeln in Skriptform abbilden, wenn man einige Restriktionen beachtet, die sich aus der objektorientierten Sichtweise ergeben.

Das Ereignis der Geschäftsregel wird im Folgenden als Aktion abgebildet, die Aktion der Geschäftsregel wird als Operation und die Bedingung wird durch das Prädikat `Guard()` abgebildet. Allerdings muss sowohl das GR-Ereignis wie die GR-Aktion Objekten zugeordnet werden, um der Skriptform und der objektorientierten Sichtweise zu genügen. Außerdem ist die Einschränkung zu beachten, dass die GR-Bedingung immer vom Objekt evaluiert wird, welches das GR-Ereignis ausgelöst hat. Schließlich müssen zweiseitige Regeln in zwei separate Interaktionen zerlegt werden. Dies soll an folgendem Beispiel einer umgangssprachlich formulierten GR demonstriert werden:

---

[1] Vgl. [Henderson-Sellers 2000: S. 44, 46].

[2] ECA = *Event Condition Action*.

[3] Vgl. [Scheer 1998: S. 124], [Hoheisel 1999], [GUIDE 1997].

Wenn ein Kreditantrag eintrifft und seine Kreditsumme kleiner als 50.000 GE ist, wird er von einem Sachbearbeiter bearbeitet; im anderen Fall bearbeitet ihn ein Gruppenleiter.

Nach dem zuvor definierten Verfahren entstehen daraus die folgenden beiden Interaktionen:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Kreditantrag	eintreffen :Action	:Interaction Guard(Kreditsumme < 50.000 GE)	:Sach-bearbeiter	bearbeiten :Operation In(:Kreditantrag)
	:Kreditantrag	eintreffen :Action	:Interaction Guard(Kreditsumme ≥ 50.000 GE)	:Gruppen-leiter	bearbeiten :Operation In(:Kreditantrag)

Skript 220: Eine Geschäftsregel

7.4.19 Geschäftsobjekttransport

Im Rahmen der Prozessverbesserung sollen prozess- und störungsbedingte Transport- und Liegezeiten eliminiert werden.[1] Werden Aufgaben auf eine Stelle zusammengefasst, entfällt auch der Transport der Geschäftsobjekte zwischen den vorher getrennten Stellen, wodurch sich erfahrungsgemäß die Gesamtbearbeitungszeit reduziert.[2] Durch eine Veränderung der Reihenfolge von Aktivitäten kann nach HARRINGTON das Hin- und Herbewegen von Leistungen vermieden werden.[3]

Der Transport von Geschäftsobjekten wird in der OBA++ durch Input-Output-Beziehungen modelliert. Er ist aber nicht auf Geschäftsobjekte beschränkt, da alle Arten von Objekten auf diese Weise „bewegt“ werden:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Entwickler Stpe(Position)	erstellen :Action	:Interaction	:Komponente	entwickeln :Operation
2.	:Entwickler Stpe(Position)	übergeben :Action	:Interaction	:Projektleiter Stpe(Position)	sichten :Operation In (:Komponente)
2.1.	:Projektleiter Stpe(Position)	sichten :Action	:Interaction	:Komponenten	freigeben :Operation
3.	:Projektleiter Stpe(Position)	übergeben :Action	:Interaction	:Entwickler Stpe(Position)	weiterreichen :Operation In (:Komponente)
4.	:Entwickler Stpe(Position)	übergeben :Action	:Interaction	:Tester Stpe(Position)	testen :Operation In (:Komponente)

Skript 221: Beispiel zum Geschäftsobjekttransport

[1] [Schulte-Zurhausen 1999: S. 105]

[2] Vgl. [Fischermanns 1997: S. 226].

[3] Vgl. [Harrington 1991: S. 147].

Wie am Beispiel leicht zu sehen ist, wird das Geschäftsobjekt `Komponente` vom Entwickler an den Projektleiter und wieder zurück zum Entwickler bewegt, der sie schließlich an einen Tester übergibt. Der Entwickler führt aber keine Verrichtungen an der Komponente mehr durch.[1] Wenn der Projektleiter die Komponente direkt übergibt, fällt eine organisatorische Schnittstelle weg:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Entwickler Stpe(Position)	erstellen :Action	:Interaction	:Komponente	entwickeln :Operation
2.	:Entwickler Stpe(Position)	übergeben :Action	:Interaction	:Projektleiter Stpe(Position)	sichten :Operation In (:Komponente)
2.1.	:Projektleiter Stpe(Position)	sichten :Action	:Interaction	:Komponenten	freigeben :Operation
3.	:Projektleiter Stpe(Position)	übergeben :Action	:Interaction	:Tester Stpe(Position)	testen :Operation In (:Komponente)

Skript 222: Vereinfachter Ablauf mit Geschäftsobjekttransport

7.4.20 Iterationen

Unnötige Iterationsschleifen sollen im Verlauf von Prozessen vermieden werden.[2] Sie entstehen insbesondere durch das Nacharbeiten von Fehlern und die dadurch notwendigen korrigierenden Prozesse.[3] Jede Rückkopplung verursacht zusätzliche Transport- und Bearbeitungszeit. Eine Möglichkeit besteht darin, in den Prozess direkt fehlervermeidende Maßnahmen einzubauen, wodurch die Fehler und damit die Rückkopplungshäufigkeit zumindest tendenziell verringert werden können.[4]

Das Phänomen von Nacharbeiten soll an einem Beispiel aus einem typischen Geschäftsprozess, der Softwareentwicklung, erläutert werden. In der Softwareentwicklung wird üblicherweise eine funktionsorientierte Stellenbildung verwendet. Softwareentwickler erstellen ein Softwaresystem, welches von Softwaretestern geprüft wird. Ist das Prüfobjekt fehlerbehaftet, wird es an die Softwareentwickler zurückgegeben. Diese korrigieren das Prüfobjekt und übergeben das korrigierte Ergebnis wieder an die Softwaretester. Dieser Zyklus wiederholt sich so lange, bis durch die

[1] Die Modellierung der Zustandsveränderungen wurde hier weggelassen.

[2] Vgl. [Eversheim 1995: S. 31], [Striening 1988]. Siehe auch [Harrington 1991: S. 132], [Fischermanns 1997: S. 218].

[3] Nacharbeiten und Iterationen entstehen nach HAMMER und CHAMPY häufig dann, wenn in Abläufen kein ausreichendes Feedback vorhanden ist. [Hammer 1994: S. 163].

[4] Vgl. [Harrington 1991: S. 150 ff.], [Fischermanns 1997: S. 218].

Prüfung kein Fehler mehr gefunden wird. Anschließend wird das Softwaresystem an die Projektleitung übergeben:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
0. | 3. / 1.	:Entwickler Stpe(Position)	erstellen :Action	:Interaction	:Komponente	entwickeln :Operation
2.	:Entwickler Stpe(Position)	übergeben :Action	:Interaction	:Tester Stpe(Position)	testen Komponente :Operation In (:Komponente)
3.	:Tester Stpe(Position)	übergeben :Action	:Interaction Guard(Komponente fehlerhaft)	:Entwickler Stpe(Position)	korrigieren Komponente :Operation In (:Komponente)
4.	:Tester Stpe(Position)	übergeben :Action	:Interaction Guard(Komponente fehlerfrei)	:Projektleitung Stpe(Position)	abnehmen :Operation In (:Komponente)

Skript 223: Iterativer Prozess der Nachbearbeitung

Dieser Prozess enthält eine organisatorische Schnittstelle zwischen `Entwickler` und `Tester` (Interaktion 2.), die wiederholt durchlaufen werden muss. Wesentlich sinnvoller wäre es, die fehlervermeidenden Maßnahmen direkt in den Entwicklungsprozess einzubauen. In einem veränderten Geschäftsprozess würden die Softwaretester vor der Entwicklung die Prüfspezifikation an die Softwareentwickler übergeben. Die Softwaretester prüfen das Softwaresystem selbst und übergeben es erst, wenn es den Test bestanden hat. Die Softwaretester prüfen anschließend nicht mehr das Softwaresystem, sondern nur noch, ob der Softwaretest ordnungsgemäß stattgefunden hat:[1] Die Korrektur von Softwarekomponenten, die nicht fehlerfrei sind, wird ohne die Beteiligung der organisatorischen Schnittstelle vorgenommen.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
1.	:Tester Stpe(Position)	übergeben Testspezifikation :Action	:Interaction	:Entwickler Stpe(Position)	erhalten :Operation In(:Testspezifikation)
1. | 6. / 2.	:Entwickler Stpe(Position)	erstellen :Action	:Interaction	:Komponente	entwickeln :Operation
3.	:Entwickler Stpe(Position)	ermittle Tests :Action	:Interaction	:Testspezifikation	gib auszuführende Testfälle an :Operation
4.	:Entwickler Stpe(Position)	testen :Action	:Interaction	:Komponente	testen :Operation
5.	:Entwickler Stpe(Position)	dokumentieren Testergebnis :Action	:Interaction	:Testprotokoll	erstellen :Operation

[1] Dem Verfasser ist aus seiner praktischen Arbeit klar, dass hier ein idealisierter Ablauf beschrieben ist.

6.	:Entwickler Stpe(Position)	korrigieren :Action	:Interaction Guard(Komponente fehlerhaft)	:Entwickler Stpe(Position)	korrigieren Komponente :Operation In (:Komponente)
7.	:Entwickler Stpe(Position)	übergeben :Action	:Interaction Guard(Tests bestanden)	:Tester Stpe(Position)	prüfe Testprotokoll :Operation In (:Komponente, :Testprotokoll)
8.	:Tester Stpe(Position)	übergeben :Action	:Interaction	:Projektleitung Stpe(Position)	abnehmen :Operation In (:Komponente)

Skript 224: Veränderter Prozess ohne Iteration über die organisatorische Schnittstelle

7.4.21 Prioritätsregeln

Wenn mehrere Aufträge bearbeitet werden müssen und die Kapazität der Aktionseinheit eine gleichzeitige Bearbeitung nicht erlaubt, müssen sie nacheinander abgearbeitet werden. In diesem Fall werden Regelungen im Bezug auf die Reihenfolge der Bearbeitung verwendet. In der betrieblichen Praxis werden häufig Prioritätsregeln (Kürzeste-Operationszeit- (KOZ), First-Come-First-Served- (FCFS) oder Liefertermin-Regel) verwendet, um solche Engpässe überwinden zu können.[1] Auf diese Weise soll unter anderem sichergestellt werden, dass kritische Aktivitäten Vorrang vor weniger wichtigen Aktivitäten erhalten.[2]

In einer solchen Situation benötigt die Aktionseinheit einen Puffer (Warteschlange), in der die auszuführenden Aufträge eingefügt werden. Dieser Puffer verwaltet alle Aufträge, und die Aktionseinheit entnimmt diesem Puffer jeweils den nächsten anstehenden Auftrag, der von ihr verarbeitet wird. Die zu verwendende Prioritätsregel stellt eine Operation dar, deren Nachbedingung ist, dass der Auftrag mit der aktuell höchsten Priorität zurückgeliefert wird. Dabei kann die anzuwendende Prioritätsregel dem Puffer oder der Aktionseinheit zugeordnet werden.

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
:Auftragsvergabe	vergibt Auftrag :Action	:Interaction	:Auftragsstapel	push :Operation In(:Auftrag)
:Ausführender	holt nächsten Auftrag :Action	:Interaction	:Auftragsstapel	ermittle Auftrag nach KOZ-Regel :Operation

[1] Vgl. [Schulte-Zurhausen 1999: S. 118], [Krüger 1994: S. 132], [Mertens 1988: S. 158]. Nach der KOZ-Regel wird der Auftrag als nächster bearbeitet, der die geringste Durchführungszeit besitzt. Nach der FCFS-Regel wird der Auftrag, der zuerst eintrifft, auch zuerst behandelt. Nach der Liefertermin-Regel wird der Auftrag mit dem nächsten Liefertermin bearbeitet.

[2] Vgl. [Harrington 1991: S. 148 f.].

					Out(:Auftrag) Post(Auftrag mit KOZ zurückgegeben und im Auftragsstapel entfernt)
	:Ausführender	bearbeitet Auftrag :Action	:Interaction	:Auftrag	bearbeiten :Operation

Skript 225: Auftragsbearbeitung über Prioritäten

Das Modell kann insofern verfeinert werden, dass die Auftragsvergabe den Auftrag in den Puffer einfügt und dem Aktionsträger mitteilt, nach welcher Regel der nächste Auftrag entnommen werden soll:

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Auftragsvergabe	vergibt Auftrag :Action	:Interaction	:Auftragsstapel	push :Operation In(:Auftrag)
	Auftragsvergabe		:Membership DomCard(1) RngCard(n)	Auftrag	
	:Auftragsvergabe	teile aktuelle Regel mit :Action	:Interaction Content (KOZ :Regel)	:Ausführender	erhalte Regel für nächsten Auftrag :Operation In(:Regel)
	:Ausführender	holt nächsten Auftrag :Action	:Interaction Content (KOZ :Regel)	:Auftragsstapel	pull :Operation In(:Regel) Out(:Auftrag) Post(Auftragsstapel nach Regel durchsucht und Auftrag mit geringstem Wert zurückgegeben und im Auftragsstapel entfernt)
	:Ausführender	bearbeitet Auftrag :Action	:Interaction	:Auftrag	bearbeiten :Operation

Skript 226: Auftragsbearbeitung über Regel

7.4.22 Typ- und Exemplarebene

Die Modellierung von Abläufen muss sowohl auf der Typ- wie auf der Ausprägungsebene möglich und unterscheidbar sein.[1]

Wie schon in Abschnitt 6.1.1 demonstriert wurde, können Geschäftsprozesse auch auf der Ebene bestimmter Objekte dokumentiert werden. Dies betrifft auch den Inhalt der Interaktion, der durch das Prädikat `Content()` abgebildet wird. Versieht man beispielsweise Skript 190 mit Objektbezeichnern, erhält man:

[1] Vgl. [Oberweis 1996: S. 33], [Oberweis 1997], [Scheer 1998b: S. 26 ff.], [Schmidt 1997], [Kueng 1995: S. 81].

Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
Peter Müller /Kunde :Person Stpe(Individual) System(external)	bittet um Angebot :Action	:Interaction Content(01289 :Artikelnummer)	Jürgen Schmidt :Sachbearbeiter Stpe(Position)	liefere Angebot :Operation

Skript 227: Modellierung auf Exemplarebene

Allerdings ist dabei immer die grundsätzliche Restriktion des Objektansatzes zu beachten, dass ein Objekt immer zur Extension einer Klasse gehört. Daraus folgt, dass nie ein Objekt ohne Klassenbezeichner verwendet werden kann.

7.5 Zusammenfassung

Mit diesem Kapitel ist das Ziel der Arbeit, die Entwicklung eines objektorientierten Ansatzes zur GPM, erreicht. Es wurde gezeigt, dass die in den Kapiteln 3 (Die Architektur des Modellierungsansatzes) bis 6 (Die Anwendung des Skriptmodells) entwickelte OBA++ zur Modellierung von Geschäftsprozessen geeignet ist und die Phänomene abbilden kann, die in der referenzierten Literatur zu diesem Thema genannt werden. Von besonderer Bedeutung ist, dass die im Bereich der Geschäftsprozessorientierung wichtige Kunden-Lieferanten-Beziehung durch die objektorientierte Interaktionsbeziehung abgebildet wird. In Rahmen dieses Kapitels wurde aber auch deutlich gemacht, welche Unterschiede zwischen der Geschäftsprozessmodellierung (GPM), der objektorientierten Softwareentwicklung (ooSe) und der objektorientierten Geschäftsprozessmodellierung (ooGPM) mit der OBA++ zu beachten sind. Die wichtigsten Unterschiede werden hier noch einmal verkürzt zusammengefasst:

- *Grenze zwischen Diskurswelt und Umwelt:* Bei der ooSe werden nur Informationsobjekte modelliert. Schon die Akteure, welche die Ereignisse für ein CIS auslösen, sind nicht Bestandteil der Diskurswelt sondern der Umwelt. Bei der ooGPM ist dagegen der gesamte organisatorische Ablauf Inhalt des Modells.
- *Modellierung von Organisationseinheiten:* Bei der GPM werden die Organisationseinheiten nicht als Klassen abgebildet. Bei einer ooGPM werden auch Organisationseinheiten als Klassen abgebildet.
- *Häufigkeit von Singletons:* Je detaillierter man die Verantwortlichkeiten der Klassen modelliert, desto häufiger haben die Klassen eine Extension, die nur ein Element umfasst.
- *Zuordnung von Aktivitäten zu Klassen:* Bei der ooGPM müssen Aktivitäten immer Klassen zugeordnet werden.

- *Keine Aufgabenanalyse ohne Bildung von Zuordnungen:* Aktivitäten können bei der ooGPM niemals losgelöst von Objekten bzw. Klassen betrachtet werden. Eine Aufgabenanalyse ist nur möglich, wenn gleichzeitig personale, sachlogische, instrumentale oder informationelle Zuordnungen gebildet werden.
- *Unterschiedlicher Rollenbegriff:* Bei der GPM besitzen Rollen eine eigene Intension. Bei der ooGPM besitzen Rollen keine eigene Intension. Ist dies notwendig, werden sie als Klassen modelliert.
- *Flüsse:* Bei der ooSe werden nur Informationsflüsse abgebildet. Bei der ooGPM werden alle Arten von Flüssen (Information, Materie, Geld, Dienst- und Arbeitsleistung) modelliert.
- *Leistungen sind Output:* Bei der GPM sind Leistungen Ergebnisse von Prozessen. Bei der ooGPM sind Leistungen die Operationen, die ein Objekt ausführt.
- *Quelle und Senke:* Bei der GPM sind Prozesse EVA-Strukturen mit Quellen und Senken. Bei der ooGPM sind Prozesse keine EVA-Strukturen. Die Begriffe Quelle und Senke sind auf die ooGPM nur begrenzt übertragbar.
- *Schnittstelle:* Ein einzelnes Ein- oder Ausgabeargument einer Operation wird bei der GPM als Schnittstelle bezeichnet. Bei der ooGPM wird die Gesamtheit der öffentlichen Operationen einer Klasse als ihre Schnittstelle bezeichnet. Bei der GPM sind Schnittstellen außerdem in erster Linie die Schnittstellen von Organisationseinheiten. Bei der ooGPM haben dagegen alle Klassen Schnittstellen.
- *Kunden-Lieferanten-Beziehungen:* Bei der GPM wird davon ausgegangen, dass die Folgeaktivität der Kunde der vorhergehenden Aktivität ist (NOAC-Prinzip). Bei der ooGPM wird diese Metapher gedreht. Der Initiator fragt beim Participant eine Leistung nach, die ihm dieser liefert (Client/Supplier-Prinzip).
- *Verallgemeinerung der KL-Beziehung:* Bei der GPM bestehen KL-Beziehungen zwischen Organisationseinheiten. Bei der ooGPM wird diese Metapher auf alle Klassen erweitert.
- *Verallgemeinerung des Vertragsmodells:* Bei der GPM werden die KL-Beziehungen zwischen Organisationseinheiten durch Verträge dokumentiert. Bei der ooGPM wird dieser Ansatz auf alle Klassen erweitert.
- *Ereignisse:* Bei der GPM finden Ereignisse in der Umwelt statt. Bei der ooGPM treten auch im System Ereignisse auf. Bei der ooGPM gibt es keine zusammengesetzten Ereignisse. Jedes (Teil-)Ereignis muss durch eine separate Interaktion „transportiert“ werden.
- *Verwendung der gleichen Metamodellelemente auf allen Modellierungsebenen:* Bei der GPM wird zwischen Prozessen, Teilprozessen, Aufgaben, Funktionen und Aktivitäten unterschieden.

Bei der ooGPM werden auf allen Modellierungsebenen immer Aktionen und Operationen verwendet.

- *Zerlegung von Prozessen vs. Zerlegung von Objekten:* Der OA basiert auf der Zerlegung eines Systems in Objekte, während die GPM auf der Zerlegung von Funktionen bzw. Prozessen beruht.
- *Operative Prozessstruktur:* Der wesentliche Unterschied zwischen objektorientierten und nicht-objektorientierten Verfahren besteht wohl darin, was ein Prozess ist. Bei nicht-objektorientierten Verfahren dominiert die zustandsorientierte Sicht: Ein Prozess ist eine Abfolge von Zuständen und Zustandsübergängen. Bei objektorientierten Verfahren stellt ein Prozess eine Abfolge von Nachrichtenübertragungen dar. Die GPM basiert also auf der Betrachtung der Kontroll- und Steuerungsaspekte. Aus diesem Grund wird die operative Prozessstruktur in der GPM als Flussdiagramm abgebildet. Davon abweichend basiert die ooGPM auf der Betrachtung der Abläufe und des funktionalen Verhaltens durch Objektinteraktionen. Hierfür werden in der OBA++ Sequenzausdrücke verwendet.
- *Konzepte zur Modellierung von Ausnahmesituationen:* Im Gegensatz zur GPM existieren in der ooGPM mit der OBA++ Konstrukte zur Modellierung von Ausnahmen.
- *Geschäftsregeln:* Geschäftsregeln können bei der ooGPM modelliert werden, wenn das Ereignis von einem Objekt ausgelöst wird, die Bedingung von einem Objekt evaluiert wird und die Aktion von einem Objekt ausgeführt wird.

8 Resümee

„Eine einheitliche Konstruktionslehre für Prozesse hat sich noch nicht herausgebildet." [Hess 1996: S. 127]

In diesem Kapitel werden im Überblick die Ergebnisse und der innovative Gehalt der Arbeit zusammengefasst. Es werden Möglichkeiten zur Weiterentwicklung aufgezeigt, und es wird der in der Arbeit eingeschlagene Weg begründet. Die Arbeit schließt mit einem Ausblick.

8.1 Ergebnisse im Überblick

Das Thema der Arbeit war die Modellbildung betrieblicher Informationssysteme zum Zweck der Entwicklung computergestützter Informationssysteme. Dabei sollten insbesondere die in einer Organisation ablaufenden Prozesse untersucht werden.

In Kapitel 1 wurde dargestellt, warum ein Ansatz zur Modellierung von Geschäftsprozessen notwendig ist, der keinen Bruch zur anschließenden Softwareentwicklung beinhaltet. Die objektorientierten und strukturierten Ansätze aus dem SE wurden gegeneinander abgegrenzt, und es wurde begründet, warum ein objektorientierter Ansatz zu wählen war. Es wurde dargelegt, welcher Motivation der hier entwickelte Ansatz entsprungen ist und welches Ziel in dieser Arbeit erreicht werden sollte: Beabsichtigt war, einen semiformalen objektorientierten Ansatz zur Modellierung von Geschäftsprozessen zu entwickeln, in dem Prozesse schwerpunktmäßig als Abfolgen von Objektinteraktionen betrachtet werden,[1] der eine simultane Modellierung der dynamischen Sicht (Prozesse) und der statischen Sicht (Strukturen) ermöglicht und der keine graphische Darstellung benötigt. Einem interdisziplinären Ansatz folgend, sollte sowohl der aktuelle Stand der Forschung in der angewandten Organisationslehre (Geschäftsprozess-management) wie auch in der angewandten Informatik (objektorientierte Softwareentwicklung) berücksichtigt werden. Ausgehend von diesem Ziel wurden Unterziele und daraus resultierende Aufgaben hergeleitet. Da das Themengebiet Geschäftsprozesse sehr weitläufig ist, wurde detailliert

[1] Dies wurde in Abschnitt 1.4 als das *Primat der Interaktionsbeziehung* bezeichnet.

dargestellt, welche Aspekte dieses Themenkomplexes nicht behandelt werden. Das Kapitel wurde mit einem Überblick über den Aufbau der Arbeit abgeschlossen.

In Kapitel 2 wurde festgestellt, dass kein verwendbares Begriffssystem für die Metapher *Objektansatz* existiert. Ausgehend vom Modellbegriff wurden die Konzepte der Objektorientierung definiert, ohne dass auf Konzepte aus dem Bereich der Programmierung zurückgegriffen werden musste. Die UML als Standardnotation der objektorientierten Modellierung wurde vorgestellt. Anschließend wurden die unterschiedlichen Vorgehensweisen bei der objektorientierten Modellierung und bestehende objektorientierte Konzepte zur GPM untersucht. Es wurde festgestellt, dass der Schwerpunkt zur ooGPM ein Ansatz zur Modellierung von Abläufen und funktionalem Verhalten sein muss. Ein weiteres Ergebnis dieses Kapitels war, dass kein bestehender Ansatz zur ooGPM als ausreichend angesehen wurde.

In Kapitel 3 wurde die OBA von RUBIN und GOLDBERG als Grundlage des hier entwickelten Ansatzes vorgestellt, weil sie dem *Primat der Interaktionsbeziehung* entspricht. Die OBA basiert auf der Idee der Skripte, einer tabellarischen Form der Darstellung von Interaktionsbeziehungen. Außerdem wurde auf die Schwächen der OBA eingegangen, die bei einer Weiterentwicklung zu beseitigen waren. Der Erweiterung wurde der Name OBA++ gegeben. Die OBA++ basiert auf einer Drei-Ebenen-Schema-Architektur, um die softwaretechnische Realisierung, das Konzeptuelle Schema und das Externe Schema der Skripte getrennt betrachten zu können. In diesem Kapitel wurde außerdem exemplarisch der Abbildungsprozess über die drei Ebenen vorgestellt.

In Kapitel 4 wurde das Konzeptuelle Schema als Begriffssystem der Metapher Objektansatz entwickelt. Dieses Schema hat die Form eines erweiterbaren UML-Metamodells und basierte auf einem Semantischen Netz.

In Kapitel 5 wurde für das Externe Schema der Skripte ein weiteres Metamodell entwickelt. Darauf aufbauend wurde gezeigt, wie der Inhalt der Skripte auf das Konzeptuelle Schema abgebildet wird und wie man aus diesem Konzeptuellen Schema ein UML-Klassendiagramm erhält. Die Flexibilität des Modellierungsansatzes wurde dadurch belegt, dass exemplarisch andere Repräsentationsformen (SOM-Vorgangs-Ereignis-Schema, UML-Sequenzdiagramme, UML-Kollaborationsdiagramme, UML-Zustandsdiagramme[1], OBA-Skripte) in ein Konzeptuelles Schema umgesetzt wurden.

In Kapitel 6 wurden Skripte zur objektorientierten Modellierung verwendet. Ein Schwerpunkt war dabei insbesondere die Anwendung der unterschiedlichen Prädikate des Konzeptuellen Schemas. Ein weiterer Schwerpunkt war die Erweiterung der Modellierungsmöglichkeiten der

[1] In Abschnitt 6.3.

UML. Ein dritter Schwerpunkt war die Beseitigung wesentlicher Unzulänglichkeiten der OBA von RUBIN und GOLDBERG.

In Kapitel 7 wurden zunächst die unterschiedlichen Phänomene zusammengefasst, die bei der Modellierung von Geschäftsprozessen zu berücksichtigen sind. Dabei wurde das umfangreiche zur Verfügung stehende Quellenmaterial ausgewertet. Es wurde gezeigt, wie das jeweilige Phänomen mit den Skripten der OBA++ abgebildet werden kann. Hier wurden außerdem die Unterschiede erarbeitet, die bei einer ooGPM im Vergleich zur betriebswirtschaftlich orientierten GPM zu berücksichtigen sind.[1]

8.2 Verwendung und Erweiterung der UML

Die Unified Modeling Language ist für die OBA++ von zentraler Bedeutung, da die OBA++ die UML sowohl verwendet als auch deren Möglichkeiten erweitert. Die UML wird verwendet, weil das Metamodell der OBA++ in der UML formuliert ist. Wie schon erwähnt, wurden außerdem existierende Konstrukte der UML mit dem Ziel erweitert, die Möglichkeiten zur GPM zu verbessern. Dies beinhaltet insbesondere die folgenden Aspekte:

- *Sender und Empfänger:* Wie die meisten objektorientierten Ansätze betrachtet die UML beim Nachrichtenaustausch nur die Empfängerseite explizit. In der OBA++ wird die Senderseite in die Betrachtung aufgenommen.
- *Ereignisse:* Interaktionen werden als Ereignisse interpretiert, die durch die Aktion des Initiators verursacht werden. Ereignisse sind also immer Objekten zugeordnet. Alle Operationen werden durch Ereignisse ausgelöst. Zusätzlich wurden sich periodisch wiederholende Ereignisse eingeführt. Außerdem können nun konjunkte, adjunkte und disjunkte Ereignisse abgebildet werden.
- *Ziele:* Die Vorschläge von BOCK[2] berücksichtigend wurde eine Möglichkeit entwickelt, die Ziele von Aktivitäten zu dokumentieren.
- *Ausbreitung:* Im Gegensatz zur UML kann in der OBA++ zwischen Unicast-, Multicast- und Broadcast-Interaktionen unterschieden werden.
- *Kontinuität:* In der OBA++ kann zwischen diskreten und kontinuierlichen Interaktionen unterschieden werden.
- *Kosten und Zeiten:* In der UML fehlen Konstrukte, um Kosten von Aktivitäten abbilden zu können. Basierend auf dem Ansatz der Prozesskostenrechnung kann in der OBA++ einer

[1] Vgl. Abschnitt 7.5.

[2] Vgl. [Bock 2000], [Bock 2001].

Aktivität ein Kostenausdruck zugeordnet werden, der sich aus einem Aktivitätentreiber und Aktivitätenkostensätzen zusammensetzt. Die Ausdrucksmöglichkeiten für zeitliche Abhängigkeiten sind in der UML ebenfalls nur schwach ausgeprägt. Hier wurde ein umfangreiches Rahmenwerk entwickelt, mit der sich solche Abhängigkeiten abbilden lassen.

- *Zugriff auf Objektinterna:* Durch den Beziehungstyp der internen Beziehung kann abgebildet werden, wie die Attribute eines Objekts durch Aktionen verändert werden.
- *Sichtbarkeit:* Die Sichtbarkeit einer Operation lehnt sich in der UML stark an den Möglichkeiten objektorientierter Programmiersprachen (insbesondere an C++) an. Hier wurde eine Erweiterung entwickelt, die eine feingranulare Modellierung ermöglicht.
- *Beschreibung der Interaktion:* In der UML dienen Interaktionen immer dem Austausch von Informationen. Hier wurden Möglichkeiten geschaffen, zwischen unterschiedlichen Arten von Strömen und dem Inhalt des Stroms zu unterscheiden sowie das Medium der Interaktion festzuhalten.
- *Pseudointeraktionen in Vererbungsbeziehungen:* In der UML lassen sich die Pseudointeraktionen „aufwärts“ entlang einer Vererbungsbeziehung nicht abbilden. Dies wird in der OBA++ durch eine besondere Form der Selbstinteraktion abgebildet.
- *Stereotype für die GPM:* Für die Modellierung von Organisationen wurde eine Reihe von Stereotypen definiert. Auch in der UML sind Stereotype zur GPM vordefiniert. Diese werden hier jedoch nicht verwendet, da ihnen eine entsprechende betriebswirtschaftliche Fundierung fehlt. Stattdessen orientiert sich der Ansatz an den Begriffen der Betriebswirtschaftslehre und insbesondere an denen der Organisationslehre.
- *Vor- und Nachbedingungen:* In der UML werden in Interaktionsdiagrammen keine Vor- und Nachbedingungen verwendet. In der OBA++ sind sie ein Bestandteil der Spezifikation der Operation.
- *Synchronisation:* Die Möglichkeiten, unterschiedliche Synchronisationsformen abbilden zu können, wurden durch zeitabhängige, eingeschränkte und befristete Interaktionen erweitert. Außerdem wurden Benachrichtigungen und Rückrufe eingeführt.
- *Sequenznummern:* Die Sequenznummern der UML wurden erweitert, um komplexere Situationen modellieren zu können. Auf diese Weise lassen sich nun auch Schleifen ausdrücken. Außerdem kann man optionale Interaktionen und Interaktionen ohne festgelegte Reihenfolge darstellen.
- *Verhalten in Fehlersituationen:* Es wurde ein umfangreiches Konzept zur Modellierung von Fehler- und Ausnahmensituationen entwickelt. Sowohl Aktivitäten wie auch Interaktionen können durch spezielle Prädikate als „*außergewöhnlich*“ gekennzeichnet werden. Bei Operati-

onen kann angegeben werden, wann Fehler eintreten und was passiert, wenn ein Vertrag zwischen Initiator und Participant nicht eingehalten werden kann.

- *Semantik von referentiellen Beziehungen:* Die Semantik referentieller Beziehungen wurde – im Vergleich zur UML – präzisiert. Aktuelle Veröffentlichungen berücksichtigend wurden insbesondere die mereologischen Beziehungen (Aggregation und Komposition) präzisiert.

8.3 Begründung des eingeschlagenen Weges

Die UML stellt in zweifacher Weise das Fundament der OBA++ dar. Durch die Verwendung der UML war die Formulierung der Metamodelle möglich, ohne auch hierfür eine eigene Notation entwickeln zu müssen. Weil die UML die etablierte Modellierungssprache des Objektansatzes ist, stellt die OBA++ eine Erweiterung der UML, aber keinen Parallelentwurf dar. Wie schon in der Einleitung erwähnt wurde, erschien es schon aus ökonomischen Gesichtspunkten sinnlos, die Landschaft der Notationen und Methoden um eine weitere zu „bereichern".

Bei der Sichtung des Schriftguts zum Thema Objektorientierung wurde nach einiger Zeit sehr deutlich, dass *der* Objektansatz nicht existiert. Die Verwendung der Begriffe weicht bei den unterschiedlichen Autoren in mehr als nur in Nuancen voneinander ab. Auch die UML beinhaltete in diesem Punkt keine Hilfestellung, da sie allgemein gültig formuliert ist. Deshalb war es notwendig, zunächst einen Objektansatz zu entwickeln. Anschließend war die UML darzustellen, weil sie in der hier vorliegenden Arbeit als Repräsentationsform des OA diente.

Die Drei-Ebenen-Schema-Architektur (Externes, Konzeptuelles und Internes Schema) wurde aus mehreren Gründen verwendet. Das Begriffssystem des Modellierungsansatzes, welches im Konzeptuellen Schema dokumentiert ist, sollte von der externen Darstellung getrennt werden. Dadurch sollte die Möglichkeit eröffnet werden, auch andere Darstellungsformen einzusetzen, so weit sie mit dem Begriffssystem kompatibel sind. Außerdem wird das Konzeptuelle Schema verwendet, um aus den Skripten die im Rahmen der ooSe so wichtigen Klassendiagramme abzuleiten. Nur durch die Abbildung des Externen Schemas auf das feingranulare Niveau des Semantischen Netzes des Konzeptuellen Schemas ist es möglich, andere Sichten (z. B. Klassendiagramme) aus den Skripten herzuleiten. Durch die Trennung des Konzeptuellen und des Internen Schemas war es möglich, von den implementierungsspezifischen Details eines bestimmten Datenbanksystems oder einer Programmiersprache zu abstrahieren.

Die Entwicklung des Metamodells des Konzeptuellen Schemas erscheint im ersten Moment als unnötiger Aufwand, war aber für die Architektur des Modellierungsansatzes notwendig. Wären die Interaktionen der Skripte direkt auf Klassendiagramme abgebildet worden, wäre die Information verlorengegangen, welche Aktivitäten der Klassen bei einer Interaktion involviert sind. Dadurch

hätte keine referentielle Integrität zwischen den Skripten und den Klassendiagrammen bestanden. Es war aber gerade ein Ziel des hier entwickelten Modellierungsansatzes, eine Verbindung zwischen dynamischer Modellierung (Skripte) und statischer Modellierung (Klassendiagramme) zu besitzen. Das konzeptuelle Schema fungiert dabei als verbindendes Element zwischen den Skripten und den Klassendiagrammen.

Bei der Herleitung einer grundsätzlichen Sichtweise auf Geschäftsprozesse war zu berücksichtigen, dass Prozesse im OA Abfolgen von Interaktionen darstellen. Deshalb konnte ein Modellierungsansatz nur auf einer Art Interaktionsdiagramm beruhen. Aus diesem Grund wurden in der Arbeit keine UML-Aktivitätendiagramme verwendet, denn diese beruhen auf dem Kontroll- und Steuerungsaspekt. Die UML-Interaktionsdiagramme wurden nicht weiterentwickelt, weil die Vielzahl der verwendeten Modellprimitive nicht darzustellen war. Unter anderem war dies auch einer der Gründe, warum keine graphische, sondern eine tabellarische Notation verwendet wurde. Die Vielzahl der Phänomene, die in den Skripten erfasst werden sollten, hätte eine aufwändige Bildsprache erfordert, die schwer verständlich gewesen wäre und zu Akzeptanzproblemen in der praktischen Anwendung geführt hätte.

Da die OBA++ ein Ansatz zur objektorientierten Modellierung von Systemen und die Geschäftsprozessmodellierung ein Anwendungsgebiet der OBA++ ist, wurde die Anwendung der Skripte im Rahmen der objektorientierten Modellierung und ihr Einsatz zur Geschäftsprozessmodellierung in zwei Kapitel getrennt dargestellt.

8.4 Zusammenfassung

Die OBA++ bietet bei der objektorientierten Modellierung von Geschäftsprozessen eine Reihe von Vorteilen: Im Gegensatz zu anderen objektorientierten Ansätzen basiert die OBA++ auf einem definierten Begriffssystem, welches in Form eines Metamodells formuliert wurde. Beachtet man bei der Modellierung dieses Metamodell, werden eine Reihe von Modellierungsfehlern vermieden. Bei der Entwicklung des Metamodells wurde das Ziel verfolgt, eine möglichst große Anzahl von Sachverhalten durch vordefinierte Modellprimitive, also durch domänenunabhängige Knoten- und Kantentypen, sowie durch Prädikate abzudecken.[1] Der Modellierungsansatz ist der erste, der durch die Drei-Ebenen-Architektur aus dem Bereich der Datenbanksysteme die externe Repräsentation eines Modells vom zugehörigen Begriffssystem trennt. Zur Modellierung wird nur ein Formalismus verwendet: Mit den Skripten wird keine graphische Notation eingesetzt, sondern eine tabellarische Darstellung. Dies ermöglicht eine kompaktere Darstellung, als dies mit den üblichen Interaktions-

[1] Dies entspricht den Prinzipien eindeutig interpretierbarer Semantischer Netze. Vgl. [Reimer 1991: S. 96 – 99].

diagrammen möglich ist. Die Einfachheit der Darstellung reduziert außerdem Akzeptanzprobleme, wie sie häufig beim Einsatz komplexer graphischer Notationen entstehen. Den Prozess der fortlaufenden Erkenntnisgewinnung unterstützt der Modellierungsansatz in der Form, dass Modelle zunächst mit einem geringen Präzisionsgrad erstellt werden können und dann sukzessiv durch Formulierung zusätzlicher Prädikate präzisiert werden. Ein weiterer Vorteil ist, dass Strukturen und Verhalten (d. h. die Prozesse) simultan erfasst werden können. Dadurch entfällt die Notwendigkeit, bei der Modellierung von Prozessen parallel Klassendiagramme zu erstellen und zu pflegen. Die Prozesse werden konsequent als Abfolgen von Interaktionen aufgefasst,[1] was genau der objektorientierten Sichtweise entspricht. Der entwickelte Modellierungsansatz verwendet keine Variante der allgemein üblichen Ablaufdiagramme.[2] Dies unterscheidet ihn von allen übrigen Ansätzen.

Die Modellierung von Interaktionen ist in der UML angesichts ihrer Bedeutung im objektorientierten Ansatz erstaunlich schwach ausgebildet. Im Wesentlichen beschränkt man sich darauf, die im Rahmen der Interaktion übertragene Nachricht mit der aufgerufenen Operation des Empfängerobjekts gleichzusetzen. Hier wurden umfangreiche Erweiterungen entwickelt. Unter anderem wurde ein Rahmenwerk für die Formulierung von Zeiten und Fristen erarbeitet, welches leistungsfähiger als in anderen objektorientierten Methoden ist. Dabei betrachtet die OBA++ explizit nicht nur die Empfängerseite der Interaktion, sondern auch die Senderseite. Im Unterschied zu anderen objektorientierten Ansätzen wird auf diese Weise deutlich, welche Aktivität bzw. welches Ereignis des Initiators die Ursache für die Ausführung der Operation des Participants ist.

Die Modellierung betriebswirtschaftlicher Abläufe und die Modellierung von Abläufen in einem Softwaresystem geschieht in gleicher Weise. Die Tatsache, dass bei der Geschäftsprozessmodellierung konsequent die gleichen Metatypen wie bei der Softwareentwicklung verwendet werden, reduziert die sog. *semantische Lücke* zwischen betriebswirtschaftlicher und softwaretechnischer Sicht. Die Ergebnisse der Geschäftsprozessmodellierung können in UML-Diagramme umgesetzt werden, wobei insbesondere das Klassendiagramm von Bedeutung ist, da es den Dreh- und Angelpunkt der gesamten objektorientierten Softwareentwicklung darstellt. Der definierte Prozess der Abbildung des Externen Schemas auf das Konzeptuelle Schema sorgt für die Integrität der Modelle. Dies stellt einen erheblichen Vorteil gegenüber dem in der UML propagierten Verfahren dar, Abläufe und funktionales Verhalten durch Use Cases und Interaktionsdiagramme zu untersuchen. Dort fehlt ein solcher Integrationsmechanismus von Modellierungsergebnissen. In

[1] Durch diese Sichtweise, Geschäftsprozesse als Abfolgen von Objektinteraktionen anzusehen, unterscheidet sich die entwickelte Methode von anderen, die sich ebenfalls als objektorientiert bezeichnen, aber tatsächlich auf funktionalen Abstraktionen und funktionaler Zerlegung beruhen. Vgl. [Mertins 1995], [Mielke 2002].

[2] Vgl. die Darstellungen am Markt befindlicher Werkzeuge in [Bullinger 2001b].

diese objektorientierte Sichtweise wurden Modellprimitive integriert, die bisher in objektorientierten Ansätzen fehlten, aber bei der Modellierung von Geschäftsprozessen notwendig sind. Der Modellierungsansatz ist schließlich erweiterbar, was eine Anpassung an die jeweilige Projektsituation ermöglicht.

8.5 Ausblick

Vor dem Hintergrund der vorliegenden Ergebnisse sind noch andere Weiterentwicklungen denkbar, die hier kurz skizziert werden sollen:

Es wäre zu untersuchen, inwieweit sich der Inhalt der Skripte zu einer Simulation von Geschäftsprozessen eignet. Dabei müssten u. a. die taxonomischen und referentiellen Beziehungen außer Acht gelassen werden. Eine Idee wäre, Petri-Netze einzusetzen. Die Operationen würden als Transitionen betrachtet, deren Vor- und Nachbedingungen als Stellen. Sowohl der Kontrollstrang wie auch der Datenfluss könnte über gefärbte Marken realisiert werden.

Wenn die Skripte zur Modellierung von CIS verwendet werden, wäre die Abbildung auf eine verbreitete Programmiersprache wie Java oder C++ von großem Interesse. Hierbei wäre es insbesondere notwendig, die Vor- und Nachbedingungen der Operationen in ausführbaren Programmcode umzuwandeln. Eine andere Möglichkeit könnte darin bestehen, an Stelle einer herkömmlichen Programmiersprache hierfür die *Action Semantics* der UML[1] zu verwenden.

Bei der Darstellung der Zeit- und Veränderungsereignisse wurde die Möglichkeit erwähnt, durch die Prädikate des Metatyps *ACTION* nicht nur sich periodisch wiederholende Ereignisse, sondern auch stochastische Ereignisse zu dokumentieren. Bei der Modellierung von Geschäftsprozessen kann diese Information von Interesse sein, weil sich dadurch zum Beispiel festhalten ließe, wie häufig eine bestimmte Ausnahmesituation eintritt.

Zu untersuchen wäre auch die Integration von Konzepten der Nachrichtentheorie. Wenn man die Interaktion als Kanal zwischen Initiator und Participant betrachtet, kann man die obere Schranke (*Kapazität*) dieses Kanals abbilden, bei deren Überscheiten nach SHANNON ein Informationsverlust unvermeidlich ist. Außerdem kann man den durch Störungen eintretenden Informationsverlust (*Entropie der Äquivokation*) und die durch Störungen in die Botschaft eingeflossenen Zeichen (*Entropie der Irrelevanz*) abbilden.[2]

Die OBA++ verwendet nur die beiden Koordinationsprotokolle, die aus SOM übernommen wurden (hierarchische und nicht-hierarchische Koordination). Wie schon im Abschnitt über

[1] Vgl. [OMG 2001b], [OMG 2002].

[2] Vgl. [Steinbuch 1974: S: 92 f.], [Rechenberg 1999: S. 192, 198].

Koordinationsprotokolle erwähnt, könnte man weitere Koordinationsformen, wie sie in der Transaktionskostentheorie zu finden sind, integrieren.

Der Fokus der Betrachtung liegt (nicht nur bei der OBA++) bei der Geschäftsprozessmodellierung auf den auszuführenden Aktivitäten. Dies können u. a. Verrichtungen, Transformationen oder Aufgaben sein. Durch die unterschiedlichen Arten von Zuordnungen werden die Aufgaben auf Aufgabenträger verteilt oder die Verrichtungen Objekten zugeordnet. Ob die durch die personale Zuordnung stattfindende Stellenbildung beispielsweise dem *Kongruenzprinzip*[1] genügt, ist nicht nur hier nicht erkennbar, sondern auch in keinem anderem Modellierungsansatz, da entsprechende Modellprimitive zur Abbildung der Kompetenz und der Verantwortung fehlen. Dies ist nur ein Beispiel, dass noch weitere Phänomene aus dem Bereich der Organisationslehre in die OBA++ integriert werden könnten, die auch im Bereich der GPM bisher nicht betrachtet werden.

8.6 Schluss

Die heutige Softwareentwicklung beruht auf dem objektorientierten Ansatz. Für die Softwareentwicklung benötigt man einen passenden – also objektorientierten – Ansatz zur Modellierung von Geschäftsprozessen. Geschäftprozesse können im objektorientierten Ansatz sinnvollerweise nur Abfolgen von Interaktionen sein. Aus diesem Grund stellt die hier entwickelte OBA++ Prozesse konsequent als Abfolgen von Objektinteraktionen dar.

Ein Modellierungsansatz basiert auf einer Metapher. Die zu verwendende Metapher war hier der Objektansatz. Problematisch war dabei allerdings, dass jeder Autor einen eigenen Objektansatz verwendet. Demzufolge gab es keinen allgemein akzeptierten Objektansatz, der nicht auf Programmiersprachenkonzepten beruht und insbesondere für die Geschäftsprozessmodellierung verwendbar wäre. Deshalb wurde für die OBA++ ein Objektansatz entwickelt, der im Rahmen der Geschäftsprozessmodellierung auch für Nicht-Programmierer verständlich ist.

Ein Modellierungsansatz benötigt außerdem ein Begriffssystem, welches in Form eines Metamodells formuliert sein sollte. Dieses Begriffsystem wurde hier als Konzeptuelles Modell bezeichnet. Im Unterschied zu anderen Modellierungsansätzen wird darüber hinaus zwischen dem Externen Schema (der Repräsentation), dem Konzeptuellen Schema (dem eigentlichen Begriffssystem) und dem Internen Schema (der softwaretechnischen Realisierung) unterschieden.

[1] Das *Kongruenzprinzip* bedeutet, dass Aufgaben, Kompetenzen und Verantwortung kongruent sein müssen. Die Einheit des Aufgabengebiets wird durch die sachgerechte Zuordnung von Rechten und Pflichten des Aufgabenträgers gewährleistet. [Bleicher 1993: S. 117].

Für das Externe und für das Konzeptuelle Schema wurden Metamodelle und für die Abbildung zwischen den Schemata wurden Transformationsregeln entwickelt.

Die UML ist der allgemein akzeptierte Standard zur objektorientierten Modellierung. Deshalb war die UML bei der Entwicklung des Modellierungsansatzes zu verwenden. Hierfür wurde sie vorgestellt und anschließend für die Erstellung der Metamodelle eingesetzt.

Es gibt schon andere Ansätze zur Geschäftsprozessmodellierung mit der UML. Diese basieren allerdings nicht auf Objektinteraktionen. Da die UML der allgemein akzeptierte Ansatz zur objektorientierten Modellierung ist, musste ein neuer Ansatz auf Objektinteraktionen beruhen und zusätzlich kompatibel zur UML sein. Die hier entwickelte OBA++ ist kompatibel zur UML, vermeidet aber die Probleme bisheriger Ansätze zur Geschäftsprozessmodellierung, die auf der UML beruhen.

In der Softwareentwicklung sind grafische Modelle üblich. Diese Modelle sind für Nicht-Softwareentwickler zum Teil nur schwer verständlich. Die Object Behavior Analysis (OBA) ist ein objektorientierter Ansatz, der auf Interaktionen beruht und eine einfache tabellarische Darstellungsweise (Skripte) verwendet. Die OBA ist jedoch unvollständig dokumentiert und hat eine Reihe von Schwächen. Im Gegensatz zu fast allen Ansätzen verwendet auch die OBA++ keine graphische Notation. Statt dessen wurde die tabellarische Form der Skripte übernommen und umfassend erweitert. Im Zuge dessen wurden wesentliche Schwächen der OBA beseitigt.

In der objektorientierten Modellierung wird zwischen der statischen Sicht (der Struktur eines Systems) und der dynamischen Sicht (der Prozesse) unterschieden. Hierfür werden unterschiedliche Diagrammtypen verwendet. Der Modellierungsansatz sollte sowohl die statische wie auch die dynamische Sicht beinhalten, damit in den frühen Phasen der Analyse nicht mehrere Diagramme erstellt, gepflegt und miteinander abgeglichen werden müssen. Im Gegensatz zu anderen Ansätzen beinhalten die Skripte der OBA++ sowohl die statische wie auch die dynamische Sicht.

Klassenmodelle sind von besonderer Bedeutung für die objektorientierte Softwareentwicklung. Aus den Modellen der Prozesse sollten Klassenmodelle hergeleitet werden können. Aus den Skripten der OBA++ ist dies möglich. Als verbindendes Glied zwischen Skript und Klassenmodell wird dabei das Semantische Netz des Konzeptuellen Schemas verwendet.

Wenn der Modellierungsansatz kompatibel zur UML sein soll, müssen die anderen Diagrammtypen der UML in den entwickelten Modellierungsansatz „übersetzt“ werden können. Es war also zu zeigen, wie man andere Diagrammtypen in Skripte oder in das Semantische Netz übersetzt. Dies wurde an mehreren Beispielen demonstriert.

Die UML beinhaltet Metatypen zur Modellierung von Geschäftsprozessen. Aber diesen Metatypen fehlt die betriebswirtschaftliche Fundierung. Die in der OBA++ zur GPM verwendeten Metatypen sind dagegen betriebswirtschaftlich fundiert.

Die Möglichkeiten der UML, Prozesse durch Interaktionen zu modellieren, sind beschränkt und waren zu vergrößern. Dafür wurden in der OBA++ umfangreiche Erweiterungen geschaffen.

Mit den bisherigen objektorientierten Ansätzen lassen sich nur schlecht zeitliche Restriktionen formulieren. Die OBA++ bietet wesentlich umfangreichere Möglichkeiten, da ein eigenes Rahmenwerk für die Formulierung zeitlicher Restriktionen entwickelt wurde.

Es gibt eine Reihe von Phänomenen, die mit einem objektorientierten Ansatz zur GPM modelliert werden können müssen. Typische Phänomene dieser Art waren zusammenzustellen. Hierfür wurde das Schrifttum gesichtet, die typischen Phänomene der GPM zusammengestellt und mit der OBA++ modelliert.

Zwischen der betriebswirtschaftlichen Sicht und der objektorientierten Sicht bestehen eine Reihe von Unterschieden, auf die weder im gesichteten Schrifttum zur objektorientierten Modellierung noch in der Literatur zur Geschäftsprozessmodellierung eingegangen wird. Diese Unterschiede zwischen betriebswirtschaftlicher und objektorientierter Sicht sind hier erstmalig herausgearbeitet worden.

Mit der OBA++ steht nun ein umfassender und betriebswirtschaftlich fundierter objektorientierter Ansatz zur Modellierung von Geschäftsprozessen zur Verfügung, der kompatibel zum Industriestandard der UML ist.

9 Anhang 1: Beispielskript

Das folgende Skript zeigt ein einfaches Produktionsunternehmen (in Kapitel 7 als Komponentenhersteller bezeichnet) mit einstufiger Auftragsfertigung. Lieferant des Produktionsunternehmens ist ein Hersteller von (Elektronik-)Bauteilen, Kunden des Unternehmens sind Produzenten von HiFi-Geräten. Das Unternehmen ist auf dem Zerlegungsgrad der Mesoebene dargestellt. Nicht dokumentiert sind die Aspekte Ausnahmen, Nachbedingungen, zeitliche Restriktionen, Kosten und die Verrichtungen an den Geschäftsobjekten.

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	/Kunde :HiFi-Produzent Stpe(Organization) System(external)	bestellt Ware :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Verkauf Stpe(Org-Unit)	erzeugt Auftrag :Operation Pre(Kundendaten enthalten; Artikel gehören zum Lieferumfang) Goal(Auftrag mit Kunden- und Artikeldaten existiert) In(:Bestellung)
	:Verkauf Stpe(Org-Unit)	überträgt Auftrag :Action	:Interaction Flow(Information) Coord(control) Concurrency(asynchronous)	:Produktions-Steuerung Stpe(Org-Unit)	Auftragsplanung durchführen :Operation Pre(Auftrag mit Kunden- und Artikeldaten vorhanden) Goal(Bauteilbedarf mit notwendigen Bauteilen existiert; Fertigungsauftrag ist erstellt) In(:Auftrag)
	:Produktions-Steuerung Stpe(Org-Unit)	meldet Bauteilbedarf :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Bauteillager Stpe(Org-Unit)	prüft Materialverfügbarkeit :Operation Pre(Bedarfsmeldung enthält mindestens ein Bauteil) Goal(Fehlende Bauteile sind bekannt) In(:Bedarfsmeldung)

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
	:Bauteillager Stpe(Org-Unit)	bestellt fehlende Teile :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Einkauf Stpe(Org-Unit)	erhält Bestellung :Operation Pre(Es existiert eine nicht-leere Bedarfsmeldung) Goal(Bestellung an Lieferant mit fehlenden Bauteilen ist vorhanden) In(:Bestellbedarf)
	/Kunde :Einkauf Stpe(Org-Unit)	bestellt Bauteile :Action	:Interaction Flow(Information) Concurrency(asynchronous)	/Lieferant :Bauteil-Produzent Stpe(Organization) System(external)	erhält Bestellung :Operation Pre(Kunde ist nicht gesperrt; Artikel gehören zum Lieferumfang) Goal(Bestellte Bauteile werden geliefert) In(:Bestellung)
	/Lieferant :Bauteil-Produzent Stpe(Organization) System(external)	liefert Bauteile :Action	:Interaction Flow(Material) Concurrency(asynchronous)	:Bauteillager Stpe(Org-Unit)	nimmt Bauteile entgegen :Operation Pre(Lieferung unversehrt) Goal(Lieferung ist eingetroffen) In(:Lieferung)
	/Lieferant :Bauteil-Produzent Stpe(Organization) System(external)	versendet Lieferschein :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Bauteillager Stpe(Org-Unit)	vergleichen Lieferung mit Lieferschein :Operation Pre(Lieferschein ist vorhanden; Lieferung entspricht Lieferschein) Goal(Lieferung ist angenommen; Wareneingangsmeldung ist erstellt) In(:Lieferschein)
	:Bauteillager Stpe(Org-Unit)	meldet Wareneingang :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Einkauf Stpe(Org-Unit)	erhält Wareneingangsmeldung :Operation Pre(True) Goal(Wareneingangsmeldung ist vorhanden) In(:Wareneingangsmeldung)
	/Lieferant :Bauteil-Produzent Stpe(Organization) System(external)	versendet Rechnung :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Einkauf Stpe(Org-Unit)	prüft Rechnung :Operation Pre(Rechnung kann Lieferschein und Bestellung zugeordnet werden; Rechnung stimmt mit Bestellung und Wareneingangsmeldung überein) Goal(Rechnung ist akzeptiert) In(:Rechnung)
	:Einkauf Stpe(Org-Unit)	reicht Rechnung weiter :Action	:Interaction Flow(Information)	:Buchhaltung Stpe(Org-Unit)	OP Kreditoren erzeugen :Operation Pre(Rechnung ist von Einkauf akzeptiert)

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
			Concurrency(asynchronous)		Goal(Offener Posten Kreditoren ist erzeugt) In(:Rechnung)
	:Buchhaltung Stpe(Org-Unit)	bezahlt Rechnung :Interaction	:Interaction Flow(Money) Concurrency(asynchronous)	:Bank Stpe(Organization) System(external)	überweist Rechnung an Empfänger :Operation Pre(Empfänger bekannt; Betrag gedeckt) Goal(Zahlung ist an Empfänger überwiesen) In(:Zahlung)
	:Bank Stpe(Organization) System(external)	meldet Zahlungsausgang :Interaction	:Interaction Flow(Information) Concurrency(asynchronous)	:Buchhaltung Stpe(Org-Unit)	gleicht OP Kreditoren aus :Operation Pre(Zahlungsbeleg kann offenem Posten Kreditoren zugeordnet werden; Betrag ist korrekt) Goal(Offener Posten Kreditoren ist ausgeglichen) In(:Zahlungsbeleg)
	:Buchhaltung Stpe(Org-Unit)	versendet Zahlungsbeleg :Interaction	:Interaction Flow(Information) Concurrency(asynchronous)	/Lieferant :Bauteil-Produzent Stpe(Organization) System(external)	gleicht OP Debitoren aus :Operation Pre(Zahlungsbeleg kann offenem Posten Debitoren zugeordnet werden; Betrag ist korrekt) Goal(Offener Posten Debitoren ist ausgeglichen) In(:Zahlungsbeleg)
	:Produktions-Steuerung Stpe(Org-Unit)	erteilt Fertigungsauftrag :Action when(Fertigungszeitpunkt ist eingetroffen)	:Interaction Flow(Information) Coord(control) Concurrency(asynchronous)	:Fertigung Stpe(Org-Unit)	fertigt Produkt :Operation Pre(Fertigung hat freie Kapazitäten) Goal(Fertigungsauftrag ist akzeptiert; Lagerabruf ist erstellt) In(:Fertigungsauftrag)
	:Fertigung Stpe(Org-Unit)	ruft Bauteile ab :Action	:Interaction Flow(Information) Concurrency(asynchronous) Content(:Lagerabruf)	:Bauteillager Stpe(Org-Unit)	Bauteile zur Fertigung liefern :Operation Pre(Bauteile sind in ausreichender Menge vorhanden) Goal(Bauteile werden geliefert) In(:Lagerabruf)
	:Bauteillager Stpe(Org-Unit)	liefert Bauteile :Action	:Interaction Flow(Material) Concurrency(synchronous) Content(:Bauteile)	:Fertigung Stpe(Org-Unit)	fertigt Produkt :Operation Pre(Gelieferte Bauteile entsprechen Lagerabruf) Goal(Produkt ist gefertigt) In(:Bauteile)
	:Fertigung Stpe(Org-Unit)	liefert Produkt :Action	:Interaction Flow(Material)	:Auslieferungslager Stpe(Org-Unit)	Produkt einlagern :Operation Pre(Produkt ist unbeschädigt)

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
			Concurrency(synchronous)		Goal(Produkt ist im Lager) In(:Produkt)
	:Fertigung Stpe(Org-Unit)	versendet Produktionsmeldung :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Auslieferungslager Stpe(Org-Unit)	erhält Produktionsmeldung :Operation Pre(Produktionsmeldung stimmt mit Produkt auf Lager überein) Goal(Produkt ist auftragsgemäß gefertigt) In(:Produktionsmeldung)
	:Fertigung Stpe(Org-Unit)	meldet Ausführung des Fertigungsauftrags :Action	:Interaction Flow(Information) Coord(feedback) Concurrency(asynchronous)	:Produktions-Steuerung Stpe(Org-Unit)	Auslieferung eines Produkts veranlassen :Operation Pre(Produktionsmeldung kann Fertigungsauftrag zugeordnet werden) Goal(Fertigungsauftrag ist erfüllt) In(:Produktionsmeldung)
	:Produktions-Steuerung Stpe(Org-Unit)	versendet Versandauftrag :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Auslieferungslager Stpe(Org-Unit)	Produkt ausliefern :Operation Pre(Versandauftrag kann Produkt auf Lager zugeordnet werden) Goal(Produkt ist an Kunde versendet) In(:Versandauftrag)
	:Auslieferungslager Stpe(Org-Unit)	versendet Produkt :Action	:Interaction Flow(Material) Concurrency(asynchronous)	/Kunde :HiFi-Produzent Stpe(Organization) System(external)	erhält Produkt :Operation Pre(Lieferung unversehrt) Goal(Lieferung ist eingetroffen) In(:Lieferung)
	:Auslieferungslager Stpe(Org-Unit)	versendet Lieferschein :Action	:Interaction Flow(Information) Concurrency(asynchronous)	/Kunde :HiFi-Produzent Stpe(Organization) System(external)	vergleicht Lieferschein mit Produkt :Operation Pre(Lieferung entspricht Lieferschein und Bestellung) Goal(Lieferung ist akzeptiert) In(:Lieferschein)
	:Auslieferungslager Stpe(Org-Unit)	Fakturierung veranlassen :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Verkauf Stpe(Org-Unit)	fakturiert Auftrag :Operation Pre(Versandmeldung entspricht Auftrag) Goal(Auftrag ist fakturiert) In(:Versandmeldung)
	:Verkauf Stpe(Org-Unit)	Offenen Posten veranlassen :Action	:Interaction Flow(Information)	:Buchhaltung Stpe(Org-Unit)	erzeugt OP Debitoren :Operation Pre(True)

	Initiator	Initiator Resp.	Connection	Participant	Participant Resp.
			Concurrency(asynchronous)		Goal(Offener Posten Debitoren existiert) In(:Versandmeldung)
	:Verkauf Stpe(Org-Unit)	versendet Rechnung :Action	:Interaction Flow(Information) Concurrency(asynchronous)	/Kunde :HiFi-Produzent Stpe(Organization) System(external)	erhält Rechnung :Operation Pre(Lieferung ist akzeptiert) Goal(Rechnung wird bezahlt) In(:Rechnung)
	/Kunde :HiFi-Produzent Stpe(Organization) System(external)	bezahlt Rechnung :Action	:Interaction Flow(Money) Concurrency(asynchronous)	:Bank System(external)	erhält Zahlung :Operation Pre(Empfänger bekannt, Betrag gedeckt) Goal(Zahlung ist überwiesen) In(:Zahlung)
	:Bank Stpe(Organization) System(external)	meldet Zahlungseingang :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Buchhaltung Stpe(Org-Unit)	gleicht OP Debitoren aus :Operation Pre(Zahlungsbeleg kann offenem Posten Debitoren zugeordnet werden) Goal(Offener Posten Debitoren existiert nicht) In(:Zahlungsbeleg)
	:Buchhaltung Stpe(Org-Unit)	informiert Verkauf über Zahlungseingang :Action	:Interaction Flow(Information) Concurrency(asynchronous)	:Verkauf Stpe(Org-Unit)	schließt Auftrag ab :Operation Pre(Zahlungsbeleg entspricht Auftrag) Goal(Auftrag ist abgeschlossen) In(:Zahlungsbeleg)

Skript 228: Geschäftsprozess der einstufigen Auftragsfertigung eines Produktionsunternehmens; dargestellt auf der Mesoebene

10 Anhang 2: Übersetzungstabelle

10.1 Stereotype

In den Skripten werden für die Stereotype englische Bezeichnungen verwendet. Die folgende Tabelle beinhaltet die entsprechenden Begriffe der Organisationslehre:

Deutscher Begriff	**Englische Übersetzung**
Abteilung	Department
Aktionseinheit	Action unit
Aktionsträger	Action bearer
Arbeitsleistung	Performance
Arbeitsmittel	Work instrument
Arbeitsträger	Work bearer
Aufgabenträger	Task bearer
Beteiligter, Mitwirkender	Participant
Entität	Entity
Geld	Money
Geldfluss	Money flow
Geschäftsobjekt	Business Object
Gruppe	Group
Information	Information
Individuum	Individual
Initiator, Urheber	Initiator
Leistung	Service
Materie	Material

Materialfluss	Material flow
Organisation	Organization
Organisationseinheit	Org-Unit
Problem	Problem
Prozeß	Process
Sachmittel	Resource
Stelle	Position
Wissen	Knowledge
Ziel	Goal

Tabelle 15: Übersetzung der im Text verwendeten Stereotype

10.2 Vordefinierte Bezeichner

Im Metamodell werden für die Prädikate englische Begriffe verwendet. Die folgende Tabelle beinhaltet deren deutsche Übersetzung:

Englischer Begriff	**Deutsche Übersetzung**
Access Mode	Zugriffsmodus
Acknowledgement Identifier	Benachrichtigungsbezeichner
Activity Mode	Aktivitätenmodus
Cardinality Expression	Kardinalitätsausdruck
Change Mode	Veränderungsmodus
Changeability Identifier	Veränderbarkeitsbezeichner
Class List	Klassenliste
Concurrency Identifier	Nebenläufigkeitsbezeichner
Concurrency Kind	Synchronisationsart
Constraint Expression	Zusicherungsausdruck
Continuity Identifier	Kontinuitätsbezeichner
Coordination Identifier	Koordinationsbezeichner

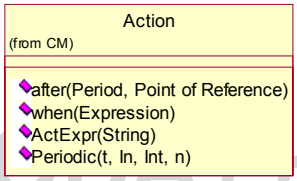
Cost Expression	Kostenausdruck
Error Condition	Fehlerbedingung
Expression	Ausdruck
Flow Type	Flusstyp
Form Identifier	Form-Bezeichner
Identifier	Bezeichner
Identifier List	Bezeichnerliste
Interaction Kind	Interaktionsart
Interaction Mode	Interaktionsmodus
Lifespan Identifier	Lebensdauerbezeichner
Operationname	Operationsname
Order Kind	Abfolgeart
Period	Zeitraum
Point of Reference	Bezugspunkt
Propagation Kind	Ausbreitungsart
Relationship Type	Beziehungstyp
Selector Expression	Auswahlausdruck
Statechange Mode	Übergangsmodus
String	Textkette
Time Expression	Zeitausdruck
Type Identifier	Typbezeichnung
Variability Identifier	Variabilitätsbezeichner
Visibility Kind	Sichtbarkeitsbezeichner

Tabelle 16: Übersetzung der im Text verwendeten Bezeichner für Prädikate

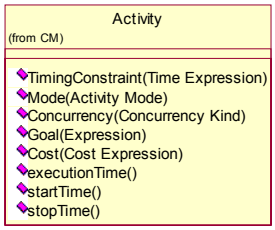
11 Anhang 3: Die Schnittstellen der Klassen des Metamodells

Die im Skriptmodell verwendeten Prädikate entsprechen im Metamodell des Konzeptuellen Schemas (vgl. Abschnitt 4.4.1) Operationen der Metaklassen. Die folgenden Abbildungen fassen die Operationen der Metaklassen zusammen.

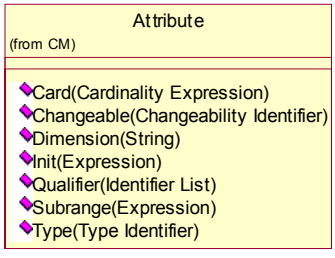
11.1 Action



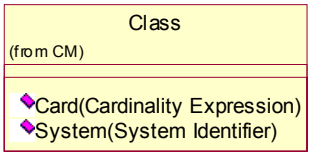
11.2 Activity



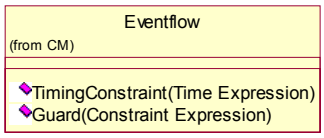
11.3 Attribute



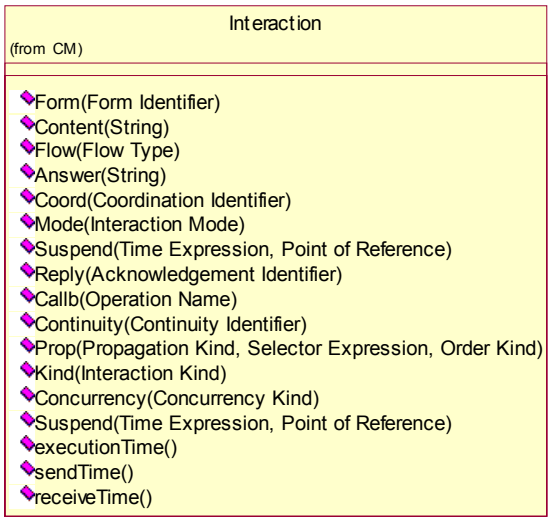
11.4 Class



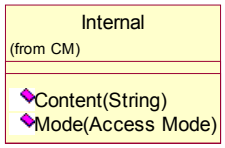
11.5 Eventflow



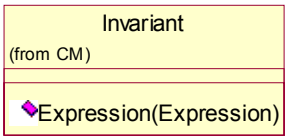
11.6 Interaction



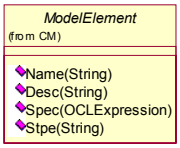
11.7 Internal



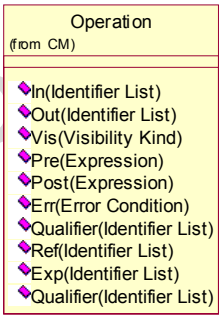
11.8 Invariant



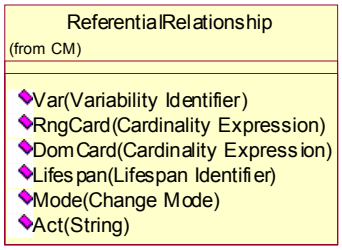
11.9 ModelElement



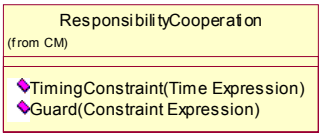
11.10 Operation



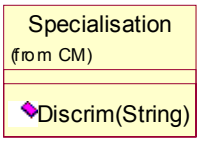
11.11 ReferentialRelationship



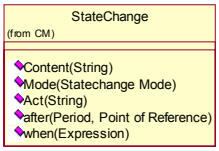
11.12 ResponsibilityCooperation



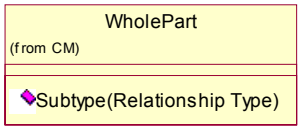
11.13 Specialisation



11.14 StateChange

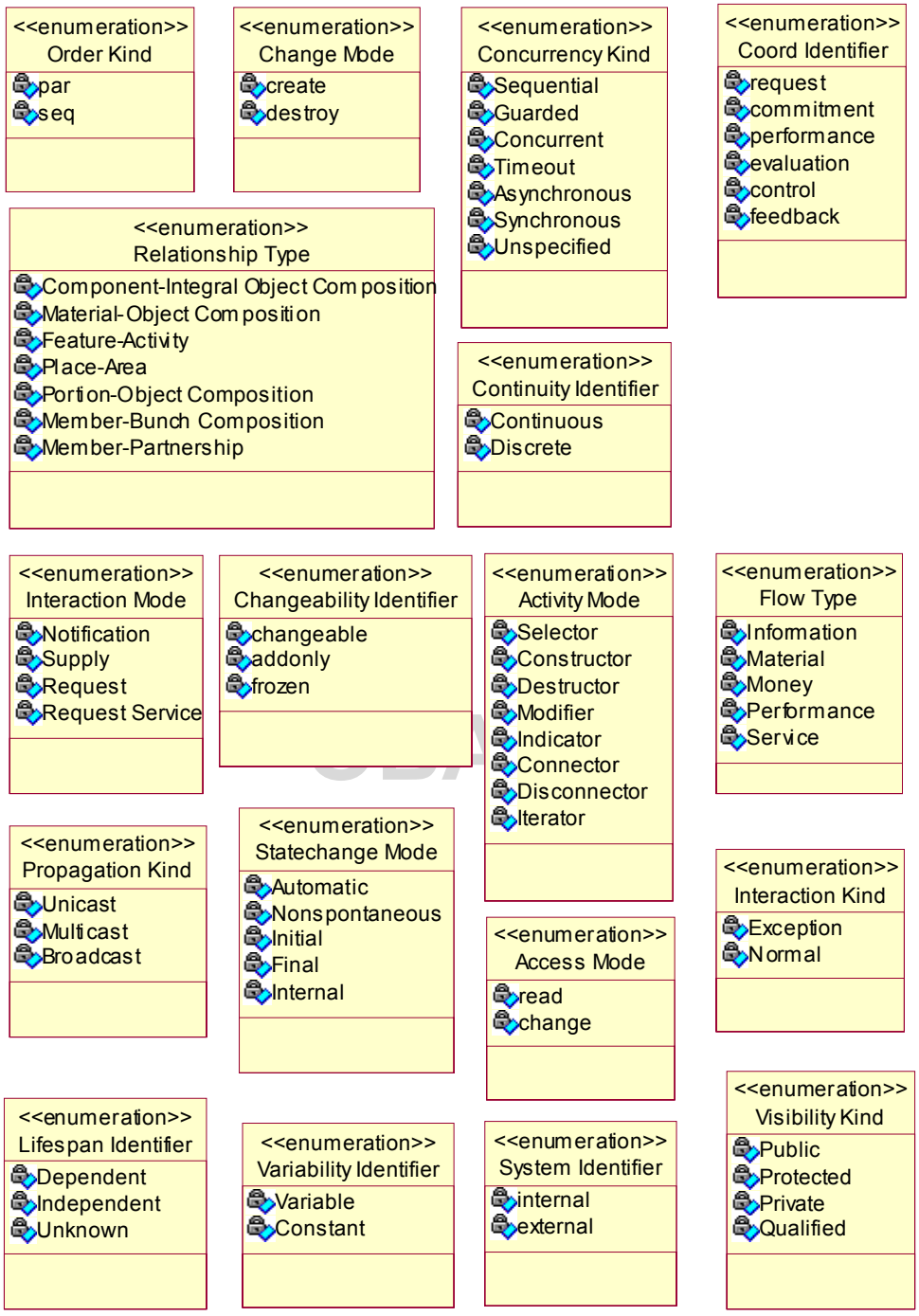


11.15 WholePart



11.16 Vordefinierte Aufzählungsdatentypen

Die folgenden Aufzählungsdatentypen sind die vordefinierten Terme der entsprechenden Prädikate (vgl. Abschnitt 5.2.6).


<<enumeration>>
Order Kind
par
seq
<<enumeration>>
Change Mode
create
destroy
<<enumeration>>
Concurrency Kind
Sequential
Guarded
Concurrent
Timeout
Asynchronous
Synchronous
Unspecified
<<enumeration>>
Coord Identifier
request
commitment
performance
evaluation
control
feedback
<<enumeration>>
Relationship Type
Component-Integral Object Composition
Material-Object Composition
Feature-Activity
Place-Area
Portion-Object Composition
Member-Bunch Composition
Member-Partnership
<<enumeration>>
Continuity Identifier
Continuous
Discrete
<<enumeration>>
Interaction Mode
Notification
Supply
Request
Request Service
<<enumeration>>
Changeability Identifier
changeable
addonly
frozen
<<enumeration>>
Activity Mode
Selector
Constructor
Destructor
Modifier
Indicator
Connector
Disconnector
Iterator
<<enumeration>>
Flow Type
Information
Material
Money
Performance
Service
<<enumeration>>
Propagation Kind
Unicast
Multicast
Broadcast
<<enumeration>>
Statechange Mode
Automatic
Nonspontaneous
Initial
Final
Internal
<<enumeration>>
Access Mode
read
change
<<enumeration>>
Interaction Kind
Exception
Normal
<<enumeration>>
Lifespan Identifier
Dependent
Independent
Unknown
<<enumeration>>
Variability Identifier
Variable
Constant
<<enumeration>>
System Identifier
internal
external
<<enumeration>>
Visibility Kind
Public
Protected
Private
Qualified

12 Anhang 4: Abbildungen, Tabellen und Skripte

12.1 Abbildungen

12.2 Tabellen

12.3 Skripte

13 Anhang 5: Glossar

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Ablauforganisation	A. ist die Gestaltung von Arbeitsprozessen. Als Fortsetzung der Aufbauorganisation ist sie (nach KOSIOL) auf die Abwicklung von Arbeitsabläufen auf der Ebene einzelner Stellen beschränkt.	1	[Kosiol 1962]
Abstrakter Datentyp	Ein a. D. wird ausschließlich über seine Operationen definiert. Die interne Repräsentation der Daten und die verwendeten Algorithmen der Operationen sind von außen nicht sichtbar.	2	[Balzert 1999: S. 23]
Abstrakte Klasse	Eine a. K. ist eine Klasse, auf deren Grundlage *keine* Objekte erzeugt werden können.	2	[BalzertH 1999: S. 533]
Abstraktion	a) Unter Abstrahieren wird Verallgemeinern, aus dem Besonderen das Allgemeine Entnehmen verstanden. b) Eine A. ist ein verallgemeinerter, unanschaulicher Begriff. c) Eine A. ist die Unterdrückung von Details und die Konzentration auf die wesentlichen Eigenschaften.	2	[Duden 1993], [Partsch 1998: S. 42]
Aggregation	a) Mit A. wird im Duden eine Anhäufung von Kenntnissen und Fakten bezeichnet, mit Aggregat ein Satz von zusammen wirkenden einzelnen Maschinen, Apparaten, Teilen. b) Mit A. wird die Beziehung zwischen einem	4	[Duden 1993]

[1] Ein Beleg ist dann angegeben, wenn die Definition einer Quelle entnommen oder in Anlehnung an eine Quelle formuliert ist.

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	Aggregat und seinen Komponenten bezeichnet.		
Aktives Objekt	Objekte werden als aktiv angesehen, wenn sie ohne ein vorhergehendes Ereignis Aktionen ausführen.	6	
Aktivitätentreiber	Ein A. bzw. ein Ressourcentreiber ist die Einflussgröße, mit deren Hilfe die Inanspruchnahme von Objekten gemessen werden kann. Aktivitätentreiber können Mengeneinheiten, Verarbeitungszeiten, Mitarbeiterstundensätze, Maschinenstundensätze etc. sein.	6	[HeilmannM 1997: S. 31]
Aktivitätenkostensatz	Der A. bzw. Prozesskostensatz ist der Kostensatz, der bei der Ausführung der Aktivität für einen Aktivitätentreiber anfällt.	6	[Raster 1994], [Coenenberg 1991], [Glaser 1992], [Schulte-Zurhausen 1999: S. 74]
Akteur	Ein *Akteur* (engl. *actor*) ist eine Entität, die sich außerhalb des aktuell betrachteten Systems befindet. Der Akteur interagiert mit dem System. Diese Interaktion stellt für das System Ereignisse dar, auf die es reagieren muss.	2	
Anwendungssysteme	A. dienen der Erzeugung, Speicherung, Übertragung, Auswertung oder Transformation von Informationen.	1	
Architektur	Eine Architektur beschreibt die Struktur eines Systems durch ihre Komponenten und ihre Beziehungen untereinander.	3	[Rechenberg 1999], [Wettstein 1993]
Argument	Das A. ist die Spezifikation der zulässigen Eingabe einer Prozedur. Innerhalb der Prozedur wird das Argument als Parameter bezeichnet.	2	[Rechenberg 1999: S. 472, 495]
Artefakt	Als A. wird im SE jedes Zwischenergebnis bezeichnet, welches während des Softwareentwicklungsprozesses erstellt wird.	2	
Assoziation	a) Die A. ist die Menge von Verknüpfungen mit einer gemeinsamen Struktur und Semantik.	4	[Rumbaugh 1993: S. 34], [Duden 1993]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	b) Der Duden bezeichnet mit A. eine ursächliche Verknüpfung von Vorstellungen.		
Attribut	a) Die Gesamtheit der Eigenschaften eines Objekts bezeichnet man als seine Struktur. Die einzelnen Eigenschaften heißen A. b) A. sind die inhärenten Eigenschaften, Qualitäten oder Besonderheiten von Konzepten. c) In der objektorientierten Modellierung werden die Attribute als Abstraktionen der Informationen verwendet, die zu einem Objekt gehören. d) A. sind die statischen, zeitinvarianten Merkmale eines Konzepts. Sie bilden das sog. Gedächtnis des Objekts.	2, 4	[Kappel 1996: S. VII, 25 f.], [Booch 1995: S. 111], [Webster 1993], [Schienmann 1997: S. 38]
Aufbauorganisation	A. ist die Zerlegung der Gesamtaufgabe einer Organisation in Teilaufgaben. Die Teilaufgaben werden zu Stellen zusammengefasst, zwischen denen sich Beziehungszusammenhänge ergeben.	1	[Kosiol 1962]
Ausnahme, Exception	a) Eine A. ist ein Anzeichen dafür, dass eine Invariante nicht erfüllt wurde oder erfüllt werden kann. b) Eine A. ist eine Fehlerabstraktion. Sie stellt eine Aktivität eines Objekts dar, durch die angezeigt wird, dass eine als außergewöhnlich betrachtete Situation eingetreten ist, auf die reagiert werden muss.	2, 4	[Booch 1995: S.63], [Oestereich 1998: S. 241], [Firesmith 1995: S. 163]
BACKUS-NAUR-Form	Eine BN-Form ist eine aus Produktionsregeln bestehende Grammatik	5	[Kopp 1988: S. 6 ff.], [Aho 1992: S. 32 ff.]
Betriebliche Informationssysteme	a) B. I. bestehen aus Menschen und Maschinen, die durch Kommunikationsbeziehungen verbunden sind und im Rahmen ihrer Aufgaben Informationen erzeugen, erfassen, übertragen, transformieren, speichern, benutzen oder bereitstellen. b) *„Ein betriebliches Informationssystem unterstützt die Leistungsprozesse*	1	[Ferstl 1998: S. 3, 8 f.], [Balzert 1996: S. 24], [Grochla 1974: S. 24, 25], [Rechenberg 1999: S. 1021], [Hansen 2001: S. 133]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	und Austauschbeziehungen innerhalb eines Betriebs sowie zwischen dem Betrieb und seiner Umwelt." [Hansen 2001: S. 133]		
Betriebswirtschaft	Eine B. ist – wie auch politische Parteien, Krankenhäuser, Schulen oder Gewerkschaften – eine Organisation, in der Menschen unter Verwendung technischer Hilfsmittel arbeitsteilig und kooperativ zusammenarbeiten. Unter dem Begriff der B. lassen sich sowohl Industrieunternehmen, Banken, Handelsbetriebe, Versicherungen, Speditionen wie auch Handwerksbetriebe und Einzelhandelsgeschäfte zusammenfassen. Neben diesen – auf Gewinnerzielung ausgerichteten – Unternehmungen lassen sich auch kommunistische Kolchosen, öffentliche Verkehrsbetriebe, und halbstaatliche Elektrizitätswerke unter dem Begriff B. zusammenfassen. Ihnen ist gemeinsam, dass sie Güter und Dienste produzieren, die menschliche Bedürfnisse befriedigen. Dabei handelt es sich in der Regel um Fremdbedarfsdeckung, wodurch B. von Haushalten unterschieden werden, die primär der Eigenbedarfsdeckung dienen.	1	[Heinen 1971: S. 26], [Heinen 1985: S. 16, 51, 115 – 119], [Bleicher 1991: S. 35], [Ferstl 1998: S. 65], [Hill 1994: S. 17]
Botschaft	Eine B. ist der Inhalt einer Nachricht.	2	
Computergestützte Informationssysteme	C. I. (CIS) stellen die automatisierten Teilsysteme des betrieblichen Informationssystems dar.	1	
Container-Beziehung	Container-Beziehungen sind Beziehungen zwischen einem Behälter und seinem Inhalt.	4	[Züllighoven 1998: S. 89 f.]
Daten	D. sind die Darstellungen von Informationen. Von D. spricht man, wenn Informationen eine einheitliche Struktur besitzen, aus einer vordefinierten Menge von Zeichen bestehen	4	[Ehrich 1989: S. 1], [Krüger 1994: S. 143], [Schmidt 1985: S. 76], [Schneider 2000]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	und in einer maschinell zu verarbeitenden und zu speichernden Form vorliegen. D. sind Informationen, die von Computern verarbeitet werden.		
Delegation, Verwendung im objektorientierten Ansatz	a) (im Objektansatz) D. ist, wenn ein Objekt im Rahmen der Erfüllung seiner Aufgabe andere Objekte mit Teilaufgaben beauftragt. b) (in der Organisationslehre) D. ist die dauerhafte Übertragung von Entscheidungsaufgaben sowie der zugehörigen Kompetenzen und Verantwortung an hierarchisch nachgeordnete Stellen.	2	[Firesmith 1995: S. 126]; [Schulte-Zurhausen 1999: S. 192]
Dekomposition	D. ist der im Software Engineering übliche Begriff für eine systematische Aufgliederung [in elementare Komponenten]. Üblicherweise ist damit eine D. von Funktionen gemeint.	1	
Dienstleistung	Eine D. ist ein immaterielles Produkt, welches von personellen und materiellen Leistungsträgern an einen externen Faktor, der sich nicht im uneingeschränkten Verfügungsbereich des Leistungsträgers befindet, erbracht wird und das teilweise eine materielle Trägersubstanz benötigt.	1	[Corsten 1990: S. 23]
dynamische Sicht	Die d. S. ist die gemeinsame Betrachtung des funktionalen Verhaltens und der Kontrollaspekte.	2	
Ebene	a) (im Objektansatz) E. ist die Granularität bzw. der Detaillierungsgrad, mit der etwas betrachtet wird. Im Objektansatz unterscheidet man die Systemebene, Elementebene und Intra-Elementebene. b) (in der GPM) E. ist die Granularität bzw. der Detaillierungsgrad, mit dem Prozesse betrachtet werden. In der GPM unterscheidet man die Makro-, Meso- und	2, 7	[Corsten 1997b: S. 16], [Rummler 1991: S. 15 ff.], [Davenport 1993: S. 90 f.]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	Mikroebene.		
Epistemisches Primitiv	Ein e. P. steht für eine Klasse gleichartiger Sachverhalte, die so allgemein sind, dass ihr Auftreten nicht an einen bestimmten Diskursbereich (oder eine Klasse bestimmter Diskursbereiche) gebunden ist – epistemische Primitive sind domänenunabhängig.	4	[Reimer 1991: S. 15]
Ereignis	a) Ein E. ist ein Zwischenfall, auf den es einer Reaktion bedarf. b) Ein E. ist ein besonderer, nicht alltäglicher Vorgang, Vorfall, Geschehnis. c) Als E. wird das Eintreten eines zu beachtenden (in der UML: „bemerkens-werten“) Umstandes bezeichnet. d) Ein E. ist ein abstraktes Modell für eine diskrete Veränderung oder eine zeitlose Aktivität.	2	[Hutt 1994: S. 169], [Duden 1993]
Extension	Die Menge der aus einer Klasse gebildeten Objekte bezeichnet man als ihre E.	2	[Burkhard 1994: S. 48]
Ganzes-Teil	Die G.-T.-Beziehung fasst alle Beziehungs-typen zusammen, in denen ein Konzept Anteil an einem anderen Konzept hat.	4	
Geschäftsprozess	a) (in der BWL) *„Allgemein ist ein G. eine zusammengehörende Abfolge von Unterneh-mensverrichtungen zum Zweck einer Leistungserstellung. Ausgang und Ergebnis des G. ist eine Leistung, die von einem internen oder externen „Kunden“ angefordert und abgenommen wird.“* b) (im Objektansatz) Ein G. ist ein Prozess, in dessen Interaktions-beziehungen hauptsächlich Informationen ausgetauscht werden. c) (hier verwendete Sichtweise) Ein G. ist ein primär informations-verarbeitender Prozess in einem sozio-technischen System, der stellen-, abteilungs- und ggf. organisationsübergreifend betrachtet	1, 7	[Scheer 1998b: S. 3]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	wird, unabhängig davon, ob er durch computerisierte Informationssysteme unterstützt wird. G. stellen grundsätzlich das Zusammenwirken von menschlichen und sachlichen Aktionsträgern dar.		
Geschäftsvorfall	Ein G. ist das ausgeführte Exemplar eines Geschäftsprozesses.	7	[Kueng 1995: S. 81]
Halbfabrikat	Ein H. ist eine Softwarebibliothek, eine Komponente, ein Rahmenwerk oder ein Generator, welcher üblicherweise heute bei der Entwicklung von Softwaresystemen verwendet wird.	1	
HAREL-Statechart	Ein H. ist ein hierarchisch geschachteltes und Parallelität abbildendes Zustandsdiagramm auf der Basis kombinierter MEALY/MOORE-Automaten. Das bedeutet, dass sowohl in Zuständen wie auch in Zustandsübergängen Aktivitäten stattfinden können.	2	[Harel 1987], [Harel 1987b], [Harel 1998], [Coleman 1994]
Information	a) I. ist zweckorientiertes, auf den Vollzug von Aktionen zur Erreichung von Zielen bezogenes Wissen. b) I. ist Wissen über ein Ereignis, einen Tatbestand oder Sachverhalt der Wirklichkeit. c) I. ist beseitigte Unsicherheit.	1	[Wild 1966: S. 97], [Heinen 1985: S. 62], [Schneider 2000], [Hering 1995: S. 3].
Information Hiding (Geheimnisprinzip)	a) Das I. H. ist das Verbergen der Interna eines Objekts. b) I. H. ist das Erzeugen und Verarbeiten von Ereignissen durch Objekte, ohne das man weiß, was in ihrem Inneren passiert.	2	[Parnas 1972]
Informationsprozess	Der I. umfasst das Erzeugen, Speichern, Übertragen, Auswerten oder Transformieren von Informationen.	1	[Wild 1966: S. 139]
Informations-technologie	I. ist die Gesamtheit aller technischen bzw. computergestützten betrieblichen Informations-	1	

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	und Kommunikationssysteme (IKS).		
Initiator	Der I. ist das Konzept, von dem eine Beziehung ausgeht.	3, 4	
Instanz	Eine I. bzw. ein Exemplar ist eine konkrete Manifestation einer Abstraktion. Die Instanzierung ist das Ergebnis der Bildung eines Exemplars (Extension) auf der Basis einer Klasse.	4	[Booch 1999: S. 185]
Intension	a) Die I. umfasst die Charakteristika bzw. Merkmale einer Klasse. b) Mit I. wird die vollständige Definition einer Abstraktion bezeichnet.	4	[Firesmith 1995: S. 219]
Interaktion	a) Die I. beinhaltet alle Formen der Nachrichtenübertragung oder Sender-Empfänger-Beziehung. b) Eine I. ist aufeinander bezogenes Handeln.	4	[Henderson-Sellers 1998], [Firesmith 1995: S. 219], [Schienmann 1997: S. 11 ff.]
Invariante	Eine I. ist eine auf Attributausprägungen und Beziehungen bezogene Abstraktion einer Integritätsregel. Invarianten müssen immer erfüllt sein.	4	[Burkhardt 1997: S. 125]
Invariante Bedingung	Eine i. B. ist eine logische Bedingung, deren Zutreffen nicht verletzt werden darf. Invariante Bedingungen gelten unabhängig vom Aufruf einer Operation.	2	[Booch 1995: S. 63]
Kapselungsprinzip	Das K. verbindet die Interna eines Objekts und die auf diese Interna zugreifenden Operationen zu einer Einheit.	2	[Berard 1993: S. 63 ff.]
Klasse	Eine K. ist die Struktur und das Verhalten gleichartiger Objekte. b) Eine K. ist ein Konzept, dessen Extension kein, ein oder mehrere Objekte enthält. c) Eine K. ist umgangssprachlich eine Gruppe von Dingen oder Begriffen mit gemeinsamen, sich von	2	[Meyer 1997], [Duden 1993]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	anderen unterscheidbaren Merkmalen.		
Klassifikation	K. ist die Zusammenfassung ähnlicher Dinge (Objekte) zu Klassen oder Kategorien.	2	[Taivalsaari 1996: S. 441]
Kollaboration	Eine K. ist in der UML ein Ausschnitt aus der statischen Modellstruktur, die genau jene Elemente enthält, die zur Erreichung eines Ziels kooperieren.	2	[Hitz 1999: S. 117]
Kommunikation	Die K. ist der wechselseitige Informationsaustausch zwischen mindestens zwei dynamischen, informationserzeugenden und informationsverarbeitenden Systemen.	1	[Klaus 1979: S. 312] [Hoyer 1988]
Konfiguration	a) Die K. (auch *Objektkonfiguration*) ist im Objektansatz die Struktur eines Systems und der Objekte zu einem bestimmten Zeitpunkt. b) Die O. ist der Teil des konkreten Zustands, der durch die aktuellen Beziehungen eines Objekts gebildet wird.	2, 6	
Kongruenzprinzip	Das K. ist eine Empfehlung zur Stellenbildung, nach der Aufgaben, Kompetenzen und Verantwortung in etwa deckungsgleich sein sollten. Damit ein Aufgabenträger seine Aufgaben ausführen kann, ist er mit den notwendigen Rechten zu versehen. Um die Kontrolle sicherzustellen, ist eine angemessene Rechenschaftspflicht (Verantwortung) vorzusehen.	8	[Krüger 1994: S. 47]
Kontrollstrang	Als K. wird die Kontrolle über einen Prozess bezeichnet. Übliche Synonyme sind Faden, Handlungsfaden oder *thread of control*.	2	[Rechenberg 1999: S. 635]
Konzept	K. ist der allgemeinste Begriff für alle Dinge der Welt, unabhängig davon, ob es ein konkretes oder abstraktes Ding beschreibt.	4	[Alhir 1998: S. 41]
Konzeptuelles Modell	Ein K. ist ein Modell einer tatsächlichen oder	2	[Frank 1997 f.: S. 3]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	gedachten Domäne unter Vernachlässigung implementierungsrelevanter Aspekte.		
Koordination	Die K. ist die Abstimmung von Objekten auf ein Ziel.	6	[Schulte-Zurhausen 1999: S. 203 ff.]
Lebenszyklus	Der L. ist die zulässige Reihenfolge, mit der die Operationen einer Klasse aufgerufen werden können. Diese Ausführungsfolgen werden auch als *gültiger Lebenszyklus* bezeichnet.	2	[Kappel 1996: S. 45], [Martin 1995: S. 90], [Shlaer 1992: S. 33]
Leistung	a) (im Objektansatz) Die Leistung (bzw. Dienstleistung) ist die Operation, die ein Objekt anderen Objekten zur Verfügung stellt. b) (in der GPM) Die Leistung ist das materielle oder immaterielle Ergebnis eines Prozesses.	2	[Waldén 1995], [Scheer 1998]
Mereologische Beziehung	Eine m. B. ist eine Beziehung zwischen den Teilen und einem Ganzen.	4	[Wedekind 1992: S. 35], [Winston 1987: S. 418]
Metamodell	Ein M. definiert die verfügbaren Arten von Modellbausteinen, die Regeln für die Verwendung von Modellbausteinen sowie die Bedeutung (Semantik) der Modellbausteine und Beziehungen. Vereinfacht ausgedrückt handelt es sich bei einem M. um ein Modell eines Modells.	2	[Ferstl 1998: S. 120], [Martin 1999: S. 307]
Metapher	Eine M. ist eine bestimmte Sichtweise, die einem Modellierungsansatz zum Grunde liegt.	1	
Methode	a) (allgemein) Eine M. ist ein planmäßiges Verfahren zur Erreichung eines Ziels. b) (im Objektansatz) Eine M. ist im Objektansatz die Realisierung bzw. Implementierung einer Operation. c) (im Software Engineering) Eine Methode ist im Software Engineering eine Einheit, die mindestens aus einem Modellierungsansatz und einem Vorgehensmodell besteht.	2	[OMG 1992: S. 36]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Modell	Ein M. ist ein innere Beziehungen und Funktionen von etwas abbildendes bzw. [schematisch] veranschaulichendes [und vereinfachendes, idealisierendes] Objekt, Gebilde.	2	[Duden 1993]
Modellierungsansatz	a) Ein M. ist ein Beschreibungsrahmen, durch den die bei der Modellierung verwendete Sichtweise auf Original und Modell sowie das bei der Modellierung verwendete Begriffs-system festgelegt wird. b) Ein M. besteht aus in dieser Arbeit aus Metapher, Metamodell, Repräsentation, Architektur und Anwendungs-beispielen.	2	[Ferstl 1998: S. 119]
Nachbedingung	Siehe *Vorbedingung*.		
Nachricht	Eine N. ist im Objektansatz die Spezifikation einer Kommunikation zwischen Objekten, durch die Informationen mit der Absicht übertragen werden, dass Aktivitäten folgen.	2	[Booch 1999: S. 210]
Namensraum	Der N. ist ein Bereich, innerhalb dessen ein Bezeichner eindeutig sein muss.	4	
Nebenläufigkeit	Aktivitäten sind nebenläufig, wenn sie gleichzeitig oder voneinander unabhängig durchgeführt werden können.	6	
Nomologische Hypothese	Als n. H. werden allgemein gültige Zusammenhänge im Sinne von gesetzes-mäßigen Eigenschaften, Zustands- und Ereigniszusammenhängen bezeichnet.	2	[Wild 1966: S. 56]
Objekt	a) O. sind greifbare und/oder sichtbare Dinge, etwas, das intellektuell wahrnehmbar ist, oder etwas, worauf sich unser Denken und Handeln bezieht. b) Objekte sind Individualkonzepte und implizit immer die Extension einer Klasse.	2, 4	[Booch 1995: S.109], [D'Souza 1998: S. 2 – 78], [Englmeier 1997: S. 9], [Martin 1993]
Objektorientiertes	Im Objektansatz ist das o. U. ein System von	1	

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Unternehmensmodell	Objekten, die untereinander in Beziehungen stehen und durch Nachrichtenaustausch miteinander und mit ihrer Umwelt interagieren.		
Objektkonfiguration	Siehe *Konfiguration*.		
Operation	Eine O. ist eine Abstraktion des Verhaltens von Objekten, welches durch den Erhalt einer Nachricht – als Handlungsaufforderung, Reiz oder Stimulus bezeichnet – ausgelöst wird.	4	
Organisation	Eine O. ist ein zielgerichtetes, offenes, dynamisches, soziotechnisches System, welches Informationen gewinnt und verarbeitet. Zielgerichtet, weil bestimmte Ziele in Form zukünftiger Zustände angestrebt werden. Offen, weil sie mit ihrer Umwelt in Austausch-beziehungen steht. Dynamisch, weil die Systemelemente durch ihr Zusammenwirken Aktivitäten zeigen. Es handelt sich um soziotechnische Systeme, weil mehrere Personen arbeitsteilig zusammenarbeiten und zur Erreichung ihrer Ziele Werkzeuge verwenden.	1	[Heinen 1985: S. 51, 53, 62, 72], [Kappler 1991: S. 76], [Bleicher 1991: S. 35], [Ferstl 1998: S. 59], [Striening 1988: S. 6], [Hill 1994: S. 17 – 26]
Participant	Der P. ist das Konzept, welches an einer Beziehung teilhat. Synonyme: Teilnehmer, Beteiligter.	3, 4	
Partition	P. ist die Zerlegung in einzelne Teile und sukzessive Konzentration auf einzelne individuelle Komponenten und Teilsysteme. Siehe auch *Dekomposition*.	2	[Partsch 1998: S. 43]
Phänomen	P. wird im Sinne von Erscheinung verwendet. Damit sind die äußeren Eigenschaften von Dingen und Prozessen gemeint, die dem Betrachter durch Anschauung bzw. Wahrnehmung oder Erfahrung gegeben sind.	2	[Klaus 1972], [Ferber 1999]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Polymorphie	P. ist die Eigenschaft, durch die die Objekte verschiedener Klassen Operationen mit gleichem Namen, aber unterschiedlicher Semantik besitzen können. Das hat zur Folge, dass unterschiedliche Objekte auf den Erhalt der gleichen Nachricht unterschiedlich reagieren.	2	[BalzertH 1999: S. 256 ff.], [Booch 1995], [Wirfs-Brock 1993], [Blair 1989]
Prädikat	a) (allgemein) P. ist eine Bestimmung von Gegenständen durch einen sprachlichen Ausdruck. b) (Verwendung in dieser Arbeit) P. sind Operationen der Metaklassen.	4	[Duden 1993]
Primitiv	Ein P. bzw. *Modellprimitiv* ist in dieser Arbeit das kleinste definierte Element eines Modellierungsansatzes oder einer Programmiersprache.	2	
Projektion	Eine P. ist die Betrachtung eines Systems unter verschiedenen Gesichtspunkten, jeweils hinsichtlich einer Teilmenge seiner Eigenschaften.	2	[Partsch 1998: S. 43]
Protokoll	Das P. einer Klasse ist die Schnittstelle und die Bedingungen, unter denen die in der Schnittstelle dokumentierten Operationen verwendet werden können.	2	[Züllighoven 1998]
Prozess	a) (allgemein) „*A continuous and regular action or succession of actions, taking place or carried on in a definite manner and leading to accomplishment of some result; a continuous operation or series of operations."* Oxford English Dictionary (zitiert nach [Graham 1997: S. 179]) b) (betriebswirtschaftlich) Ein Prozess ist das Zusammenwirken von Menschen, Material, Sachmittel, Richtlinien, Anweisungen, Methoden und Informationen, welches die Erstellung einer Leistung zum Ziel hat. c)	1, 7	[Schulte-Zurhausen 1995: S. 50, 59 ff.], [Striening 1988: S. 59]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	(objektorientiert) Ein P. ist eine Abfolge von Objektinteraktionen, durch die die beteiligten Objekte Informationen austauschen und Verhalten zeigen.		
Prozesskette	Eine P. ist eine Abfolge funktional zusammenhängender Aktivitäten.	7	[Schulte-Zurhausen 1999: S. 53 f.], [Eversheim 1995: S. 15]
Referentielle Beziehung	Eine r. B. ist eine Beziehung, bei der ein Konzeptknoten einen Verweis (Referenz) auf einen anderen Konzeptknoten besitzt.	4	[Firesmith 1997: S. 105]
Repräsentation	a) (allgemein) R. ist die verwendete konkrete Form einer Darstellung. b) (Verwendung in dieser Arbeit) Die R. legt fest, welche äußere Form ein Modell besitzt.	2	
Sachmittel	Ein S. ist ein materielles Hilfsmittel zur Prozessabwicklung.	6	[Schulte-Zurhausen 1999: S. 50, 67]
Schnittstelle	Die S. ist die Gesamtheit der Operationen einer Klasse.	2	
Semiformaler Ansatz	Als s. A. wird ein Ansatz bezeichnet, der eine definierte Syntax und eine Menge vordefinierter Primitive verwendet. Die Verwendung natürlicher Sprache ist nur in eingeschränkter Weise zugelassen. S. A. verfügen über keine definierte Semantik. Formale Ansätze sind dagegen Spezifikationssprachen wie VDM oder Z. Diese verfügen über eine definierte Semantik und basieren in der Regel auf mathematischen und logischen Kalkülen.	1	[Fensel 1994: S. 4], [Jones 1990], [Wordsworth 1992], [Andrews 1991], [Andrews 1997], [Lano 1995], [Sheppard 1994], [Turner 1994]
Signatur	Die S. setzt sich zusammen aus dem Namen einer Operation, dem Typ ihres Rückgabewertes und dem Typ ihrer Argumente.	5	[Wirfs-Brock 1993: S. 21]
Singleton	Ein S. ist eine Klasse, deren Extension ein Element beinhaltet.	6	[Rumbaugh 1999: S. 430]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Skript	a) Ein S. ist eine tabellarische Darstellungsform, die primär zur Abbildung von Objektinteraktionen verwendet wird. b) Der Begriff *Skript* stammt aus der Kognitionspsychologie und wird dort als Schema für bestimmte Ereignisabfolgen verwendet. c) In der Informatik versteht man unter Skript die prototyphaft beschriebene Folge von Ereignissen und Aktionen in einem bestimmten Kontext.	1	[Gibson 1990], [Rubin 1992], [Rubin 1994b], [Banyard 1995], [Schank 1972], [Schank 1975], [Schank 1977], [Bobrow 1975], [Rechenberg 1999: S. 986]
Software Engineering	Das S. E. ist ein Teilgebiet der praktischen Informatik, welches seit dem Ende der 60er-Jahre um eine ingenieurmäßige Softwareentwicklung bemüht ist. Als *ingenieurgemäß* werden Tätigkeiten angesehen, wenn sie planbar, wiederholbar, überprüfbar, methodisch unterstützt Produkte hervorbringen, die festgesetzten Qualitätsansprüchen genügen.	1	[Naur 1968]
Softwareentwicklung	a) S. ist ein aus mehreren Phasen bestehender Prozess, der mit dem Feststellen der notwendigen Leistungen eines zu erstellenden Softwaresystems beginnt und mit der Ausmusterung dieses Systems endet. b) Gegenstand der Softwareentwicklung ist die Abbildung von zu automatisierenden Aufgaben eines Informationssystems auf Rechner- und Kommunikationssysteme.	1	[Jacobson 1994], [Züllighoven 1998: S. 142], [Ferstl 1998]
Spezialisierung	Die S. ist das Ergebnis einer Abstraktionsbildung durch Klassifikation, Kategorisierung bzw. durch Subsumption.	4	[Henderson-Sellers 1998: Appendix E]
statische Sicht	Die s. S. ist die Betrachtung der Struktur. Die Struktur sind die Elemente des Systems und die Beziehungen zwischen diesen Elementen im Sinne der zeitinvarianten Anordnung der	2	

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	Elemente.		
Stereotyp	Ein S. ist ein Erweiterungsmechanismus, um neue problemspezifische Modellelemente definieren zu können, ohne ein Metamodell ändern zu müssen.	6	[Henderson-Sellers 2000: S. 33, 24], [Booch 1999: S. 29], [Frank 1997d: S. 48, 61], [Atkinson 2000: S. 34], [Oestereich 1998: S. 355]
Struktur	Eine S. beschreibt die Elemente eines Systems und die Beziehungen zwischen diesen Elementen im Sinne der zeitinvarianten Anordnung der Elemente.	2	
Substitutionsprinzip	Nach dem S. nimmt mit abnehmender Variabilität betrieblicher Tatbestände die Tendenz zu generellen Regelungen zu.	1	[Gutenberg 1973. S. 240]
Synchronisation	S. ist die Abstimmung von Aktivitäten im Bezug auf ihre Nebenläufigkeit.	6	
System	a) (im Objektansatz) S. sind innerhalb der Welt eine geordnete Menge (Gemeinschaft bzw. Gesellschaft) von interagierenden Objekten, für die man entscheidet, dass sie als ein Ganzes angesehen werden sollen. S. sind als Strukturen von Objekten definiert, die einander Nachrichten zusenden, auf Ereignisse reagieren, in Beziehungen zueinander stehen, von anderen Objekten Dienste in Form von Operationen anfordern und durch die Ausführung von Diensten ihren Zustand und den des Systems verändern. b) (in der Systemtheorie) Ein S. ist eine Menge von Elementen, die Eigenschaften haben und zwischen denen Beziehungen existieren. Ein S. ist von seiner Umwelt abgrenzbar. Ein Element eines S. kann selbst ein S. sein.	2	[Embley 1992], [Stoyan 1991: S. 184], [Saake 1993], [Berard 1993: S. 166, 175], [Reenskaug 1996: S. 36], [Klaus 1979: S. 800, 806, 807]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
Systemanalyse	Die S. ist ein Verfahren, durch welches die Elemente und Beziehungen von Systemen, die im Anfangsstadium unbekannt sind, durch sukzessive Annäherung ermittelt, Schwachstellen festgestellt, und darauf aufbauend ein neues System entworfen und realisiert wird.	2	[Krallmann 1994: S. 4]
Szenario	Ein S. ist eine mögliche Abfolge von Ereignissen.	2	[Linssen 1999b]
Taxonomie	Die T. ist der Zweig der Systematik, der sich mit dem praktischen Vorgehen bei der Klassifizierung von Lebewesen in systematischen Kategorien befasst.	4	[Duden 1993], [Meyer 1997: S. 864]
Taxonomische Beziehung	Eine t. B. ist eine Beziehung, bei der ein Konzept in irgendeiner Form durch ein anderes Konzept klassifiziert wird oder als Basis für dessen Definition verwendet wird.	4	
Term	Ein T. ist das aktuelle Argument eines Prädikats.	4	
Transition	Eine T. ist eine durch ein Ereignis ausgelöste Zustandsveränderung bzw. ein Zustandsübergang, wenn sie unteilbar (atomar) ist und ohne Zeitverbrauch stattfindet.	4	[OMG 1992: S. 37], [Embley 1992: S. 10, 61], [de Champeaux 1993: S. 67, 356]
Transaktion	a) In der Wirtschaftswissenschaft ist eine Transaktion der Prozess der Klärung und Vereinbarung eines Leistungsaustauschs. b) In der Informatik ist eine Transaktion eine vollständige Ausführung einer Aktivität.	6, 7	[Ferstl 1998: S. 185], [Lockemann 1993: S. 415 ff.], [Picot 1982], [Picot 1990]
Transaktionskosten	T. sind eine Form von Informationskosten, die zur Koordination wirtschaftlicher Leistungsbeziehungen notwendig sind. T. sind auch die Kosten, die beim An- und Verkauf von Wertpapieren anfallen.	7	[Picot 1982: S. 270]
Überladen	Eine Operation ist überladen, wenn mehrere	2	[Kappel 1996: S. 21 f.]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	Operationen einer Klasse den gleichen Namen, aber eine unterschiedliche Semantik haben.		
Unternehmen	Ein U. ist in der objektorientierten Sichtweise ein System von Objekten, die untereinander in Beziehungen stehen und durch Nachrichten-austausch miteinander und mit ihrer Umwelt interagieren.	1	
Verantwortlichkeit	V. sind in der objektorientierten Sichtweise Dienstleistungen, die ein Objekt aufgrund seiner Verträge bewerkstelligen kann. Eine Dienstleistung kann entweder die Ausführung einer Aktion oder die Lieferung von Informationen sein.	2	[Wirfs-Brock 1993: S. 63]
Vererbung	V. zwischen zwei Klassen bedeutet, dass eine Klasse ihre Intension an Erben weitergibt und diese nur nach bestimmten Regeln neue Elemente zur Intension hinzufügen bzw. ererbte verändern dürfen. Die erbende Klasse stellt eine Spezialisierung der vererbenden Klasse dar.	2, 4	[Frank 1998b], [Taivalsaari 1996], [Hitz 1999: S. 45 ff.], [Alhir 1998: S. 59 f.], [Meyer 1997: S. 459 ff., S. 808 ff.], [Liskov 1994], [Wegner 1988], [Papurt 1996], [Züllighoven 1998: S. 35]
Verhalten	a) (in der Systemtheorie) V. ist die Gesamtheit der möglichen Reaktionen eines dynamischen Systems (Systemverhalten) oder des Elements eines solchen Systems (Verhalten eines Elements) auf äußere Einwirkungen. b) (im Objektansatz) V. ist die Art und Weise, wie ein Objekt (1) in Form von Zustandsveränderungen reagiert und (2) durch Übergabe von Nachrichten an andere Objekte agiert. V. drückt im Objektansatz umgangssprachlich aus, dass man Objekte verändern kann, ihnen etwas übergeben kann, etwas von ihnen erhalten kann, etwas mit ihnen tun kann oder sie	2	[Klaus 1979: S. 888]

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	auffordern kann, etwas zu tun bzw. sich zu verändern.		
Vertrag	Ein V. spezifiziert im objektorientierten Ansatz die Bedingungen, unter denen ein Objekt eine Operation ausführt.	2	
Vorbedingung	Vor- und Nachbedingungen sind auf Operationen bezogene Abstraktionen von Integritätsregeln, die eingehalten werden müssen. Die V. muss vor dem Beginn der Operation eingehalten sein, die Nachbedingung muss nach der Beendigung der Operation eingehalten werden.	4	[Firesmith 1995: S. 27 f., 335 f.], [Martin 1999: S. 203], [Kappel 1996], [Waldén 1995], [Cook 1994]
Wirtschaft und Verwaltung	Der Bereich der W. u. V. umfasst Industrie und Handel, Banken und Versicherungen, Dienstleistungsunternehmen sowie öffentliche Betriebe und öffentliche Verwaltungen.	1	
Workflow	a) Ein W. ist der automatisierte Teil eines Geschäftsprozesses. b) Ein W. ist ein Vorgang, der von einem Workflow-Managementsystem ausgeführt wird.	1	[WfMC 1996], [Jablonski 1997: S. 24]
Zeitpunkt	Ein Z. ist eine ausdehnungslose Zeitangabe.	6	
Zeitdauer	Eine Z. ist die Größe eines Zeitintervalls. Zeitintervalle sind Paare von absoluten Zeitpunkten (Anfangszeitpunkt und Endzeitpunkt), die in einer Reihenfolge-beziehung zueinander stehen. Die Zeitdauer ist die Differenz zwischen Endzeitpunkt und Anfangszeitpunkt.	6	
Ziel	Ein Z. ist ein zu erreichender Zustand.	6	[Nordsieck 1955: S. 28]
Zustand (Status)	a) Der Z. umfasst die (normalerweise statischen) Attribute des Objekts und die aktuellen (normalerweise dynamischen) Werte dieser Attribute sowie die Beziehungen, die das	1, 4	[Booch 1995: S. 112], [OMG 1992: S. 36], [Coleman 1992: S. 9], [Englmeier 1997: S. 34],

Begriff	Definition	Def. in Kap.	Literaturverweis, Fundstelle[1]
	Objekt zu anderen Objekten besitzt. b) Der Z. stellt einen Vektor aller Attribute mit ihren aktuellen Werten und den Beziehungen dar, die ein Objekt zu einem bestimmten Zeitpunkt besitzt. c) Der Z. ist eine Abstraktion, die einer bestimmten Kombination von Attributwerten und aktuellen Beziehungen einen für das Objekt eindeutigen Namen gibt.		[Martin 1995: S. 87], [de Champeaux 1993: S. 64], [Rumbaugh 1993: S. 107]
Zustandsveränderung	Eine Z. ist die Modifikation des Wertes eines oder mehrere Attribute oder Beziehungen eines Objekts.	2	

14 Verwendete Literatur

[Abbott 1983] *Abbott, Russell J.:* Program Design by Informal English Descriptions. In: Communications of the ACM, Band 26, Nr. 11, 1983, S. 882 – 894.

[Adam 1996] *Adam, Dietrich:* Planung und Entscheidung. Modelle – Ziele – Methoden. 4. vollst. überarb. und wesentlich erw. Auflage. Wiesbaden 1996.

[Agha 1986] *Agha, Gul A.:* Actors: A Model of Concurrent Computation in Distributed Systems. Cambridge, MA, 1986.

[Agha 1986b] *Agha, Gul A.:* Concurrent Object-Oriented Programming. In: Communications of the ACM, Band 33, Nr. 9, 1990, S. 125 – 141.

[Aho 1992] *Aho, Alfred V.; Sethi, Ravi; Ullmann, Jeffrey D.:* Compilerbau. Band 1. 2. Nachdruck, Bonn 1992.

[Alexander 1977] *Alexander, Christopher; Ishikawa, Sara; Silverstein, Murray; Jacobson, Max; Fiksdahl-King, Ingrid; Angel, Shlomo:* A Pattern Language. Towns, Buildings, Construction. New York 1977.

[Alhir 1998] *Alhir, Sinan Si:* UML in a Nutshell. A Desktop Reference. Cambridge, Köln, Paris u. a. 1998.

[Allen 1983] *Allen, James F.:* Maintaining Knowledge about Temporal Intervals. In: Communications of the ACM, Band 26, Nr. 11, November 1983, S. 832 – 843.

[Allen 1995] *Allen, Arthur; de Champeaux, Dennis:* Extending the Statechart Formalism: Event Scheduling & Disposition. In: Proceedings OOPSLA' 1995. Austin, Texas 1995, S. 1 – 16.

[Allen 1998] *Allen, Paul; Frost, Stuart:* Component-Based Development for Enterprise Systems. Applying The SELECT Perspective. New York 1998.

[Allweyer 1998] *Allweyer, Thomas:* Adaptive Geschäftsprozesse. Rahmenkonzept und Informationssysteme. Dissertation, Wiesbaden 1999.

[Alt 2000] *Alt, Rainer; Zbornik, Stefan:* Elektronische Geschäftsabwicklung mit zwischenbetrieblichem Workflow. In: HMD. Praxis der Wirtschaftsinformatik, Heft 213, 2000, S. 89 – 103.

[Amberg 1993] *Amberg, Michael:* Konzeption eines Software-Architekturmodells für die objektorientierte Entwicklung betrieblicher Anwendungssysteme. Dissertation, Bamberg 1993.

[America 1991] *America, Pierre:* Designing an Object-Oriented Programming Language with Behavioral Subtyping. In: *de Bakker, J. W.; de Roever, W. P.; Rozenberg, G. (Hrsg.):* Foundations of Object-Oriented Languages. Berlin, Heidelberg, New York 1991, S. 60 – 90.

[Amstel 1987] *van Amstel, J. J.:* Programmieren: Die Entwicklung von Algorithmen in Pascal. Bonn 1987.

[Andersson 1995] *Andersson, Michael; Bergstrand, Johan:* Formalizing Use Cases with Message Sequence Charts. Master Thesis, Department of

Communication Systems at Lund Institute of Technology. Malmö, Mai 1995.

[Andrews 1991] *Andrews, Derek; Ince, Darrel:* Practical Formal Methods with VDM. New York, St. Louis, San Fancisco 1991.

[Andrews 1997] *Andrews, Derek:* A Theory and Practise of Program Development. London 1997.

[AndrewsD 1994] *Andrews, Dorine C.; Stalick, Susan K.:* Business Engineering. The Survival Guide. Englewood Cliffs, NJ, 1994.

[Arnold 1993] *Arnold, Robert S. (Hrsg.):* Software Reengineering. Washington 1993.

[Atkinson 1991] *Atkinson, Colin:* Object-Oriented Reuse, Concurrency and Distribution. An Ada-based Approach. Wokingham, England 1991.

[Atkinson 2000] *Atkinson, Colin; Kühne, Thomas; Henderson-Sellers, Brian:* To Meta or Not to Meta – That is the Question. In: Journal of Object-Oriented Programming, Dezember 2000, S. 32 – 35.

[Austin 1994] *Austin, John L.:* Zur Theorie der Sprechakte. 2. Auflage, Stuttgart 1994.

[Baetge 1974] *Baetge, Jörg:* Betriebswirtschaftliche Systemtheorie. Opladen 1974.

[Baklund 1995] *Baklund, Erik; Civello, Franco; Knag, Klaus; Mitchell, Richard:* Extending Interaction Diagrams. In: Report on Analysis & Design, Juli/August 1995, S. 24 – 33.

[Balzert 1996] *Balzert, Helmut:* Lehrbuch der Software-Technik. Software-Entwicklung. Heidelberg 1996.

[Balzert 1998] *Balzert, Helmut:* Lehrbuch der Software-Technik. Software-Qualitätssicherung, Software-Management, Unternehmensmodellierung. Heidelberg 1998.

[BalzertH 1995] *Balzert, Heide:* Methoden der objektorientierten Systemanalyse. Wien, Zürich 1995.

[BalzertH 1996] *Balzert, Heide:* Objektorientierte Systemanalyse. Konzepte, Methoden, Beispiele. Heidelberg, Berlin, Oxford 1996.

[BalzertH 1999] *Balzert, Heide:* Lehrbuch der Objektmodelierung. Analyse und Entwurf. Heidelberg, Berlin 1999.

[Bamberg 1985] *Bamberg, Günter; Coenenberg, Gerhard:* Betriebswirtschaftliche Entscheidungslehre. 4. Auflage. München 1985.

[Bamberg 1987] *Bamberg, Günter; Baur, Franz:* Statistik. 5. Auflage, München 1987.

[Banyard 1995] *Banyard, Philip; Cassells, Annette; Green, Patrick; u. a.:* Einführung in die Kognitionspsychologie. München 1995.

[Bauer 1981] *Bauer, F. L.:* Programming as Fulfillment of a Contract. In: *Henderson, P.:* System Design. Pergamon Infotech Ltd., Infotech State of the Art Report 9, ohne Ort, S. 165 – 174.

[Bea 1993] *Bea, Franz Xaver; Dichtl, Erwin; Schweitzer, Marcell; u. a. (Hrsg.):* Allgemeine Betriebswirtschaftslehre. Band 2: Führung. 6. Auflage, Stuttgart 1993.

[Bea 1994] *Bea, Franz Xaver; Dichtl, Erwin; Schweitzer, Marcell; u. a. (Hrsg.):* Allgemeine Betriebswirtschaftslehre. Band 3: Der Leistungsprozess. 6. Auflage, Stuttgart 1994.

[Bear 1990] *Bear, Stephen, Allen, Phillip, Coleman, Derek; Hayes, Fiona:* Graphical Specification of Object Oriented Systems. In: Proceedings ECOOP/OOPSLA 1990, 21.-25. Oktober 1990, S. 28 – 37.

[Becker 1995] *Becker, Jörg:* Strukturanalogien in Informationsmodellen. Ihre Definition, ihr Nutzen und ihr Einfluss auf die Bildung von Grundsätzen ordnungsgemäßer Modellierung (GoM). In: [König 1995], S. 133 – 150.

[Becker 1995b] *Becker, Jörg; Rosemann, Michael; Schütte, Reinhard:* Grundsätze ordnungsgemäßer Modellierung. In: Wirtschaftsinformatik, 1995, Band 37, Nr. 5, S. 435 – 445.

[Becker 1996] *Becker, Jörg; Vossen, Gottfried:* Geschäftsprozessmodellierung und Workflow-Management: Eine Einführung. In: [Vossen 1996], S. 17 – 26.

[Becker 1996b] *Becker, Jörg; Rosemann, Michael (Hrsg.):* Workflowmanagement – State-of-the-Art aus Sicht von Theorie und Praxis. Proceedings zum Workshop vom 10. April 1996. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Arbeitsbericht Nr. 47, Münster 1996.

[Becker 2000] *Becker, Jörg; Kugeler, Martin; Rosemann, Michael (Hrsg.):* Prozessmanagement. Ein Leitfaden zur prozessorientierten Organisationsgestaltung. Berlin, Heidelberg, New York 2000.

[Behringer 1997] *Behringer, Dorothea:* Modelling Global Behaviour with Scenarios in Object-Oriented Analysis. Dissertation Ecole Polytechnique Federale de Lausanne Nr. 1655. Lausanne 1997.

[Ben-Abdallah 1997] *Ben-Abdallah, Hanene; Leue, Stefan:* Expressing and Analyzing Timing Constraints in Message Sequence Chart Specifications. Technical Report 97 – 04, Department of Electrical and Computer Engineering, University of Waterloo, 1997.

[Ben-Abdallah 1997b] *Ben-Abdallah, Hanene; Leue, Stefan:* Timing Constraints in Message Sequence Chart Specifications. In: Formal Description Techniques 10. Proceedings of the Tenth International Conference on Formal Description Techniques FORTE/PSTV'97, Osaka, Japan, November 1997.

[Benjamin 1994] *Benjamin, Perakath C.; Menzel, Christopher P.; Mayer, Richard J.; Fillion, Florence; Futrell, Michael T.; deWitte, Paula S.; Lingineni, Madhavi:* IDEF5 Method Report. Armstrong Laboratory AL/HRGA. Wright-Patterson Air Force Base, Ohio, September 1994. (http://www.idef.com, Stand 16.10.2000)

[Berard 1995] *Berard, Edward V.:* Be Careful with „Use Cases“. Gaithersburg, Maryland 1995: The Object Agency.

[Berard 1993] *Berard, Edward V.:* Essays on Object-Oriented Software Engineering. Band 1. Englewood Cliffs 1993.

[Berthel 1992] *Berthel, Jürgen:* Informationsbedarf. In: [Frese 1992], Spalte 872 – 886.

[Bertram 1996] *Bertram, Martin:* Das Unternehmensmodell als Basis der Wiederverwendung bei der Geschäftsprozessmodellierung. In: [Vossen 1996], S. 81 – 100.

[Bertram 1997] *Bertram, Ulrich; Winkler, Klaus:* Fallabschließende Bearbeitung als Leitprinzip für das Management von Geschäftsprozessen in Versicherungsunternehmen. In: [Corsten 1997], S. 121 – 151.

[Berztiss 1996] *Berztiss, Alfs:* Software Methods für Business Reengineering. New York, Berlin, Heidelberg 1996.

[Bicarregui 1994] *Bicarregui, Juan C.; Fitzgerald, John S.; Lindsay, Peter A.; Moore, Richard; Ritchie, Brian:* Proof in VDM: A Practitioner's Guide. London, Berlin, Heidelberg 1994.

[Birkenbihl 1993] *Birkenbihl, Vera F.:* Kommunikation als Erfolgsfaktor. In: [Scharfenberg 1993], S. 477 – 487.

[Blaha 1992] *Blaha, Michael:* Models of Models. In: Journal of Object-Oriented Programming, September 1992, S. 13 – 18.

[Blair 1989] *Blair, Gordon S.; Galagher, John J.; Malik, Javad:* Genericity vs. Inheritance vs. Delegation vs. Conformance. In: Journal of Object-Oriented Programming, September/Oktober 1989, S. 11 – 17.

[Bleicher 1991] *Bleicher, Knut:* Organisation. Strategien – Strukturen – Kultur. 2., vollständig neu bearbeitete und erweiterte Auflage. Wiesbaden 1991.

[Bleicher 1993] *Bleicher, Knut:* Organisation. In: [Bea 1993], S. 103 – 186.

[Bobrow 1975] *Bobrow, Daniel G.; Collins, Allan (Hrsg.):* Representation and Understanding. Studies in Cognitive Science. New York, San Francisco, London 1975.

[Bock 1997] *Bock, Conrad; Odell, James:* A More Complete Model of Relations and their Implementation: Part I: Relations as Object Types. (http://www.intellicorp.com/ooieonline/relation1.html, Stand 01.06.2000) Ursprünglich erschienen in: Journal of Object-Oriented Programming, Band 10, Nr. 3, Juni 1997.

[Bock 1997b] *Bock, Conrad; Odell, James:* A More Complete Model of Relations and their Implementation: Part II: Mappings. (http//www.intellicorp.com/ooieonline/relation2.html, Stand 01.06.2000) Ursprünglich erschienen in: Journal of Object-Oriented Programming, Band 10, Nr. 6, Oktober 1997.

[Bock 1998] *Bock, Conrad; Odell, James:* A More Complete Model of Relations and their Implementation: Roles. In: Journal of Object-Oriented Programming, Mai 1998, S. 51 – 54.

[Bock 1998b] *Bock, Conrad; Odell, James:* A More Complete Model of Relations and their Implementation: Aggregation. In: Journal of Object-Oriented Programming, September 1998, S. 68 – 70, 85.

[Bock 1999] *Bock, Conrad:* Three Kinds of Behavior Models. In: Journal of Object-Oriented Programming, Juli/August 1999, S. 36 – 39.

[Bock 2000] *Bock; Conrad:* Goal-Driven Modeling. In: Journal of Object-Oriented Programming, September 2000, S. 48 – 50, 56.

[Bock 2001] *Bock; Conrad:* Goal-Driven Modeling, Part 2. In: Journal of Object-Oriented Programming, März 2001, S. 25 – 28.

[Bohr 1979] *Bohr, Kurt:* Produktionsfaktorsysteme. In: [Kern 1979], Spalte 1481 – 1493.

[Booch 1994] *Booch, Grady:* Objektorientierte Analyse und Design. Mit praktischen Anwendungsbeispielen. Bonn 1994.

[Booch 1995] *Booch, Grady:* Objektorientierte Analyse und Design. Mit praktischen Anwendungsbeispielen. 1. korrigierter Nachdruck, Bonn 1995.

[Booch 1995b] *Booch, Grady; Rumbaugh, James:* Unified Method for Object-Oriented Development. Documentation Set Version 0.8. Santa Clara, CA, 1995.

[Booch 1996] *Booch, Grady; Rumbaugh, James; Jacobson, Ivar:* The Unified Modeling Language for Object-Oriented Development. Documentation Set Version 0.9 Addendum. Santa Clara, CA, 1996.

[Booch 1999] *Booch, Grady; Rumbaugh, James; Jacobson, Ivar:* The Unified Modeling Language User Guide. Reading, MA, u. a. 1999.

[Borberg 2000] *Borberg, Bernd; Müller, Heiko; Kick, Ulrich:* Von der Geschäftsprozessanalyse zur UML – eine pragmatische Methodik. In: Objektspektrum, Heft 2, 2000, S. 68 – 72.

[Borgida 1984] *Borgida, Alexander; Mylopoulos, John; Wong, K. T.:* Generalization/Specialization as a Basis for Software Specification. In: [Brodie 1984], S. 87 – 114.

[Brachman 1977] *Brachman, Ronald J.:* What's in a concept: structural foundations for Semantic Networks. In: International Journal of Man-Machine Studies, Band 9, 1977, S. 127 – 152.

[Brachman 1979] *Brachman, Ronald J.:* On the Epistomological Status of Semantic Networks. In: [Findler 1979], S. 3 – 50.

[Brachman 1983] *Brachman, Ronald J.:* What IS-A is and isn't: Analysis of Taxonomic Links in Semantic Networks. In: IEEE Computer, Oktober 1983, S. 30 – 36.

[Brachman 1985] *Brachman, Ronald J.:* An Overview of the KL-ONE Knowledge Representation System. In: Cognitive Science, Band 9, 1985, S. 171 – 216.

[Brenner 1995] *Brenner, Walter; Keller, Gerhard (Hrsg.):* Business Reengineering mit Standardsoftware. Frankfurt a. M., New York 1995.

[Brenner 1995b] *Brenner, Walter; Hamm, W.:* Prinzipien des Business Reengineering. In: [Brenner 1995], S. 17 – 43.

[Breu 1998] *Breu, R.; Grosu, R.; Huber, F.; Rumpe, B.; Schwerin, W.:* Systems, Views and Models of UML. In: [Schader 1998], S. 93 – 108.

[Breutmann 1992] *Breutmann, Bernd; Burkhardt, Rainer:* Objektorientierte Systeme. Grundlagen – Werkzeuge – Einsatz. München 1992.

[Brockhaus 2000] *Bibliographisches Institut & F. A. Brockhaus AG:* Der Brockhaus Multimedial 2001 Premium. Mannheim 2000.

[Brodie 1984] *Brodie, M. L.; Mylopoulos, J.; Schmidt, J. W. (Hrsg.):* On Conceptual Modeling. New York 1984.

[Bruce 1972] *Bruce, Bertram C.:* A Model for Temporal References and Its Application in a Question Answering Program. In: Artificial Intelligence, Band 3, 1972, Seite 1 – 25.

[Brynjolfsson 1993] *Brynjolfsson, Erik:* The Productivity Paradox of Information Technology. In: Communications of the ACM, Band 26, Nr. 12, Dezember 1993, S. 67 – 77.

[Budd 1997] *Budd, Timothy:* An Introduction to Object-Oriented Programming. 2. Auflage, Reading, MA, 1997.

[Buhr 1996] *Buhr, R. J. A.; Casselman; R. S.:* Use Case Maps for Object-Oriented Systems. Upper Saddle River, NJ, 1996.

[Bullinger 1993] *Bullinger, Hans-Jörg; Niemeier, Joachim:* Methodik zur Organisationsplanung im Büro. In: [Scharfenberg 1993], S. 103 – 123.

[Bullinger 1996] *Bullinger, Hans-Jörg; Warnecke, Hans Jürgen (Hrsg.):* Neue Organisationsformen im Unternehmen. Ein Handbuch für das moderne Management. Berlin, Heidelberg, New York 1996.

[Bullinger 2001] *Bullinger, Hans-Jörg:* Enterprise Process Management – Innovative Unternehmensprozesse zielorientiert gestalten. Vortrag auf der [IAO 2001].

[Bullinger 2001b] *Bullinger, Hans-Jörg*; *Schreiner, Peter (Hrsg.):* Business Process Management Tools. Eine evaluierende Marktstudie über aktuelle Werkzeuge. Fraunhofer-Institut für Arbeitswissenschaft und Organisation (IAO). Stuttgart 2001.

[Bungert 1995] *Bungert, Winfried, Heß, Helge:* Objektorientierte Geschäftsprozessmodellierung. In: Information Management, Heft 1, 1995, S. 52 – 63.

[Burkhardt 1994] *Burkhardt, Rainer:* Modellierung dynamischer Aspekte mit dem Objekt-Prozess-Modell. Dissertation, Ilmenau 1994.

[Burkhardt 1997] *Burkhardt, Rainer:* UML – Unified Modeling Language. Objektorientierte Modellierung für die Praxis. Bonn 1997.

[Buschmann 1998] *Buschmann, Frank; Meunier, Regine; Rohnert, Hans:* Pattern-orientierte Software-Architektur. Ein Pattern-System. Bonn 1998.

[Carmichael 1994] *Carmichael, Andy (Hrsg.):* Object Development Methods. New York 1994.

[Cercone 1975] *Cercone, Nick:* Representing Natural Language in Extended Semantic Networks. Technical Report TR75 - 11, Department of Computing Science, The University of Alberta. Edmonton, Alberta, Canada 1975.

[Castellani 2000] *Castellani, Xavier; Habrias, Henri; Perrin, Phillipe:* A Synthesis on the Definitions and Notations of Cardinalities or Relationships. In: Journal of Object-Oriented Programming, 13. Jahrgang, Nr. 6, Oktober 2000, S. 32 – 35.

[Cercone 1975b] *Cercone, Nick; Schubert, Len:* Toward a State Based Conceptual Representation. In: Proceedings of the 4th International Joint Conference on Artificial Intelligence. Cambridge, MA, 1975.

[Chen 1976] *Chen, Peter Pin-Shan:* The Entity-Relationship Model – Toward a Unified View of Data. In: ACM Transactions on Database Systems, Band 1, Nr. 1, März 1976, S. 9 – 36.

[Chen 1994] *Chen, R.; Scheer, A.-W.:* Modellierung von Prozessketten mittels Petri-Netz-Theorie. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes. Heft 107, Saarbrücken 1994.

[Chonoles 1995] *Chonoles, Michael J.; Gilliam, Clinton C.:* Real-Time object-oriented System Design using the Object Modeling Technique (OMT). In: Journal of Object-Oriented Programming, Juni 1995, S. 16 – 24.

[Chonoles 1995b] *Chonoles, Michael J.; Schardt, James A.; Magrogan, Phil J.:* Bridging Business Processes and Object Technology. In: Report on Analysis & Design, Band 2, Nummer 3, 1995, S. 49 – 55.

[Coad 1991] *Coad, Peter; Yourdon, Edward:* Object-Oriented Analysis. 2nd Ed. Englewood Cliffs, NJ, 1991.

[Coad 1994] *Coad, Peter; Yourdon, Edward:* OOA. Objektorientierte Analyse. München 1994.

[Cockburn] *Cockburn, Alistair:* Structuring Use Cases with Goals. Human and Technology, 7691 Dell Road, Salt Lake City, UT, ohne Jahr.

[Coenenberg 1991] *Coenenberg, Adolf G.; Fischer, Thomas M.:* Prozesskostenrechnung - Strategiche Neuorientierung in der Kostenrechnung. In: DBW, Band 51, Heft 1, 1991, S. 21 – 38.

[Coleman 1992] *Coleman, Derek; Hayes, Fiona; Bear, Stephan:* Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design. In: IEEE Transactions on Software Engineering, Band 18, Nr. 1, Januar 1992, S. 9 – 18.

[Coleman 1994] *Coleman, Derek; Arnold, Patrick; Bodoff, Stephanie; Dollin, Chris; Gilchrist, Helena; Hyes, Fiona; Jeremes, Paul:* Object-Oriented Development. The Fusion Method. Englewood Cliffs, NJ, 1994.

[Constantine 1992] *Constantine, Larry L. (Panel); Jacobson, Ivar; Page-Jones, Meilir; Palmer, John; Weiss, Steven:* From Events to Objects: The Heresy of Event-Orientation in a World of Objects. In: ACM SIGPLAN Notices, Vol. 27, Nr. 10, Oktober 1992 (zgl. OOPSLA '1992 Conference Proceedings, 18. – 22. Oktober 1992, Vancouver, British Columbia, Kanada.) S. 295 – 297.

[Cook 1990] *Cook, William R.:* Object-Oriented Programming versus Abstract Data Types. In: Rozenberg, G.; Bakker, J. W. de: Foundations of Object-Oriented Languages. Berlin, Heidelberg, New York 1990, S. 151 – 178.

[Cook 1994] *Cook, Steve; Daniels, John:* Designing Object Systems. Object-Oriented Modelling with Syntropy. New York, London, Toronto 1994.

[Cook 1994b] *Cook, Steve; Daniels, John:* Object Communication. In: Journal of Object-Oriented Programming, September 1994, S. 14 – 23.

[Corsten 1986] *Corsten, Hans:* Zur Verkürzung der Durchlaufzeiten bei Büroarbeiten. In WiSu, Heft 8-9, 1986, S. 426 – 431.

[Corsten 1990] *Corsten, Hans:* Betriebswirtschaftslehre der Dienstleistungsunternehmen. 2., durchgesehene Auflage. München 1990.

[Corsten 1994] *Corsten, Hans (Hrsg.):* Betriebswirtschaftslehre. München 1994.

[Corsten 1997] *Corsten, Hans (Hrsg.):* Management von Geschäftsprozessen. Theoretische Ansätze – Praktische Beispiele. Stuttgart, Berlin, Köln 1997.

[Corsten 1997b] *Corsten, Hans:* Geschäftsprozessmanagement – Grundlagen, Elemente und Konzepte. In: [Corsten 1997], S. 9 – 57.

[Coulson-Thomas 1994] *Coulson-Thomas, Colin (Hrsg.):* Business Process Re-engineering: Myth & Reality. London 1994.

[Cube 1970] *Cube, Felix von:* Was ist Kybernetik? 3. Auflage, München 1970.

[Curtis 1992] *Curtis, Bill; Kellner, Marc I.; Over, Jim:* Process Modeling. In: Communications of the ACM, Band 35, Nr. 9, September 1992, S. 75 – 90.

[D'Souza 1998] *D'Souza, Desmond F.; Wills, Alan Cameron:* Objects, Components, and Frameworks with UML: The Catalysis Approach. Reading, MA, 1998. (Nicht publizierte Vorab-Fassung vom 20.7.1998)

[D'Souza 1999] *D'Souza, Desmond F.; Wills, Alan Cameron:* Objects, Components, and Frameworks with UML: The Catalysis Approach. Reading, MA, 1999.

[Daenzer 1994] *Daenzer, W. F.; Huber, F. (Hrsg.):* Systems Engineering. Methodik und Praxis. 8. verbesserte Auflage, Zürich 1994.

[Daniels 1970] *Daniels, Alan; Yeates, Donald; Erbach, Karl F.:* Gundlagen der Systemanalyse. Köln-Braunsfeld 1970.

[Date 1990] *Date, C. J.:* An Introduction to Database Systems. Band 1, 5. Auflage, Reading, MA, 1990.

[Davenport 1993] *Davenport, Thomas H.; Short, James E.:* The New Industrial Engineering: Information Technology and Business Process Redesign. In: [Arnold 1993], S. 83 – 99. (Ursprünglich in: Sloan Management Review, Band 31, 1990, S. 11 – 27.)

[Davenport 1993b] *Davenport, Thomas H.:* Process Innovation. Reengineering Work through Information Technology. Boston, MA, 1993.

[Davis 1991] *Davis, Tim R. V.:* Internal Service Operations: Strategies for Increasing their Effectiveness and Controlling their Costs. In: Organizational Dynamics, Autumn 1991, S. 5 – 22.

[Davis 1993] *Davis, Alan M.:* Software Requirements. Objects, Functions, and States. Revision, Englewood Cliffs, NJ, 1993.

[Dawes 1991] *Dawes, John:* The VDM-SL Reference Guide. London 1991.

[de Champeaux 1993] *de Champeaux, Dennis; Lea, Douglas; Faure, Penelope:* Object-Oriented System Development. Reading, MA, 1993.

[Decker 1996] *Decker, Stefan; Erdmann, Michael:* A Unifying View on Business Process Modelling and Knowledge Engineering. Institut für angewandte Informatik und formale Beschreibungsverfahren der Universität Karlsruhe (TH). Bericht 344, Oktober 1996.

[Delnef 1998] *Delnef, Alexander:* Ein Evaluationsraster für methodische Ansätze zur Geschäftsprozessgestaltung. Dissertation, Regensburg 1998.

[Denert 1991] *Denert, Wofgang:* Software-Engineering. Methodische Projektabwicklung. 1. korrigierter Nachdruck. Berlin, Heidelberg, New York u. a. 1991.

[Denning 1992] *Denning, Peter J.:* Work is a Closed Loop Process. In: American Scientist, Band 80, Nr. 4, Juli/August 1992, S. 314 – 317.

[Dernbach 1993] *Dernbach, Wolfgang:* Organisation und Wettbewerbsfähigkeit. In: [Scharfenberg 1993], S. 125 – 159.

[Desfray 1994] *Desfray, Philippe:* Object Engineering. The Fourth Dimension. Wokingham, England, u. a. 1994.

[Diaz 1999] *Diaz, I.; Matteo, A.:* Objectory Process Stereotypes. In: Journal of Object-Oriented Programming, 12. Jahrgang, Juni 1999, S. 29 – 38.

[Dinkhoff 1996] *Dinkhoff, Guido:* Entwicklung Workflow-Management-geeigneter Software-Systeme. In: [Vossen 1996], S. 405 – 421.

[Dischinger 1995] *Dischinger, Guido:* Objektorientierter Fachentwurf: Zur Eignung objektorientierter Ansätze für das fachliche Entwerfen von Anwendungssoftware. Dissertation, Wiesbaden 1995.

[Douglas 1999] *Douglas, Bruce Powell:* Doing Hard Time. Developing Real-Time Systems with UML, Object, Frameworks, and Patterns. Reading, MA, 1999.

[Douglas 2000] *Douglas, Bruce Powell:* Real-Time UML. Second Edition. Developing Efficient Objects for Embedded Systems. Reading, MA, 2000.

[Dröschel 1998] *Dröschel, Wolgang; Heuser, Walter; Midderhoff, Rainer (Hrsg.):* Inkrementelle und objektorientierte Vorgehensweise mit dem V-Modell 97. München, Wien 1998.

[Duden 1993] *Wissenschaftlicher Rat und Mitarbeiter der Dudenredaktion (Hrsg.):* Duden »Das große Wörterbuch der deutschen Sprache«: in acht Bänden. Mannheim, Leipzig, Wien, Zürich. 2., völlig neu bearbeitete und erweiterte Auflage 1993.

[Duden 2000] *Dudenredaktion:* Duden. Die deutsche Rechtschreibung. 22., völlig neu bearbeitete und erweiterte Auflage. Mannheim, Leipzig, Wien, Zürich 2000.

[Dürr 1995] *Dürr, E. H.; Plat, N. (Hrsg.):* VDM++ Language Reference Manual. ESPRIT 6500 Afrodite. Doc. ID. AFRO/CG/ED/LRM/V11, Rijksuniversiteit Utrecht WP-No. LT-A1, Utrecht 1995.

[Eckert 1995] *Eckert, Gabriel; Kempe, Magnus:* Modeling with Objects and Values. Issues and Perspectives. In: Report on Object Analysis & Design, Band 1, Nr. 5, Januar/Februar, 1995, S. 20 – 26.

[Ehrich 1989] *Ehrich, Hans-Dieter; Gogolla, Martin; Lipeck, Walter:* Algebraische Spezifikation abstrakter Datentypen. Stuttgart 1989.

[Eichhorn 1979] *Eichhorn, Wolfgang:* Die Begriffe Modell und Theorie in der Wirtschaftswissenschaft. In: [Raffée 1979], S. 60 – 104.

[Elgass 1993] *Elgass, Petra; Krcmar, Helmut:* Computergestützte Geschäftsprozessplanung. In: Information Management. Heft 1, 1993, S. 42 – 49.

[Elgass 1996] *Elgass, Petra; Krcmar, Helmut; Oberweis, Andreas:* Von der informalen zur formalen Geschäftsprozessmodellierung. In: [Vossen 1996], S. 125 – 139.

[Ellsberger 1997] *Ellsberger, Jan; Hogrefe, Dieter; Saram, Amardeo:* SDL. Formal Object-oriented Language for Communicating Systems. Hemel Hempstead 1997.

[Embley 1992] *Embley, David W.; Kurtz, Barry D.; Woodfield, Scott N.:* Object-Oriented Systems Analysis. A Model-Driven Approach. Englewood Cliff, NJ, 1992.

[Engelmann 1995] *Engelmann, Thomas:* Business Process Reengineering. Grundlagen – Gestaltungsempfehlungen – Vorgehensmodell. Wiesbaden 1995.

[Englmeier 1997] *Englmeier, Gisbert:* Objektstrukturen – praxisbezogenes Konzept der Attribut-/Methodenvererbung. Berlin 1997.

[Erdl 1992] *Erdl, Günter; Schönecker, Horst G.:* Geschäftprozessmanagement. Vorgangssteuerungssysteme und integrierte Vorgangsbearbeitung. Baden-Baden 1992.

[Erichson 1993] *Erichson, Bernd; Hammann, Peter:* Beschaffung und Aufbereitung von Informationen. In: [Bea 1993], S. 187 – 225.

[Eriksson 2000] *Eriksson, Hans-Erik; Penker, Magnus:* Business Modeling with UML. Business Patterns at Work. New York, Chichester, Weinheim 2000.

[Eversheim 1995] *Eversheim, Walter:* Prozessorientierte Unternehmensorganisation. Berlin, Heidelberg, NewYork 1995.

[Fahrwinkel 1995] *Fahrwinkel, Uta:* Methode zur Modellierung und Analyse von Geschäftsprozessen zur Unterstützung des Business Process Reengineering. Dissertation, Paderborn 1995.

[Fensel 1994] *Fensel, Dieter:* Über den Sinn formaler Spezifikationssprachen. Institut für angewandte Informatik und formale Beschreibungsverfahren der Universität Karlsruhe (TH). Bericht 293, Karslruhe 1994.

[Ferber 1999] *Ferber, Rafael:* Philosophische Grundbegriffe. Eine Einführung. München 1999.

[Ferstl 1991] *Ferstl, Otto K.; Sinz, Elmar J.:* Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik, Band 33, Nr. 6, 1991, S. 477 – 491.

[Ferstl 1993] *Ferstl, Otto K.; Sinz, Elmar J.:* Geschäftsprozessmodellierung. In: Wirtschaftsinformatik, Band 35, Nr. 6, 1993, S. 589 – 592.

[Ferstl 1993b] *Ferstl, Otto K., Sinz Elmar J.:* Grundlagen der Wirtschaftsinformatik. Band 1. München 1993.

[Ferstl 1994] *Ferstl, Otto K.; Sinz, Elmar J.:* Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. Bamberger Beiträge zur Wirtschaftsinformatik Nr. 21, Universität Bamberg, 1994.

[Ferstl 1994b] *Ferstl, Otto K.; Sinz, Elmar J.:* Tool-Based Business Process Modeling using the SOM Approach. Bamberger Beiträge zur Wirtschaftsinformatik Nr. 19, Universität Bamberg, 1994.

[Ferstl 1994c] *Ferstl, Otto K.; Sinz, Elmar J.:* From Business Process Modeling to the Specification of Distributed Business Application Systems – An Object-Oriented Approach. Bamberger Beiträge zur Wirtschaftsinformatik Nr. 20, Universität Bamberg, 1994.

[Ferstl 1995] *Ferstl, Otto K.; Sinz, Elmar J.:* Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: Wirtschaftsinformatik, Band 37, Nr. 3, 1995, S. 209 – 220.

[Ferstl 1995b] *Ferstl, Otto K.; Mannmeusel, Thomas:* Gestaltung industrieller Geschäftsprozesse. In: Wirtschaftsinformatik, Band 37, Nr. 5, 1995, S. 446 – 458.

[Ferstl 1996] *Ferstl, Otto K., Sinz ElmarJ.:* Geschäftsprozessmodellierung im Rahmen des Semantischen Objektmodellls. In: [Vossen 1996], S. 47 – 61.

[Ferstl 1997] *Ferstl, Otto K., Sinz Elmar J.:* Modeling of Business Systems Using the Semantic Object Model (SOM) - A Methodological Framework. Bamberger Beiträge zur Wirtschaftsinformatik Nr. 43, Universität Bamberg, 1997.

[Ferstl 1998] *Ferstl, Otto K., Sinz Elmar J.:* Grundlagen der Wirtschaftsinformatik. Band 1. 3. völlig überarbeitete und erweiterte Auflage. München 1998.

[Findler 1971] *Findler, Nicholas V.; Chen, David:* On the Problem of Time, retrieval of Temporal Relations, Causality, and Co-Existance. In: *Cooper, D. C. (Hrsg.):* Proceedings of the 2nd International Joint Conference on Artificial Intelligence, London, UK, September 1971, Seite 531 – 545.

[Findler 1979] *Findler, Nicholas V. (Hrsg.):* Associative Networks. Representation and Use of Knowledge by Computers. New York, San Francisco, London 1979.

[Firesmith 1993] *Firesmith, Donald G.:* An expanded View of Messages. In: Journal of Object-Oriented Programming, Juli/August 1993, S. 51 – 52.

[Firesmith 1994] *Firesmith, Donald G.:* Modeling the Dynamic Behavior of Systems, Mechanisms, and Classes. In: Journal of Object-Oriented Programming, Juli/August 1994, S. 32 – 36, 47.

[Firesmith 1995] *Firesmith, Donald G.; Eykholt, Edward M.:* Dictionary of Object Technology. The Definitive Desk Reference. New York u. a. 1995.

[Firesmith 1995b] *Firesmith, Donald G.:* Use Cases: the Pros and Cons. Knowledge Systems Corporation. 4001 Western Parkway. Cary, North Carolina 27513. In: Report on Analysis & Design, Juli/August 1995, S. 2 – 6.

[Firesmith 1996] *Firesmith, Donald:* The Inheritance of State Models. In: Report on Analysis & Design, März/April 1996, S. 13 – 15.

[Firesmith 1997] *Firesmith, Donald; Henderson-Sellers, Brian; Graham, Ian:* OPEN Modeling Language (OML) Reference Manual. New York u. a. 1997.

[Firesmith 1998] *Firesmith, Donald; Henderson-Sellers:* Clarifying specialized forms of association in UML and OML. In: Report on Analysis & Design/ Journal of Object-Oriented Programming, Band 11, Nr. 2, Mai 1998. (Hier zitiert nach http://www.markv.com/OPEN, Stand 1.10.1998)

[Firesmith 1998b] *Firesmith, Donald G.; Henderson-Sellers, Brian:* Upgrading OML to Version 1.1. Part 1. Referential Relationships. In: Report on Analysis & Design/Journal of Object-Oriented Programming, Band 11, Nummer 3, Juni 1998. (Hier zitiert nach http://www.markv.com/OPEN, Stand 1.10.1998)

[Firesmith 1998c] *Firesmith, Donald G.; Henderson-Sellers, Brian:* Upgrading OML to Version 1.1. Part 2. Additional Concepts and Notation. In: Report on Analysis & Design/Journal of Object-Oriented Programming, September 1998, S. 61 – 67.

[Firesmith 1998d] *Firesmith, Donald G.; Krutsch, Scott A.; Stowe, Marshall; Hendely, Greg:* Documenting a complete Java application using OPEN. Harlow, England, Reading, MA, u. a. 1998.

[Fischermanns 1997] *Fischermanns, Guido; Liebelt, Wolfgang:* Grundlagen der Prozessorganisation. Schriftenreihe “Der Organisator” Band 9, 4. überarbeitete und erweiterte Auflage, Gießen 1997.

[Floyd 1996] *Floyd, Robert W.; Beigel Richard:* Die Sprache der Maschinen. Bonn 1996.

[Floyd 1998] *Floyd, Christiane:* Modellierung - ein Handgriff zur Wirklichkeit. In: [Pohl 1998], S. 21 – 26.

[Foegen 2001] *Foegen, Malte; Battenfeld, Jörg:* Die Rolle der Architektur in der Anwendungsentwicklung. In: Informatik-Spektrum, Band 24, Heft 5, 2001, S. 290 – 301.

[Fowler 1998] *Fowler, Martin; Scott, Kendolf:* UML konzentriert. Die neue Standard-Objektmodellierungssprache anwenden. Bonn 1998.

[Fowler 1999] *Fowler, Martin:* Analysemuster. Wiederverwendbare Objektmodelle. Bonn 1999.

[Fowler 2000] *Fowler, Danielle; Henderson-Sellers, Brian; Younessi, Houman:* Business Engineering: Is Process the Key?. In: Journal of Object-Oriented Programming, November 2000, S. 37 – 42.

[Frank 1998d] *Frank, Ulrich:* Object Oriented Modelling Languages: State of the Art and Open Research Questions. In: [Schader 1998], S. 14 – 31.

[Frank 1994] *Frank, Ulrich:* Multiperspektivische Unternehmensmodellierung. Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung. München, Wien 1994.

[Frank 1995] *Frank, Ulrich:* MEMO: Ein werkzeuggestützte Methode zum integrierten Entwurf von Geschäftsprozessen und Informationssystemen. In: [König 1995], S. 67 – 81.

[Frank 1995b] *Frank, Ulrich:* MEMO - eine Methode zur objektorientierten Unternehmensmodellierung. In: Objektspektrum, Heft 6, 1995, S. 43 – 47.

[Frank 1996] *Frank, Ulrich:* Delegation: Eine sinnvolle Ergänzung gängiger objektorientierter Modellierungskonzepte. In: [MobIS 1996], S. 16 – 19.

[Frank 1997] *Frank, Ulrich:* Zur Standardisierung objektorientierter Modellierungssprachen: Eine kritische Betrachtung des State of the Art am Beispeil der Unified Modeling Language. In: [MobIS 1997], S. 1- 5.

[Frank 1997b] *Frank, Ulrich:* Enriching Object-Oriented Methods with Domain Specific Knowledge. Outline of a Method for Enterprise Modeling. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 4, Juli 1997.

[Frank 1997c] *Frank, Ulrich; Halter, Sören:* Enhancing Object-Oriented Software Development with Delegation. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 2, Januar 1997.

[Frank 1997d] *Frank, Ulrich; Prasse, Michael:* Ein Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen - veranschaulicht am Beispiel von OML und UML. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 6, September 1997.

[Frank 1997e] *Frank, Ulrich:* Toward a Standardization of Object-Oriented Modeling Languages. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 3, Mai 1997.

[Frank 1997f] *Frank, Ulrich:* Möglichkeiten und Grenzen einer objektorientierten Modellierung. In: Tagungsband der STJIA'97. Erfurt 1997, S. 96 – 102. (http://www.uni-koblenz.de/~iwi/, Stand 17.04.2002)

[Frank 1998] *Frank, Ulrich:* The MEMO-Metamodell. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 9, Juni 1998.

[Frank 1998b] *Frank, Ulrich:* The MEMO Object Modeling Language (MEMO-OML). Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 10, Juni 1998.

[Frank 1998c] *Frank, Ulrich:* Applying the MEMO-OML: Guidelines and Examples. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 11, Juli 1998.

[Frank 1999] *Frank, Ulrich:* Eine Architektur zur Spezifikation von Sprachen und Werkzeugen für die Unternehmensmodellierung. In: Proceedings der MobIS-Fachtagung 1999, Bamberg 1999. Zugleich: Rundbrief der GI-Fachgruppe 5.10, 6. Jahrgang, Heft 1, Oktober 1999, S. 154 – 169.

[Frank 1999b] *Frank, Ulrich:* MEMO. Visual Languages for Enterprise Modeling. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz, Nr. 18, Juni 1999.

[Franz 1994] *Franz, Stefan:* Informations-Management als Basis für Prozess-Management. In: [Gaitanides 1994], S. 226 – 244.

[Frese 1992] *Frese, Erich (Hrsg.):* Handwörterbuch der Organisation. 3., völlig neu gestaltete Auflage. Enzyklopädie der Betriebswirtschaftslehre Band 2. Stuttgart 1992.

[Frick 1995] *Frick, Andrea:* Der Software-Entwicklungsprozess. Ganzheitliche Sicht. Grundlagen zu Entwicklungs-Prozess-Methoden. München 1995.

[Friederichs 1994] *Friederichs, Johann:* Die Rolle der Informatik bei der strategischen Neuausrichtung eines Unternehmens. In: HMD. Praxis der Wirtschaftsinformatik, Nr. 180, 1994, S. 71 – 81.

[Gaitanides 1983] *Gaitanides, Michael:* Prozessorganisation. München 1983.

[Gaitanides 1992] *Gaitanides, Michael:* Ablauforganisation. In: [Frese 1992], Spalte 1 – 18.

[Gaitanides 1994] *Gaitanides, Michael; Scholz, Rainer; Vrohlings, Alwin; Raster, Max (Hrsg.):* Prozess-Management. München 1994.

[Gaitanides 1994b] *Gaitanides, Michael; Scholz, Rainer; Vrohlings, Alwin:* Grundlagen und Zielsetzungen. In: [Gaitanides 1994], S. 1 – 19.

[Galler 1997] *Galler, Jürgen:* Vom Geschäftsprozessmodell zum Workflow-Modell. Dissertation, Wiesbaden 1997.

[Gallus 1979] *Gallus, Gerald:* Betriebsmittel, Begriff und Arten. In: [Kern 1979], Spalte 354 – 361.

[Gamma 1995] *Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John:* Design Patterns. Elements of Reusable Object-Oriented Software. Reading, MA, 1995.

[Gausemeier 1995] *Gausemeier, Jürgen; Flink, Alexander; Schlake, Oliver*. Szenario – Management. Planen und Führen mit Szenarien. München, Wien 1995.

[Gebert 1992] *Gebert, Diether:* Kommunikation. In: [Frese 1992], Spalte 1110 – 1121.

[Gehring 1998] *Gehring, Herrmann:* Betriebliche Anwendungssysteme. Kurseinheiten 1 und 2. Kurs 0825 aus 10/98 der Fernuniversität. Hagen 1998.

[Gerstenmeier 1995] *Gerstenmeier, Jochen (Hrsg.):* Einführung in die Kognitionspsychologie. Basel 1995.

[Gibson 1990] G*ibson, Elizabeth:* Objects – Born and Bread. In: Byte, Oktober 1990, S. 245 – 254.

[Gillibrand 2000] *Gilibrand, David:* Essential Business Object Design. In: Communications of the ACM, Band 43, Nr. 2, 2000, S. 117 – 119.

[Ginbayashi 1992] *Ginbayashi, Jun:* Analysis of Business Processes Specified in Z against an E-R Data Model. Technical Monograph PRG-103. Oxford University Computing Laboratory Programming Research Group. Oxford, Dezember 1992.

[Girth 1994] *Girth, Werner:* Methoden und Techniken für Prozessanalysen und Redesign. In: [Krickl 1994], S. 139 – 156.

[Glaser 1992] *Glaser, Horst:* Prozesskostenrechnung – Darstellung und Kritik. In: Zfbf, Band 44, Nr. 3, 1992, S. 275 – 288.

[Goodenough 1975] *Goodenough, John B.:* Exception Handling: Issues and a Proposed Notation. In: Communications of the ACM, Band 18, Nummer 12, Dezember 1975, S. 683 – 696.

[Gottlob 1996] *Gottlob, Georg; Schrefl, Michael; Röck, Brigitte:* Extending Object-Oriented Systems with Roles. In: ACM Transaction on Information Systems, Band 14, Nr. 3, Juli 1996, S. 268 – 296.

[Götze 1991] *Götze, Uwe:* Szenario-Technik in der strategischen Planung. Wiesbaden 1991.

[Götzer 1995] *Götzer, Klaus Georg:* Workflow: Unternehmenserfolg durch effizientere Arbeitsabläufe; Technik, Einsatz, Fallstudien. München 1995.

[Grabowski 1993] *Grabowski, Jens; Graubmann, Peter; Rudolph, Ekkart:* The Standardization of Message Sequence Charts. In: Proceedings of the IEEE Software Engineering Standards Symposium 1993.

[Graham 1991] *Graham, Ian:* Object-Oriented Methods. Wokingham, England, 1991.

[Graham 1997] *Graham, Ian; Henderson-Sellers, Brian; Younessi, Houman:* The OPEN Process Specification. Harlow, England, u. a. 1997.

[Graham 1998] *Graham, Ian:* Requirements Engineering and Rapid Development. Harlow, England 1998.

[Grob 1995] *Grob, Heinz Lothar; Volck, Stefan:* Abbildung von Geschäftsprozessen mit ereignisgesteuerten Prozessketten. In: WiSt, Heft 11, November 1995, S. 604 – 608.

[Grochla 1974] *Grochla, Erwin; und andere:* Integrierte Gesamtmodelle der Datenverarbeitung. Entwicklung und Anwendung des Kölner Integrationsmodells (KIM). München, Wien 1974.

[Grochla 1975] *Grochla, Erwin (Hrsg.):* Organisationstheorie. 1. und 2. Teilband, Stuttgart 1975.

[Grochla 1975b] *Grochla, Erwin; Wittmann, Waldemar (Hrsg.):* Handwörterbuch der Betriebswirtschaft. Vierte, völlig neu gestaltete Auflage. Stuttgart 1975.

[Grochla 1975c] *Grochla, Erwin:* Betriebliche Planung und Informationssysteme. Entwicklung und aktuelle Ansätze. Reinbek 1975.

[Grochla 1975d] *Grochla, Erwin; Kubicek, Herbert:* Einführung in die informationssystem-orientierten Ansätze. In: [Grochla 1975], S. 428 – 444.

[Grochla 1978] *Grochla, Erwin:* Einführung in die Organisationstheorie. Stuttgart 1978.

[Grochla 1982] *Grochla, Erwin:* Grundlagen der organisatorischen Gestaltung. Stuttgart 1982.

[GUIDE 1997] *GUIDE International (Hrsg.):* GUIDE Business Rule Project. Final Report, Revision 1.2, Oktober 1997, Chicago 1997.

[Gutenberg 1973] *Gutenberg, Erich:* Grundlagen der Betriebswirtschaft. Band 1: Die Produktion. 20. Auflage, Berlin, Heidelberg, New York 1973.

[Guttag 1977] *Guttag; John V.:* Abstract Data Types and the Development of Data Structures. In: Communications of the ACM, Band 20, Nr. 6, Juni 1977, S. 396 – 404.

[Guttag 1978] *Guttag; John V.; Horning, Jim J.:* The Algebraic Specification of Abstract Data Types. In: Acta Informatica, Band 10, 1978, S. 27 – 52.

[Grünwald 1991] *Grünwald, Michael:* Die Anfänge der abendländischen Philosophie. München 1991.

[Gruhn 1996] *Gruhn, Volker:* Geschäftsprozess-Management als Grundlage der Software-Entwicklung. In: Informatik Forschung und Entwicklung, Heft 11, 1996, S. 94 – 101.

[Györkös 1995] *Györkös, Jozsef; Krisper, Marjan; Mayr, Heinrich C. (Hrsg.):* Re-Technologies für Information Systems. Conference Proceedings ReTIS'95. Wien, München 1995.

[Hahn 1985] *Hahn, Dietger:* Planungs- und Kontrollrechnung. 3., vollständig überarbeitete Auflage, Wiesbaden 1985.

[Haist 1989] *Haist, Fritz; Hansjörg Fromm:* Qualität im Unternehmen. Prinzipien – Methoden – Techniken. München 1989.

[Hall 1990] *Hall, Anthony:* Seven Myths of Formal Methods. In: IEEE Software, September 1990, S. 11 – 19.

[Hamel 1992] *Hamel, Winfried:* Zielsysteme. In: [Frese 1992], Spalte 2634 – 2652.

[Hammer 1993] *Hammer, Michael; Champy, James:* Reengineering the Corporation. New York 1993.

[Hammer 1993b] *Hammer, Michael:* Reengineering Work: Don't Automate, Obliterate. In: [Arnold 1993], S. 74 – 82. (Ursprünglich in Harvard Business Review, 68. Jahrgang, Heft 4, 1990, S. 104 – 112)

[Hammer 1994] *Hammer, Michael; Champy, James:* Business Reengineering. Die Radikalkur für das Unternehmen. 4. Auflage, Frankfurt am Main, New York 1994.

[Hanfland 1993] *Hanfland, Ulrich:* Der Organisator als Fachpromotor, Moderator, Innovator. In: [Scharfenberg 1993], S. 265 – 292.

[Hansen 2001] *Hansen, Hans Robert:* Wirtschaftsinformatik. Band 1, 8. völlig neubearbeitete und erweiterte Auflage, Stuttgart 2001.

[Harel 1985] *Harel, David; Pnueli, Amir:* On the Development of Reactive Systems. In: *Krzystaf, R. Apt:* Logics and Models of Concurrent Systems. Berlin, Heidelberg, New York 1985, S. 477 – 498.

[Harel 1987] *Harel, David:* Statecharts: A Visual Formalism for complex Systems. In: Science of Computer Programming, Band 8, 1987, S. 231 – 274.

[Harel 1987b] *Harel, David; Pnueli, Amir; Schmidt, Jeanette P.; Sherman, R.:* On the Formal Semantics of Statecharts. In: Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, 22. – 25. Juni 1987, S. 54 – 64.

[Harel 1988] *Harel, David:* On visual Formalisms. In: Communications of the ACM, Band 31, Nr. 5, Mai 1988, S. 514 – 530.

[Harel 1998] *Harel, David:* Modeling Reactive Systems with Statecharts. The Statemate Approach. New York, San Fracisco u. a. 1998.

[Harrington 1991] *Harrington, H. James:* Business Process Improvement. New York 1991.

[Harrison 1993] *Harrison, H. J.; Pratt, M. D.:* A Methodology for Reengineering Businesses. In: Planning Review, März/April 1993, S. 6 – 11.

[Hars 1993] *Hars, Alexander; Zimmermann, V.; Scheer, A.-W.:* Entwicklungslinien für die computergestützte Modellierung von Aufbau- und Ablauforganisation. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes. Heft 105, Saarbrücken 1993.

[Hars 1994] *Hars, Alexander:* Referenzdatenmodelle: Grundlagen effizienter Datenmodellierung. Dissertation, Wiesbaden 1994.

[Hasenkamp 1994] *Hasenkamp, Ulrich (Hrsg.):* Einführung von CSCW-Systemen in Organisationen. Tagungsband der D-CSCW ’94. Braunschweig, Wiesbaden 1994.

[Hasekamp 1994b] *Hasenkamp, Ulrich; Kirn, Stefan; Sysring, Michael (Hrsg.):* CSCW. Computer Supported Cooperative Work. Bonn 1994.

[Hatley 1993] *Hatley, Derek J.; Pribhai, Imtiaz A.:* Strategien der Echtzeit-Programmierung. München 1993.

[Heeg] *Heeg, Georg; Bücker, Matthias:* Object Behavior Analysis – Von der Suche nach Objekten. Dortmund, ohne Jahr.

[Heilmann 1994] *Heilmann, Heidi:* Workflow Management: Integration von Organisation und Informationsverarbeitung. In: HMD. Praxis der Wirtschaftsinformatik, Heft 176, 1994, S. 8 – 21.

[HeilmannM 1997] *Heilmann, Matthias L.:* Geschäftsprozess-Controlling. Bern, Stuttgart, Wien 1997.

[Heinen 1971] *Heinen, Edmund:* Grundlagen betriebswirtschaftlicher Entscheidungen. Das Zielsystem der Unternehmung. Wiesbaden 1971.

[Heinen 1985] *Heinen, Edmund:* Einführung in die Betriebswirtschaftslehre. 9., verbesserte Auflage, Wiesbaden 1985.

[Heinen 1991] *Heinen, Edmund (Hrsg.):* Industriebetriebslehre. Entscheidungen im Industriebetrieb. 9., vollst. neu bearbeitete und erweiterte Aufllage. Wiesbaden 1991.

[Heinen 1991b] *Heinen, Edmund:* Industriebetriebslehre als entscheidungsorientierte Unternehmensführung. In: [Heinen 1991], S. 1 – 71.

[Heinen 1991c] *Heinen, Edmund; Dietel, Bernhard:* Kostenrechnung. In: [Heinen 1991], S. 1157 – 1314.

[Heinrich 1990] *Heinrich, Lutz J.; Burgholzer, Peter:* Systemplanung. Planung und Realisierung von Informations- und Kommunikationssystemen. Band 1. Der Prozess der Systemplanung, der Vorstudie und der Feinstudie. München 1991.

[Heinrich 1994] *Heinrich, Lutz J.:* Systemplanung. Planung und Realisierung von Informatik-Projekten. Band 2. Der Prozess der Grobprojektierung, der Feinprojektierung und der Installierung. München 1994.

[Helbig 1996] *Helbig, Hermann:* Künstliche Intelligenz und automatische Wissensverarbeitung. 2., stark bearbeitete Auflage. Bonn 1996.

[Henderson-Sellers 1994] *Henderson-Sellers, Brian; Edward, Julian:* Book Two of Object-Oriented Knowledge. The Working Object. Sydney 1994.

[Henderson-Sellers 1997] *Henderson-Sellers, Brian:* OPEN Relationships – Compositions and Containments. In: Report on Analysis & Design/Journal of Object-Oriented Programming, Band 10, Part 7, S. 51 – 55, 72, November/Dezember 1997.

[Henderson-Sellers 1998] *Henderson-Sellers, Brian; Simons, Anthony; Younessi, Houman:* The OPEN Toolbox of Techniques. Harlow, England, u. a. 1998.

[Henderson-Sellers 1998b] *Henderson-Sellers, Brian:* OPEN Relationships – Associations, Mappings, Dependencies and Uses. In: Report on Analysis & Design/Journal of Object-Oriented Programming, Band 10, Part 9, S. 49 – 57, Mai 1998. (Hier zitiert nach http://www.markv.com/OPEN)

[Henderson-Sellers 1998c] *Henderson-Sellers, Brian:* OO Diagram Connectivity. In: Journal of Object-Oriented Programming, November/Dezember 1998, S. 60 – 68.

[Henderson-Sellers 1999] *Henderson-Sellers, Brian; Graham, Ian:* Metalevel Relationship Cardinalities. In: Journal of Object-Oriented Programming, März/April 1999, S. 51 – 58.

[Henderson-Sellers 2000] *Henderson-Sellers, Brian; Bhuvan, Unhelkar:* OPEN Modeling with UML. Harlow, England, 2000.

[Hendrix 1979] *Hendrix, G. G.:* Encoding Knowledge in Partitioned Networks. In: [Findler 1979], S. 51 – 92.

[Hering 1995] *Hering, Ekbert; Gutekunst, Jürgen; Dyllong, Ulrich:* Informatik für Ingenieure. Düsseldorf 1995.

[Herrmann 1998] *Herrmann, Thomas; Scheer, A.-W.; Weber, H.:* Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Managment-Systemen 1. Von der Erhebung zum Sollkonzept. Heidelberg 1998.

[Herrmann 1998b] *Herrmann, Thomas; Scheer, A.-W.; Weber, H.:* Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Managment-Systemen 2. Von der Sollkonzeptentwicklung zur Implementierung von Workflow-Managementanwendungen. Heidelberg 1998.

[Hess 1995] *Hess, Thomas, Brecht; Leo; Österle, Hubert:* Stand und Defizite der Methoden des Business Process Reengineering. In: Wirtschaftsinformatik, 35. Jahrgang, Nr. 5, 1995, S. 480 – 486.

[Hess 1996] *Hess, Thomas, Brecht; Leo*. State of the Art des Business Process Redesign. Wiesbaden 1996.

[Heuer 1995] *Heuer, Andreas; Saake, Gunter:* Datenbanken. Konzepte und Sprachen. Bonn 1995.

[Hewitt 1977] *Hewitt, Carl:* Viewing Control Structures as Patterns of Passing Messages. In: Artificial Intelligence, Band 8, 1977, S. 323 – 364.

[Hewitt 1986] *Hewitt, Carl:* Offices are Open Systems. In: ACM Transactions on Office Information Systems, Band 4, Nr. 3, Juli 1986, S. 271 – 287.

[Hill 1994] *Hill, Wilhelm; Fehlbaum, Raymond; Ulrich, Peter:* Organisationslehre 1. 5. Auflage, Bern 1994.

[Hillier 1988] *Hillier, Frederick S.; Liebermann, Gerald J.:* Operations Research. 4. Auflage, München 1988.

[Hinterhuber 1994] *Hinterhuber, Hans H.:* Paradigmenwechsel: Vom Denken in Funktionen zum Denken in Prozessen. In: Journal für Betriebswirtschaft, 2/94, 1994, S. 58 – 75.

[Hirschberger 2000] *Hirschberger; Johannes:* Geschichte der Philosophie. 2 Bände, Lizenzausgabe der 12. Auflage von 1980, Frankfurt am Main, 2000.

[Hitch 1973] *Hitch, Charles:* An Appreciation of Systems Analysis. In: [Optner 1973], S. 19 – 36.

[Hitz 1999] *Hitz, Martin; Kappel, Gerti:* UML@Work. Von der Analyse bis zur Realisierung. Heidelberg 1999.

[HMD 1998] HMD. Praxis der Wirtschaftsinformatik. Heft 198, Business Process (Re-)Engineering, November 1998.

[Hoag 1973] *Hoag, Malcolm W.:* An Introduction to Systems Analysis. In: [Optner 1973], S. 37 – 52.

[Hoffmann 1992] *Hoffmann, W.; Scheer, A.-W., Backes, R.:* Konzeption eines Ereignisklassifikationssystems in Prozessketten. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes, Heft 95, Saarbrücken 1992.

[Hoffmann 1992b] *Hoffmann, Friedrich:* Aufbauorganisation. In: [Frese 1992], Spalte 208 – 221.

[Hoheisel 1999] *Hoheisel, Holger:* Berücksichtigung temporaler Konstrukte bei der Geschäftsprozessmodellierung durch Geschäftsregeln. In: [MobIS 1999], S. 108 – 126.

[Hoogeboom 1991] *Hoogeboom, B.; Halang, W. A.:* The Concept of Time in Software Engineering for Real Time Systems. In: Third International Conference on Software Engineering for Real Time Systems. 16. – 18. September 1991, London 1991, Seite 156 – 163.

[Horn 1993] *Horn, Erika; Schubert, Wolfgang:* Objektorientierte Software-Konstruktion. Grundlagen – Modelle – Methoden – Beispiele. München 1993.

[Hoyer 1988] *Hoyer, Rudolf:* Organisatorische Voraussetzungen der Bürokommunikation. Rechnergestützte, prozessorientierte Planung von Büroinformations- und -kommunikationssystemen. Münster 1988.

[Huber 1997] *Huber, Heinrich; Poestges, Axel:* Geschäftsprozessmanagement. In: [Corsten 1997], S. 75 – 93.

[Hubert 2002] *Hubert, Richard:* Convergent Architecture: building model-driven J2EE systems with UML. New York, Chichester, Weinheim u. a. 2002.

[Hutt 1994] *Hutt, T. F.:* Object Analysis and Design. Comparison of Methods. New York u. a. 1994.

[HWO 1992] *Frese, Erich (Hrsg.):* Handwörterbuch der Organisation. Enzyklopädie der Betriebswirtschaftslehre, Band 2. Stuttgart 1992.

[IACS 1996] *The Institute of Applied Computer Science. The VDM Tool Group:* The IFAD VDM++ Language. Doc. Id. IFAD-VDM-41, Ver. 1.1, Dezember, ohne Ort, 1996.

[IAO 2001] *Institut Arbeitswirtschaft und Organisation der Fraunhofer-Gesellschaft:* Enterprise Process Management. Konzepte, Methoden und Werkzeuge für die Gestaltung moderner Geschäftsprozesse. Forum, 3. und 4. Mai 2001. Institutszentrum Stuttgart der Fraunhofer-Gesellschaft. Stuttgart, 2001

[IBM 1979] *IBM Deutschland GmbH:* HIPO. Eine Design-Hilfe und Dokumentationstechnik. IBM Form GC12-1296-1. Ohne Ort, 1979.

[IBM 1984] *International Business Machines:* Business Systems Planning. 4. Auflage, GE20-0527-4, Atlanta 1984.

[IBM 1993] *International Business Machines:* IBM Line of Visibility Engineering Methodology. Consultant's Guide. Release 1.1, IBM Canada, 2. Auflage, ohne Ort, Dezember 1993.

[IBM 1997] *International Business Machines Object-Oriented Technology Center:* Developing Object-Oriented Software. An Experience-Based Approach. Upper Saddle River, NJ, 1997.

[IDEF0 1993] *Computer Systems Laboratory, National Institute of Standards and Technology:* IDEF0. "FIPS Integration Definition for Function

Modeling (IDEF0)". Federal Information Processing Standards Publication 183, 1993. (http://www.idef.com, Stand 16.10.2000)

[IDEF1X 1993b] *Computer Systems Laboratory, National Institute of Standards and Technology:* IDEF1X. "FIPS Integration Definition for Information Modeling (IDEF1X)". Federal Information Processing Standards Publication 184, 1993. (http://www.idef.com, Stand 16.10.2000)

[IDEF3 1995] *Mayer, Richard J.; Menzel, Christopher P.; Painter, Michael K.; deWitte, Paula S.; Blinn, Thomas.; Perakath, Benjamin:* IDEF3 Process Description Capture Method Report. Wright-Patterson AFB, Ohio, AL-TR-1995-XXXX. September 1995. (http://www.idef.com, Stand 16.10.2000)

[IDEF4 1995b] *Knowledge Based Systems, Inc.:* Information Integration for Concurrent Engineering (IICE). IDEF4 Object-Oriented Design Method. Draft. College Station, Texas, Januar 1995. (http://www.idef.com, Stand 16.10.2000)

[ITU-T 1993] *International Telecommunication Union:* Criteria for the use and applicability of formal description techniques. Message Sequence Chart (MSC). ITU-T Recommendation Z.120. 3/93. International Telegraph & Telephone Consultative Committee (CCITT), Geneve 1993.

[Isernhagen 2000] *Isernhagen, Rolf:* Softwaretechnik in C und C++. Modulare, objektorientierte und generische Programmierung. München 2000.

[Jablonski 1995] *Jablonski, Stefan; Stein, Katrin:* Die Eignung objektorientierter Analysemethoden für das Workflow Management. In: HMD. Praxis der Wirtschaftsinformatik, Heft 185, 1995, S. 95 – 115.

[Jablonski 1995b] *Jablonski, Stefan:* Workflow-Management-Systeme. Modellierung und Architektur. Bonn 1995.

[Jablonski 1997] *Jablonski, Stefan; Böhm, Markus; Schulze, Wolfgang (Hrsg.):* Workflow Management. Entwicklung von Anwendungen und Systemen. Heidelberg 1997.

[Jablonski 1997] *Jablonski, Stefan:* Anwendungsfelder für Workflow-Management in Unternehmen. Vortrag auf der [IAO 2001].

[Jackson 1995] *Jackson, Michael:* Software Requirements & Specifications. A lexikon of practise, principles and prejudices. Wokingham, England, 1995.

[Jacobson 1994] *Jacobson, Ivar; Christerson, Magnus; Jonsson, Patrik; Övergaard, Gunnar:* Object-Oriented Software Engineering. A Use Case Driven Approach. Wokingham, England, Reprint 1994.

[Jacobson 1995] *Jacobson, Ivar; Ericsson, Maria; Jacobson, Agneta:* The Object-Advantage. Business Process Reengineering with Object Technology. Wokingham, England, Reprint 1995.

[Jacobson 1995b] *Jacobson, Ivar; Cristerson, Magnus:* A Growing Consensus on Use Cases. In: Journal of Object-Oriented Programming. März/April 1995. S.15 – 19.

[Jacobson 1999] *Jacobson, Ivar; Booch, Grady; Rumbaugh, James:* The Unified Software Development Process. Reading, MA, u. a. 1999.

[Jacobson 2000] *Jacobson, Ivar:* The Road to the Unified Software Development Process, Cambridge, UK, 2000.

[Jaeschke 1996] *Jaeschke, Peter:* Geschäftsprozessmodellierung mit INCOME. In: [Vossen 1996], S. 142 – 162.

[Jarke 1999] *Jarke, Matthias:* Scenarios for Modeling. In: Communications of the ACM, Band 42, Nr. 1, Januar 1999, S. 47 – 48.

[Johann 2000] *Johann, Michael; Gille, Marc:* Global Processes – Globale Geschäftsprozesse als Denkweise. In: sw development, Nr. 1, November/Dezember 2000, S. 14 – 19.

[Johansson 1994] *Johannson, Henry; McHugh, Patrick; Pendelbury, A. John; Wheeler, William A.:* Business Process Reengineering. Breakpoint Strategies for Market Dominance. Chichester, New York, Brisbane. Reprint 1994.

[Jones 1990] *Jones, Cliff B.:* Systematic Software Development using VDM. 2. Auflage, Englewood Cliffs, NJ, 1990.

[Joos 1998] *Joos, Stefan; Berner, Stefan; Glinz, Martin; Arnold, Martin:* Stereotypen und ihre Verwendung in objektorientierten Modellen – Eine Klassifikation. In: [Pohl 1998].

[Kappel 1996] *Kappel, Gerti; Schrefl, Michael:* Objektorientierte Informationssysteme. Konzepte, Darstellungsmittel, Methoden. Wien, New York 1996.

[Kappler 1991] *Kappler, Ekkehard; Rehkugler, Heinz:* Konstitutive Entscheidungen. In: [Heinen 1991], S. 73 – 240.

[Kappler 1991b] *Kappler, Ekkehard; Rehkugler, Heinz:* Kapitalwirtschaft. In: [Heinen 1991], S. 897 – 1069.

[Kargl 1989] *Kargl, Herbert:* Fachentwurf für DV-Anwendungssysteme. München 1989.

[Kaschek 1995] *Kaschek, Roland; Kohl, Claudia; Mayr, Heinrich C.:* Cooperations - An Abstraction Concept Suitable for Business Process Re-Engineering. In: [Györkös 1995], S. 161 – 172.

[Keller 1992] *Keller, Gerhard; Nüttgens, M.; Scheer, A.-W.:* Semantische Prozessmodellierung auf der Grundlage “Ereignisgesteuerter Prozessketten”. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes, Heft 89, Saarbrücken 1992.

[Keller 1993] *Keller, Gerhard:* Informationsmanagement in objektorientierten Organisationstrukturen. Dissertation, Wiesbaden 1993.

[Keller 1995] *Keller, Gerhard, Popp, Karl:* Referenzmodelle für Geschäftsprozesse. In: HMD. Praxis der Wirtschaftsinformatik, Heft 187, 1995, S. 94 – 117.

[Keller 1996] *Keller, Gerhard; Schröder, Gerd:* Geschäftsprozessmodelle: Vergangenheit – Gegenwart – Zukunft. In: Management & Computer, 4. Jahrgang, Heft 2, 1996, S. 77 – 88.

[Keller 1998] *Keller, Gerhard; Teufel, Thomas:* SAP R/3 prozessorientiert anwenden. Iteratives Prozess-Prototyping zur Bildung von Wertschöpfungsketten. Bonn u. a. 1998.

[Kemper 1999] *Kemper, Alfons:* Datenbanksysteme. Eine Einführung. 3., korr. Auflage. München 1999.

[Kern 1979] *Kern, Werner (Hrsg.):* Handwörterbuch der Produktionswirtschaft. Enzyklopädie der Betriebswirtschaftslehre Band 7. Stuttgart 1979.

[Kern 1979b] *Kern, Werner:* Produktionswirtschaft. In: [Kern 1979], Spalte 1647 – 1660.

[Kieser 1992] *Kieser, Alfred:* Abteilungsbildung. In: [Frese 1992], Spalte 57 – 72.

[Kieser 1996] *Kieser, Alfred:* Business Process Reengineering – neue Kleider für den Kaiser? In: [Perlitz 1996], S. 235 – 251.

[Kilberth 1994] *Kilberth, Klaus; Gryczan, Guido; Züllinghoven, Heinz:* Objektorientierte Anwendungsentwicklung. Konzepte, Strategien, Erfahrungen. 2. verbesserte Auflage. Braunschweig 1994.

[Kim 1994] *Kim, Stefan; Unland, Rainer:* Flexible Organisation durch Workflow Management? Oder: Zum Problem der Modellierung von Geschäftsprozessen. In: [Hasenkamp 1994], S. 13 – 27.

[Kirsche 1994] *Kirsche, Thomas; Reinwald, Berthold; Wedekind, Hartmut:* Processing Dynamic Interactions in Cooperative Databases. In: Proceedings 27th Annual Hawaii International Conference on System Sciences, HICSS-27, Hawaii 1994. S. 733 – 742.

[Klaus 1972] *Klaus, Georg; Buhr, Manfred (Hrsg.):* Philosophisches Wörterbuch. 2 Bände, Berlin (Ost) 1972.

[Klaus 1979] *Klaus, Georg; Liebscher, Heinz:* Wörterbuch der Kybernetik. 2 Bände. 4., völlig überarbeitete Auflage. Frankfurt a. M. 1979.

[Kleinknecht 1976] *Kleinknecht, Reinhard; Wüst, Eckehard:* Lehrbuch der elementaren Logik. Band 2: Prädikatenlogik. München 1976.

[Kleinknecht 1976a] *Kleinknecht, Reinhard; Wüst, Eckehard:* Lehrbuch der elementaren Logik. Band 1: Aussagenlogik. München 1976.

[Kleinsorge 1994] *Kleinsorge, Peter:* Geschäftsprozess. In: [Masing 1994], S. 49 – 64.

[Klepzig 1997] *Klepzig, Heinz-Jürgen; Schmidt, Klaus-J.:* Prozess-Management mit System. Unternehmensabläufe konsequent optimieren. Wiesbaden 1997.

[Kling 1980] *Kling, Rob:* Social Analysis of Computing: Theoretical Perspectives in Recent Empirical Research. In: ACM Computing Surveys, Band 12, Nr.1, 1980, S. 61 – 110.

[Kling 1982] *Kling, Rob; Scacchi, Walt:* The Web of Computing: Computing Technology as Social Organization. In: Advances in Computers, Band 21, 1982, S. 1 – 90.

[Klotz 1993] *Klotz, Ulrich:* Vom Taylorismus zur Objektorientierung. In: [Scharfenberg 1993], S. 161 – 199.

[Kobryn 1999] *Kobryn, Chris:* UML 2001: A Standardization Odyssey. In: Communications of the ACM, Band 42, Nr. 10, 1999, S. 29 – 37.

[Koch 1997] *Koch, Michael; Vogel, Thomas:* Von der Vertikale in die Horizontale. In: [Corsten 1997b], S. 59 – 71.

[Kocher 1998] *Kocher, Hartmut:* Die Modellierung komplexer Software-Systeme. In: microsoft system journal, Heft 5, 1998, S. 54 – 58.

[Köhler 1975] *Köhler, Richard:* Modelle. In: [Grochla 1975b], Spalte 2701 – 2716.

[König 1995] *König, Wolfgang (Hrsg.):* Wirtschaftsinformatik ’95: Wettbewerbsfähigkeit, Innovation, Wirtschaftlichkeit. Heidelberg 1995.

[Kopp 1988] *Kopp, Herbert:* Compilerbau: Grundlagen, Methoden, Werkzeuge. München 1988.

[Koreimann 1972] *Koreimann, Dieter S.:* Systemanalyse. Berlin, New York 1972.

[Korthaus 1998b] *Korthaus, Axel:* Using UML for Business Object Bases Systems Modeling. In: [Schader 1998], S. 220 – 237.

[Kosiol 1961] *Kosiol, Erich:* Bürowirtschaftliche Forschung. Berlin 1961.

[Kosiol 1962] *Kosiol, Erich:* Organisation der Unternehmung. Wiesbaden 1962.

[Kosiol 1966] *Kosiol, Erich:* Die Unternehmung als wirtschaftliches Aktionszentrum, Reinbek 1966.

[Kosiol 1968] *Kosiol, Erich:* Grundlagen und Methoden der Organisationsforschung. 2., überarbeitete und erweiterte Auflage, Berlin 1968.

[Krallmann 1994] *Krallmann, Hermann:* Systemanalyse im Unternehmen. Geschäftsprozessoptimierung, Partizipative Vorgehensmodelle, Objektorientierte Analyse. Unter Mitarbeit von *Derszteler, Gérard; u. a.* München, Wien 1994.

[Kraus 1994] *Kraus, Herbert:* Historische Entwicklung von Organisationstrukturen – Ursache für die Notwendigkeit neuer Organisationskonzepte? In: [Krickl 1994], S. 3 – 15.

[Krcmar 1996] *Krcmar, Helmut; Zerbe, Stefan:* Negotiation enabled Workflow (NEW): Workflowsysteme zur Unterstützung flexibler Geschäftsprozesse. In: [Becker 1996b], S. 28 – 26.

[Krcmar 1997] *Krcmar, Helmut; Schwarzer, Bettina; Zerbe, Stefan:* Innovativer Werkzeugeinsatz zur Unterstützung prozessorientierter Organisation. In: [Corsten 1997], S. 155 – 193.

[Kredel 1999] *Kredel, Heinz; Yoshida, Akitosh:* Thread- und Netzwerkprogrammierung mit Java. Heidelberg 1999.

[Krickl 1994] *Krickl, Christian (Hrsg.):* Geschäftsprozessmanagement: prozessorientierte Organisationsgestaltung und Informationstechnologie. Heidelberg 1994.

[Krickl 1994b] *Krickl, Christian:* Business Redesign. Prozessorientierte Organisationsgestaltung und Informationstechnologie. In: [Krickl 1994], S. 17 – 38.

[Kristen 1995] *Kristen, Gerald:* Business Engineering. In: Journal of Object-Oriented Programming, Band 2, Nr. 2, Juli/August 1995, S. 14 – 19.

[Kristensen 1996] *Kristensen, Bent Bruun; Østerbye, Kasper:* Roles: Conceptual Abstraction Theory and Practical Language Issues. In: Theory and Practice of Object Systems, Band 2, Nr. 3, 1996, S. 143 – 160.

[Kruchten 1998] *Kruchten, Philippe:* The Rational Unified Process. Reading, MA, 1998.

[Kruchten 1999] *Kruchten, Philippe:* Der Rational Unified Process. Bonn 1999.

[Krüger 1992] *Krüger, Wilfried:* Aufgabenanalyse und -synthese. In: [Frese 1992], Spalte 221 – 236.

[Krüger 1994] *Krüger, Wilfried:* Organisation der Unternehmung. 3., verbesserte Auflage. Köln 1994.

[Kruse 1992] *Kruse, Christian; Scheer, A.-W.:* Modellierung und Analyse dynamischen Systemverhaltens. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes, Heft 94, Saarbrücken 1992.

[Kruse 1996] *Kruse, Christian:* Referenzmodellgestütztes Geschäftsprozessmanagement. Dissertation, Wiesbaden 1996.

[Kueng 1995] *Kueng, Peter; Schrefl, Michael:* Spezialisierung von Geschäftsprozessen am Beispiel der Bearbeitung von

Kreditanträgen. In: HMD. Praxis der Wirtschaftsinformatik, Heft 185, 1995. S. 78 – 94.

[Kueng 1995b] *Kueng, Peter; Bichler, Peter, Schrefl, Michael:* Geschäftsprozessmodellierung: ein zielbasierter Ansatz. Institutsbericht 95.04, Universität Linz, Institut für Wirtschaftsinformatik, Data & Knowledge Engineering, September 1995.

[Kueng 1996] *Kueng, Peter; Bichler, Peter; Schrefl, Michael:* Geschäftsprozessmodellierung: ein zielbasierter Ansatz. In: Information Management, Heft 2, 1996, S. 40 – 50.

[Kueng 1996b] *Kueng, Peter; Bichler, Peter:* How to compose an Object-Oriented Business Process Model? IPG Working Paper, Januar 1996. (http://www.citeseer.nj.nec.com/kueng96how.html, Stand 11.10.2000)

[Kueng 2000] *Kueng, Peter:* Leistungsmessung von Geschäftsprozessen mit Hilfe der Informationstechnologie. In: WiSu, Nr. 6, 2000, S. 829 – 835.

[Lano 1995] *Lano, Keith:* Formal Object-Oriented Development. London, Berlin, Heidelberg, New York 1995.

[Laske 1992] *Laske, Stephan; Weiskopf, Richard:* Hierarchie. In: [Frese 1992], Spalte 791 – 807.

[Lausen 1988] *Lausen, Georg:* Modeling and Analysis of the Behavior of Information Systems. In: IEEE Transaction on Software Engineering, Band 14, Nr. 11, 1988, S. 1610 – 1620.

[Lea 1997] *Lea, Doug:* Concurrent Programming in Java. Entwurfsprinzipien und Muster. Bonn 1997.

[Lecoeuche 1995] *Lecoeuche, Hugues; Sourrouille Jean Louis:* A Dynamic Model Extension to Integrate State Inheritance and Objects. In: Report on Analysis & Design, Band 2, Juli/August, 1995, S. 37 – 43.

[Leffingwell 2000] *Leffingwell, Dean; Widrig, Don:* Managing Software Requirements. A Unified Approach. Reading, MA, 2000.

[Lehmann 1975] *Lehmann, Helmut:* Probleme einer systemtheoretisch-kybernetischen Untersuchung betrieblicher Systeme. In: [Grochla 1975], S. 567 – 586.

[Lehmann 2000] *Lehmann, Frank R.; Ortner, Erich:* Vorgehensmodell beim Fachentwurf von Workflow-Management-Anwendungen mit einem Repositorium. In: Wirtschaftsinformatik, Band 42, Nr. 1, 2000, S. 47 – 55.

[Lehner 1991] *Lehner, Franz; Auer-Zizzi, Werner; Bauer, Robert; u. a.:* Organisationslehre für Wirtschaftsinformatiker. München 1991.

[Leibnitz 1979] *Leibnitz, Gottfried Wilhelm:* Monadologie. Stuttgart 1979.

[Leymann 1996] *Leymann, Frank:* Transaktionskonzepte für Workflow-Management-Systeme. In: [Vossen 1996], S. 335 – 352.

[Liebelt 1992] *Liebelt, Wolfgang:* Ablauforganisation, Methoden und Techniken der. In: [Frese 1992], Spalte 19 – 34.

[Liebelt 1992b] *Liebelt, Wolfgang; Sulzberger, Markus:* Grundlagen der Ablauforganisation. Gießen 1989.

[Liebmann 1984] *Liebmann, Wolfgang:* Text- und Datenverarbeitung im Büro als Bestandteil des unternehmensweiten Informationssystemkonzeptes. In: [Witte 1984], S. 211 – 224.

[Liebmann 1997] *Liebmann, Hans-Peter (Hrsg.):* Vom Business Process Reenginering zum Change Management. Wiesbaden 1997.

[Linssen 1996] *Linssen, Oliver:* Wiederverwendung im Software Engineering. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 94-5, Universität Wuppertal, September 1994.

[Linssen 1996] *Linssen, Oliver:* Zur Modellierung von Geschäftsprozessen mit der Object Behavior Analysis. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 96-1, Universität Wuppertal, November 1996.

[Linssen 1998] *Linssen, Oliver:* Systemanalyse im objektorientierten Umfeld. Methoden und Vorgehensweisen zur Erstellung von Pflichtenheften. In: VOCA GmbH: Tagungsband der Devcon ’98. Nürnberg 1998.

[Linssen 1998b] *Linssen, Oliver:* Object Behavior Analysis. Ein Ansatz zur Untersuchung von Abläufen in objektorientierten Systemen. In: VOCA GmbH: Tagungsband der Devcon ’98. Nürnberg 1998.

[Linssen 1999] *Linssen, Oliver:* Objektorientierte Modellierung mit der UML und dem Unified Software Development Process (USDP). Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 99-4, Universität Wuppertal, Oktober 1999.

[Linssen 1999b] *Linssen, Oliver:* Szenarien und Use Cases in der objektorientierten Modellierung. In: HMD. Praxis der Wirtschaftsinformatik. Heft 210, 36. Jahrgang, Dezember 1999, S. 69 – 83.

[Linssen 1999c] *Linssen, Oliver; Flygare, Marcel; Fischer, Michael:* Das objektorientierte Datenbankmanagementsystem Jasmine: Übersicht und Einsatzmöglichkeiten. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 99-2, Universität Wuppertal, Juni 1999.

[Linssen 1999d] *Linssen, Oliver:* Objektorientierter Entwurf mit ODBMS am Beispiel CA-Jasmine. Klassenentwurf und Abfragen. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 99-5, Universität Wuppertal, Oktober 1999.

[Linssen 1999e] *Linssen, Oliver; Lange, Thorsten:* Datenmodellierung mit dem Entity-Relationship-Ansatz*:* Abbildungsprozess von Realitätsbeobachtungen bis zur SQL DDL-Anweisung. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 99-1, Universität Wuppertal, Mai 1999.

[Linssen 1999f] *Linssen, Oliver:* Die objektorientierte Modellierungssprache UML (Unified Modeling Language). Umfang und kritische Reflexion eines kommenden Industrie-Standards. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 99-3, Universität Wuppertal, Oktober 1999.

[Linssen 1999g] *Linssen, Oliver:* Das objektorientierte Konstruktionshandbuch nach dem Werkzeug-&-Material-Ansatz. Buch-Rezension. In: Informatik-Spektrum, Band 22, Dezember 1999, S. 459 – 460.

[Lipeck 1989] *Lipeck, Udo W.:* Dynamische Integrität von Datenbanken. Grundlagen der Spezifikation und Überwachung. Habilitationsschrift, Braunschweig, Berlin, Heidelberg, New York 1989.

[Liskov 1974] *Liskov, Barbara; Zilles, Stephen N.:* Programming with Abstract Data Types. In: ACM SIGPLAN Notices, 9. Jahrgang, Heft 4, 1974, S. 50 – 59.

[Liskov 1986] *Liskov, Barbara; Guttag, John:* Abstraction and Specification in Program Development. Cambridge 1986.

[Liskov 1993] *Liskov, Barbara; Guttag, John:* A New Definition of the Subtype Relation. In: Nierstrasz, Oskar M. *(Hrsg.):* ECOOP '93 – Object-Oriented Programming. Berlin, Heidelberg, New York 1993, Seite 118 – 141.

[Liskov 1994] *Liskov, Barbara; Wing, Jeannette M.:* A Behavioral Notion of Subtyping. In: ACM Transactions on Programming Languages and Systems, Band 16, Nr. 6, November 1994, S. 1811 – 1841.

[Lockemann 1993] *Lockemann, Peter C.; Krüger, Gerhard; Krumm, Heiko:* Telekommunikation und Datenhaltung. München 1993.

[Lohoff 1993] *Lohoff, Petra; Lohoff, Heinz-Günter:* Verwaltung im Visier. Optimierung der Büro- und Dienstleistungsprozesse. In: Zeitschrift Führung + Organisation, 62. Jahrgang, Nr. 4, Juli/August 1993, S. 248 – 254.

[Loos 1998] *Loos, Peter; Allweyer, Thomas:* Process Orientation and Object Orientation – An Approach for Integrating UML and Event-Driven Process Chains (EPC). In*:* Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes. Heft 144, Saarbrücken 1998.

[Lorenz 1984] *Lorenz, Gert:* Integration von Kommunikationsformen im Büro. In: [Witte 1984], S. 225 – 241.

[Louden 1994] *Louden, Kenneth C.:* Programmiersprachen. Grundlagen, Konzepte, Entwurf. Bonn, u. a. 1994.

[Ludwig 1994] *Ludwig, Börries; Krcmar, Helmut:* Verteiltes Problemlösen in Gruppen mit CONSUL. In: [Hasenkamp 1994], S. 167 – 186.

[Lutz 1997] *Lutz, Wolf-Guido:* Das objektorientierte Paradigma: Struktur und organisationstheoretische Perspektiven einer Softwaretechnologie. Dissertation, Wiesbaden 1997.

[Macharzina 1995] *Macharzina, Klaus:* Unternehmensführung. Das internationale Managementwissen. Konzepte – Methoden – Praxis. 2., aktualisierte und erweiterte Auflage. Wiesbaden 1995.

[Malischewski 1997] *Malischewski, Carsten:* Generierung von Spezifikationen betrieblicher Anwendungssysteme auf der Basis von Geschäftsprozessmodellen. Dissertation, Aachen 1997.

[Malone 1990] *Malone, Thomas W.; Crowston, Kevin:* What is Coordination Theory an How Can it Help Design Cooperative Systems? In: Proceedings Conference on Computer Supported Cooperative Work (CSCW), Los Angeles, CA, 7.-10. Oktober 1990. New York, NY, 1990, S. 357 – 370.

[Mandler 1984] *Mandler, Jean Matter:* Stories, Scripts, and Scenes: Aspects of Schema Theory. Hillsdale, NJ, London 1984.

[Mandrioli 1992] *Mandrioli, Dino; Meyer, Bertrand (Hrsg.):* Advances in Object-Oriented Software Engineering. Englewood Cliffs, NJ, 1995.

[Mann 1975] *Mann, Floyd C.:* Organisatorische Auswirkungen der Automatisierung im Bürobereich. In: [Grochla 1975], S. 498 – 506.

[Marshall 2000] *Marshall, Chris:* Enterprise Modeling with UML: Designing succesful Software through Business Analysis. Reading, MA, 2000.

[Martin 1990] *Martin, James:* Information Engineering, Book II: Planning and Analysis. Englewood Cliffs, NJ, 1990.

[Martin 1992] *Martin, James; Odell, James J.:* Object-Oriented Analysis and Design. Englewood Cliffs, NJ, 1992.

[Martin 1993] *Martin, James:* Principles of Object-Oriented Analysis and Design. Englewood Cliffs, NJ, 1993.

[Martin 1995] *Martin, James; Odell, James J.:* Object-Oriented Methods: A Foundation. Englewood Cliffs, NJ, 1995.

[Martin 1999] *Martin, James; Odell, James J.:* Objektorientierte Modellierung mit UML: Das Fundament. München 1999.

[Masing 1994] *Masing, Walter (Masing):* Handbuch Qualitätsmanagement. 3. gründlich überarbeitete und erweiterte Auflage. München 1994.

[Mattheis 1993] *Mattheis, Peter:* Prozessorientierte Informations- und Organisationsstrategie. Analyse, Konzeption, Realisierung. Wiesbaden 1993.

[Mayer 1995] *Mayer, Richard J.; Crump, John W.; Fernandes, Ronald; Keen, Arthur; Painter, Michael K.:* Compendium of Methods Report. 45433 - 7604. Wright-Patterson Air Force Base, Ohio Juni 1995. (http://www.idef.com, Stand 16.10.2000)

[Mayer] *Mayer, Richard J.; deWitte, Paula S.:* Delivering Results: Evolving BPR from Art to Engineering. College Station, Texas, ohne Jahr. (http://www.idef.com, Stand 16.10.2000)

[Mayr 1993] *Mayr, Heinrich C.; Wagner, Roland (Hrsg.):* Objektorientierte Methoden für Informationssysteme. Fachtagung der GI-Fachgruppe EMISA. Berlin, Heidelberg, New York 1993.

[McGregor 1993] *McGregor, John D.; Dyer, Douglas M.:* A Note on Inheritance and State Machines. In: ACM SIGSOFT Software Engineering Notes, Band 18, Nr. 4, Oktober 1993, S. 61 – 69.

[McLaughlin 1998] *McLaughlin, Michael J.; Moore, Alan:* Real-Time Extensions to UML. Timing, concurrency, and hardware interfaces. In: Dr. Dobb's Journal. Dezember 1998. (Zitiert nach: www.ddj.com/articles/1998/9812g/9812g.htm, Stand 17.10.1999)

[McMenamin 1988] *McMenamin, Stephen; Palmer, John F.:* Strukturierte Systemanalyse. München 1988.

[Medina-Mora 1992] *Medina-Mora, Raul; Winograd, Terry; Flores, Fernando:* The Action Workflow Approach to Workflow Management Technology. In: Proceedings of the 4th Conference on Computer Supported Cooperative Work, November 1992, Toronto, Canada. S. 281 – 288.

[Mellis 1998] *Mellis, Werner; Herzwurm, Georg; Stelzer, Dirk:* TQM der Softwareentwicklung. 2., verbesserte Auflage, Braunschweig 1998.

[Mertens 1988] *Mertens, Peter:* Integrierte Informationsverarbeitung 1. Administrations- und Dispositionssysteme in der Industrie. 8., völlig neu bearbeitete und erweiterte Auflage. Wiesbaden 1988.

[Mertins 1995] *Mertins, Kai; Jochem, Roland:* Unternehmensmodellierung – Basis für Reengineering und Optimierung von Geschäftsprozessen. In: [König 1995], S. 99 – 112.

[Meyer 1985] *Meyer, Bertrand:* On Formalism in Specifications. In: IEEE Software, Januar 1985, S. 6 – 26.

[Meyer 1990] *Meyer, Bertrand:* Objektorientierte Softwareentwicklung. München 1990.

[Meyer 1992] *Meyer, Bertrand:* Applying "Design by Contract“. In: IEEE Computer, Oktober 1992, S. 40 – 51.

[Meyer 1992b] *Meyer, Bertrand:* Eiffel: The Language. Reprint with Corrections. New York, London, Toronto 1992.

[Meyer 1992c] *Meyer, Bertrand:* Design by Contract. In: [Mandrioli 1992], S. 1 – 50.

[Meyer 1995] *Meyer, Bertrand:* Object Success. London, New York, Toronto 1995.

[Meyer 1997] *Meyer, Bertrand:* Object-Oriented Software Construction. 2. Auflage, Upper Saddle River, NJ, 1997.

[Meyer 1997b] *Meyer, Bertrand:* Eiffel's Design by Contract: Predecessors and Original Contributions. (http://www.elj.com/eiffel, Stand 16.12.1999) Santa Barbara, CA, 1997.

[Meyer 1999] *Meyer, Bertrand:* Building bug-free O-O software: An introduction to Design by Contract. (http://eiffel.com/doc, Stand 16.12.1999) Santa Barbara, CA, 1999.

[MeyerM 1996] *Meyer, Manfred:* Operations Research Systemforschung. Eine Einführung in die praktische Bedeutung. 4., überarbeitete Auflage, Stuttgart, Jena 1996.

[Meyer-Schönherr 1992] *Meyer-Schönherr, Mirko:* Szenario-Technik als Instrument der strategischen Planung. Ludwigsburg, Berlin 1992.

[Mielke 2002] *Mielke, Carsten:* Geschäftsprozesse. UML-Modellierung und Anwendungs-Generierung. Heidelberg 2002.

[Miers 1994] *Miers, Derek:* Use of Tools and Technology within a BPR Initiative. In: [Coulson-Thomas 1994], S. 142 – 165.

[Miller 1956] *Miller, George A.:* The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. In: Psychological Review, Jahrgang 63, Heft 2, 1956, S. 81 – 97.

[Miller 1978] *Miller, James Grier:* Living Systems. New York, St. Louis, San Francisco 1978.

[Milling 1981] *Milling, Peter:* Systemtheoretische Grundlagen zur Planung der Unternehmenspolitik. Berlin 1981.

[Milner 1989] *Milner, Robin:* Communication and Concurrency. New York 1989.

[MobIS 1996] *Gesellschaft für Informatik, Fachbereich Wirtschaftsinformatik, Fachausschuß 5.2, Fachgruppe 5.2.1* (MobIS)*:* Rundbrief des GI-Fachausschusses 5.2, 3. Jahrgang, Heft 1, September 1996.

[MobIS 1997] *Gesellschaft für Informatik, Fachbereich Wirtschaftsinformatik, Fachausschuß 5.2, Fachgruppe 5.2.1* (MobIS)*:* Rundbrief des GI-Fachausschusses 5.2, 4. Jahrgang, Heft 1, September 1997.

[MobIS 1998] *Gesellschaft für Informatik, Fachbereich Wirtschaftsinformatik, Fachausschuß 5.2, Fachgruppe 5.2.1* (MobIS)*:* Rundbrief des GI-Fachausschusses 5.2, 5. Jahrgang, Heft 2, September 1998.

[MobIS 1999] *Gesellschaft für Informatik, Fachbereich Wirtschaftsinformatik, Fachausschuß 5.10* (MobIS)*:* Rundbrief des GI-Fachausschusses 5.10, 6. Jahrgang, Heft 1, Oktober 1999.

[MobIS 2000] *Gesellschaft für Informatik, Fachbereich Wirtschaftsinformatik, Fachausschuß 5.10* (MobIS)*:* Rundbrief des GI-Fachausschusses 5.10, 7. Jahrgang, Heft 1, Oktober 2000.

[Morabito 1999] *Morabito, Joseph; Sack, Ira; Bhate, Anikulmar:* Organization Modeling. Innovative Architectures for the 21st Century. Upper Saddle River, NJ, 1999.

[Morton 1991] *Morton, Michael S. Scott (Hrsg.):* The Corporation of the 1990s. Information Technology and Organizational Transformation. New York 1991.

[Morton 1991b] *Morton, Michael S. Scott:* Introduction. In: [Morton 1991], S. 3 – 23.

[Mühlen 1996b] *Mühlen, Michael zur; Rosemann, Michael:* Der Lösungsbeitrag von Metamodellen beim Vergleich von Workflowmanagementsystemen. In: [Becker 1997b], S. 12 – 21.

[Mühlpfordt 1999] *Mühlpfordt, Angela:* Objektorientierte Geschäftsprozessmodellierung auf der Basis der Unified Modeling Language. Dissertation, Berlin 1999.

[Müller 1994] *Müller, Roland; Rupper, Peter (Hrsg.):* Process Reengineering. Prozesse optimieren und auf den Kunden ausrichten. Zürich 1994.

[Müller 1994b] *Müller, Werner; Vogel, Gunther:* dtv-Atlas zur Baukunst. Tafeln und Texte. Band 1. Allgemeiner Teil, Baugeschichte von Mesopotanien bis Byzanz. München 1994.

[Müller-Ettrich 1999] *Müller-Ettrich, Gunter:* Objektorientierte Prozessmodelle. UML einsetzen mit OOTC, V-Modell, Objectory. Bonn 1999.

[Müller-Luschnat 1993] *Müller-Luschnat, G., Hesse, W., Heydenreich, H.:* Objektorientierte Analyse und Geschäftsvorfallsmodellierung. In: [Mayr 1993], S. 74 – 90.

[Mylopoulos 1999] *Mylopoulos, John; Chung, Lawrence; Yu, Eric:* From Object-Oriented to Goal-Oriented Requirements Analysis. In: Communications of the ACM, Band 42, Januar 1999, S. 31 – 37.

[Nagel 1992] *Nagel, Peter:* Zielformulierung, Techniken der. In: [Frese 1992], Spalte 2626 – 2634.

[Naur 1968] *Naur, P.; Randell, P. (Hrsg.):* Software Engineering: Report on a Conference by the NATO Science Comittee. NATO Scientific Affairs Division, Brüssel 1968.

[Nippa 1993] *Nippa, Michael:* Informationstechnik – Motor und Bremse des organisatorischen Wandels. In: [Scharfenberg 1993], S. 323 – 345.

[Noack 1999] *Noack, Jörg; Schienmann, Bruno:* Objektorientierte Vorgehensmodelle im Vergleich. In: Informatik-Spektrum, 22. Jahrgang, S. 166 – 180.

[Nordsieck 1955] *Nordsieck, Fritz:* Rationalisierung der Betriebsorganisation. 2., überarbeitete Auflage von „Grundlagen der Organisationslehre". Stuttgart 1955.

[Nordsieck 1972] *Nordsieck, Fritz:* Betriebsorganisation. Lehre und Technik. 2., bearbeitete Auflage. Textband. Stuttgart 1972.

[Nordsieck 1972b] *Nordsieck, Fritz:* Betriebsorganisation. Lehre und Technik. 2., bearbeitete Auflage. Tafelband. Stuttgart 1972.

[Nüttgens 1998] *Nüttgens, M.; Feld, T.; Zimmerman, V.:* Business Process Modeling with EPC and UML: Transformation or Integration? In: [Schader 1998], S. 250 – 261.

[Nygaard 1986] *Nygaard, Kristen:* Basic Concepts in Object-Oriented Programming. In: ACM SIGPLAN Notices, Band 21, Nr. 10, Oktober 1986, S. 128 – 132.

[Oberweis 1991] *Oberweis, Andreas; Stucky, Wolgang:* Die Behandlung von Ausnahmen in Software-Systemen: Eine Literaturübersicht. Wirtschaftsinformatik, 33. Jahrgang, Heft 6, Dezember 1991, S. 492 – 502.

[Oberweis 1996] *Oberweis, Andreas:* Modellierung und Ausführung von Workflows mit Petri-Netzen. Dissertation, Stuttgart 1996.

[Oberweis 1997] *Oberweis, Andreas:* Geschäftsprozessmodellierung. Vortragsfolien, Frankfurt am Main, Oktober 1997. (http://lwi2.wiwi.uni-frankfurt.de/publikationen/tutorien/tut1/tsld001.htm, Stand 12.10.2000)

[Odell 1992] *Odell, James:* Dynamic and multiple classification. In: Journal of Object-Oriented Programming, Januar 1992, S. 45 – 48.

[Odell 1997] *Odell, James; Ramackers, Guus:* Toward a Formalization of OO Analysis. (http//www.intellicorp.com/ooieonline/odellformal.html, Stand 01.06.2000) Ursprünglich erschienen in: Journal of Object-Oriented Programming, Band 10, Nr. 4, Juni 1997.

[Oeldorf 1998] *Oeldorf, Gerhard; Olfert, Klaus:* Materialwirtschaft. Ludwigshafen 1998.

[Oestereich 1998] *Oestereich, Bernd:* Objektorientierte Softwarentwicklung. Analyse und Design mit der Unified Modeling Language. 4., aktual. Auflage, München, Wien 1998.

[Oestereich 1999] *Oestereich, Bernd (Hrsg.); Hruschka, Peter; Josuttis, Nicolai, Kocher, Hartmut, Krasemann, Hartmut; Reinhold, Markus:* Erfolgreich mit Objektorientierung. München 1999.

[Olsen 1996] *Olsen, Anders; Fægemand, Ove; Mølle-Pederson, Birger; Reed, Rick; Smith J. R. W.:* Systems Engineering Using SDL-92. Third Impression. Amsterdam, Lausanne, New York, Oxford, Shannon, Tokyo 1994.

[OMG 1992] *Object Management Group, Inc. OMG Object Analysis and Design Special Interest Group:* Object Analysis and Design. Band 1, Reference Model, Draft 7.0, Reference No. 92.3.1, März 1992.

[OMG 1999] *Object Management Group, Inc.:* OMG Unified Modeling Language Specification (draft). Ver. 1.3a1, Januar 1999.

[OMG 1999b] *Object Management Group, Inc.:* Meta Object Facility (MOF) Specification. Ver. 1.3 RTF, Juni 1999.

[OMG 2001] *Object Management Group, Inc.:* OMG Unified Modeling Language Specification (draft). Ver. 1.4, Februar 2001.

[OMG 2001b] *Object Management Group, Inc.:* Action Semantics for the UML. OMG ad/2001-03-01. Revised 24. März 2001.

[OMG 2002] *Object Management Group, Inc.:* OMG Unified Modeling Language Specification (Action Semantics). UML Ver. 1.4 with Action Semantics. Final Adopted Specification. Januar 2002.

[Opitz 1997] *Opitz, Otto:* Mathematik. Lehrbuch für Ökonomen. München 1997.

[Optner 1973] *Optner, Stanford L. (ed.):* Systems Analysis. Selected Writings. Middlesex, England, 1973.

[Orfali 1996] *Orfali, Robert; Harkey, Dan; Edwards, Jeri:* The Essential Distributed Objects Survival Guide. New York, Chichester, Brisbane 1996.

[Orfali 1997] *Orfali, Robert; Harkey, Dan; Edwards, Jeri:* Abenteuer Client/Server. Bonn 1997.

[Österle 1995] *Österle, Hubert:* Business Engineering. Prozess- und Systementwicklung. Band 1: Entwurfstechniken. Berlin, Heidelberg, New York 1995.

[Page-Jones 1994] *Page-Jones, Meilir:* Inheritance, Object Aggregation and Interobject Messaging. In: Report on Analysis & Design, Band 1, Nr. 2, 1994, S. 42 – 47.

[Page-Jones 1994b] *Page-Jones, Meilir:* Object States, Transitions, and Asynchronous Messaging. In: Report on Analysis & Design, Band 1, Nr. 3, 1994, S. 10 –15.

[Page-Jones 1995] *Page-Jones, Meilir:* Strukturiertes Systemdesign. Ein praktischer Leitfaden. München 1995.

[Page-Jones 2000] *Page-Jones, Meilir:* Fundamentals of Object-Oriented Design in UML. Reading, MA, 2000.

[Papurt 1994] *Papurt, David M.:* Type and Abstract Data Type. In: Report on Analysis & Design, Band 1, Nr. 2, Juli/August 1994, S. 11 – 14.

[Papurt 1994b] *Papurt, David M.:* Attribute and Association. In: Report on Analysis & Design, Band 1, Nr. 4, November/Dezember 1994, S. 14 – 17.

[Papurt 1996] *Papurt, David M.:* Generalization and Polymorphism. In: Report on Analysis & Design, Band 2, Nr. 5, Januar/Februar 1996, S. 13 – 16.

[ParcPlace 1992] *ParcPlace Systems Inc.:* Object-Oriented Methodology Workshop. Ohne Ort, 1992.

[ParcPlace 1995] *ParcPlace Systems Inc.:* MethodWorks User's Guide. Sunnyvale, CA, 1995.

[Parnas 1972] *Parnas, D. L.:* On the Criteria to be Used in Decomposing Systems into Modules. In: Communications of the ACM, Band 15, Nr. 12, 1972, S. 1053 – 1058.

[Partsch 1998] *Partsch, Helmuth:* Requirements-Engineering systematisch. Berlin, Heidelberg, New York 1998.

[Partridge 1994] *Partridge, Chris:* Modelling the real world: Are classes abstractions or objects? In: Journal of Object-Oriented Programming, November/Dezember 1994, S. 39 – 45.

[Peckham 1988] *Peckham, Joan; Maryanski, Fred:* Semantic Data Models. In: ACM Computing Surveys, Band 20, Nr. 3, September 1988, S. 153 – 189.

[Perlitz 1996] *Perlitz, Manfred; Offinger; Andreas, Reinhardt; Michael, Schug, Klaus (Hrsg.):* Reengineering zwischen Anspruch und Wirklichkeit. Ein Managementansatz auf dem Prüfstand. Wiesbaden 1996.

[Peterson 1981] *Peterson, James L.:* Petri Net Theory and the Modeling of Systems. Englewood Cliffs, NJ, 1981.

[Pfitzinger 1997] *Pfitzinger, Elmar:* Geschäftsprozess-Management. Steuerung und Optimierung von Geschäftsprozessen. Deutsches Institut für Normung e. V., Berlin 1997.

[Picot 1982] *Picot, Arnold:* Transaktionskostenansatz in der Organisationstheorie: Stand der Diskussion und Aussagewert. In: Die Betriebswirtschaft, 42. Jahrgang, 1982, S. 267 – 284.

[Picot 1988] *Picot, Arnold; Franck, Egon:* Die Planung der Unternehmensressource Information (Teil 1). In: WiSu. Das Wirtschaftsstudium, 17. Jahrgang, Heft 10, Oktober 1988, S. 544 – 549.

[Picot 1988b] *Picot, Arnold; Franck, Egon:* Die Planung der Unternehmensressource Information (Teil 2). In: WiSu. Das Wirtschaftsstudium, 17. Jahrgang, Heft 11, November 1988, S. 608 – 613.

[Picot 1990] *Picot, Arnold; Dietl, Helmut:* Transaktionskostentheorie. In: WiSt, Heft 4, April 1990, S. 178 – 184.

[Picot 1991] *Picot, Arnold; Reichwald, Ralf:* Informationswirtschaft. In: [Heinen 1991], S. 241 – 397.

[Picot 1994] *Picot, Arnold; Maier, Matthias:* Ansätze der Informationsmodellierung und ihre betriebswirtschaftliche Bedeutung. In: Zeitschrift für betriebswirtschaftliche Forschung (zfbf), Band 46, Nr. 2, 1994, S. 107 – 126.

[Picot 1994b] *Picot, Arnold; Reichwald, Ralf:* Auflösung der Unternehmung? Vom Einfluss der IuK-Technik auf Organisationsstrukturen und Kooperationsformen. In: ZfB, Band 64, Heft 5, 1994, S. 547 – 570.

[Picot 1995] *Picot, Arnold:* Prozessorganisation – Eine Bewertung der neuen Ansätze aus der Sicht der Organisationslehre. Freiberger Arbeitspapaiere 95/5, Bergakademie Freiberg 1995.

[Picot 1996] *Picot, Arnold; Reichwald, Ralf; Wigand, Rolf T.:* Die grenzenlose Unternehmung. Information, Organisation und Management. 2., aktualisierte Auflage, Wiesbaden 1996.

[Picot 1999] *Picot, Arnold; Dietl, Helmut; Franck, Egon:* Organisation. Eine ökonomische Perspektive. 2. Auflage, Stuttgart 1999.

[Pietsch 1994] *Pietsch, Wolfram; Steinbauer, Dieter:* Business Process Reengineering. In: Wirtschaftsinformatik, Jahrgang 36, 1994, Nr. 5, S. 502 – 505.

[Pohl 1998] *Pohl, K.; Schürr, A.; Vossen, G. (Hrsg.):* Modellierung '98 (Proceedings), Bericht Nr. 6/98-I, Angewandte Mathematik und Informatik, Universität Münster; erschienen als CEUR Workshop Proceedings, Vol-9. (http://www.SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-9/, Stand 21.08.2000)

[Pomberger 1993] *Pomberger, Gustav; Blaschek, Günther:* Software Engineering. Prototyping und objektorientierte Software-Entwicklung. München 1993.

[Popp 1994] *Popp, Karl Michael*: Spezifikation der fachlichen Klassen-Beziehungs-Struktur objektorientierter Anwendungssysteme auf der Grundlage von Modellen der betrieblichen Diskurswelt. Dissertation, Bamberg 1994.

[Porter 1989] *Porter, M. E.:* Wettbewerbsvorteile. Spitzenleistungen erreichen und behaupten. 5. Auflage, Frankfurt a. M. 1999.

[Prasse 1998] *Prasse, Michael:* Die Objektklassifikatoren Typ, Klasse, Rolle und Schnittstelle innerhalb der objektorientierten Modellierungssprache MEMO-OML. In: [MobIS 1998], S. 23 – 30.

[Pratt 1997] *Pratt, Terrence; Zelkowitz, Marvin:* Programmiersprachen. Design und Implementierung. München 1997.

[Probst 1992] *Probst, Gilbert*: Organisation. Strukturen, Lenkungsinstrumente und Entwicklungsperspektiven. Landsberg am Lech 1992.

[Pritsker 1989] *Pritsker, A. Alan B.; Sigal, C. Elliot; Hammesfahr, R. D. Jack:* SLAM II. Network Models for Decision Support. Englewood Cliffs, NJ, 1989.

[Quade 1968] *Quade, E. S. ; Boucher, W. I.:* Systems Analysis and Policy Planning. New York 1986.

[Quibeldey-Cirkel 1994] *Quibeldey-Cirkel, Klaus:* Das Objekt-Paradigma in der Informatik. Stuttgart 1994.

[Raasch 1993] *Raasch, Jörg:* Systementwicklung mit Strukturierten Methoden. 3., bearbeitete, und erweiterte Auflage. München 1993.

[Raffée 1979] *Raffée, Hans; Abel, Bodo (Hrsg.):* Wissenschaftstheoretische Grundfragen der Wirtschaftswissenschaften. München 1979.

[Raster 1994] *Raster, Max:* Prozessarchitektur und Informationsverarbeitung. In: [Gaitanides 1994], S. 123 – 142.

[Raster 1994b] *Raster, Max; Frank Schmidt:* Prozessorientierte Kostenrechnung in der Materialwirtschaft – ein Anwendungsbeispiel. In: [Gaitanides 1994], S. 182 – 207.

[Rathgeb 1994] *Rathgeb, Michael:* Einführung von Workflow-Management-Systemen. In: [Hasenkamp 1994b], S. 45 – 66.

[Rational 1999] *Rational Unified Process Development Team:* Rational Unified Process. Version 5.5. Ohne Ort, 31.8.1999.

[Rational RT] *Rational Software Corporation:* Unified Modeling Language for Real-Time Systems Design. Santa Clara, CA, ohne Jahr.

[Raufer 1995] *Raufer, Heinz; Morschheuser, Stefan; Enders, Wolfgang:* Ein Werkzeug zur Analyse und Modellierung von Geschäftsprozessen als Voraussetzung für effizientes Workflow-Management. In: Wirtschaftsinformatik, Band 37, Nr. 5, 1995, S. 467 – 479.

[Raufer 1997] *Raufer, Heinz:* Dokumentenorientierte Modellierung und Controlling von Geschäftsprozessen. Dissertation, Wiesbaden 1997.

[Rauschecker 2000] *Rauschecker, Josef:* Ein Business Process Reengineering-Ansatz zur integrierten Organisationsgestaltung und Anwendungsentwicklung. Dissertation, Hagen 2000.

[Rechenberg 1999] *Rechenberg, Peter; Pomberger, Gustav:* Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, München 1999.

[Reenskaug 1996] *Reenskaug, Trygve:* Working with Objects. The OOram Software Engineering Method. Greenwich 1996.

[REFA 1991/1] *REFA Verband für Arbeitsstudien und Betriebsorganisation e. V.:* Methodenlehre der Betriebsorganisation. Planung und Steuerung Teil 1. München 1991.

[REFA 1991/6] *REFA Verband für Arbeitsstudien und Betriebsorganisation e. V.:* Methodenlehre der Betriebsorganisation. Planung und Steuerung Teil 6. München 1991.

[Reichwald 1992] *Reichwald, Ralf; Nippa, Michael:* Informations- und Kommunikationsanalyse. In: [Frese 1992], Spalte 855 – 872.

[Reimer 1991] *Reimer, Ulrich:* Einführung in die Wissensrepräsentation. Netzartige und schema-basierte Repräsentationsformate. Stuttgart 1991.

[Reiß 1992] *Reiß, Michael:* Arbeitsteilung. In: [Frese 1992], Spalte 167 – 178.

[Riehle 2000] *Riehle, Dirk:* Framework Design. A Role Modeling Approach. Dissertation, ETH Zürich Nr. 13509, Zürich 2000.

[Rittgen 1998] *Rittgen, Peter:* Prozesstheorie der Ablaufplanung. Stuttgart 1998.

[Rittgen 1999] *Rittgen, Peter:* Objektorientierte Analyse mit EMK. In: [MobIS 1999], S. 8 – 22.

[Rittgen 2000] *Rittgen, Peter:* Quo vadis EPK in ARIS? Ansätze zu syntaktischen Erweiterungen und einer formalen Semantik. In: Wirtschaftsinformatik, Band 42, Nr. 1, 2000, S. 27 – 35.

[Rohloff 1995] *Rohloff, Michael:* Integrierte Informationssysteme durch Modellierung von Geschäftsprozessen. In: [König 1995], S. 83 – 98.

[Rohloff 1996] *Rohloff, Michael:* An Object Oriented Approach to Business Process Modelling. In: [Scholz-Reiter 1996], S. 251 – 264.

[Rolland 1998] *Rolland, Colette; Souveyet, Carine; Achour, Camille Ben:* Guiding Goal Modeling Using Scenarios. In: IEEE Transaction on Software Engineering, Band 24, Nr. 12, Dezember 1998, S. 1055 – 1071.

[Ropohl 1979] *Ropohl, Günter:* Eine Systemtheorie der Technik. Zur Grundlegung der allgemeinen Technologie. München 1979.

[Rosemann 1996] *Rosemann, Michael; Schulte, Rainer:* Effiziente Prozessgestaltung im Rechnungswesen. In: [Vossen 1996], S. 193 – 207.

[Rosenberg 1999] *Rosenberg, Doug; Scott, Kendall:* Use Case Driven Object Modeling with UML. A Practical Approach. Reading, MA, 1999.

[Rosenstengel 1982] *Rosenstengel, Bernd; Winand, Udo:* Petri-Netze. Eine anwendungsorientierte Einführung. Braunschweig 1982.

[Ross 1977] *Ross, Douglas T.:* Structured Analysis for Requirements Definition. In: IEEE Transaction on Software Engineering, Band 3, Nr. 1, 1977, S. 6 – 15.

[Ross 1977b] *Ross, Douglas T.:* Structured Analysis (SA): A Language for Communicating Ideas. In: IEEE Transaction on Software Engineering, Band 3, Nr. 1, 1977, S. 16 – 34.

[Rubin 1992] *Rubin, Kenneth S.; Goldberg, Adele:* Object Behavior Analysis. In: Communications of the ACM, 1992, Band 35, Nr. 9, S. 48 – 62.

[Rubin 1993] *Rubin, Kenneth S.; Goldberg, Adele:* Getting to why. In: Journal of Object-Oriented Programming, Band 6, Nr. 4, 1993. Supplement. S. 5, 8 – 13.

[Rubin 1994] *Rubin, Kenneth S.; McClaughry, Patrick; Pellegrini, David:* Modeling Rules Using Object Behavior Analysis and Design. In: Object Magazine, Juni 1994, S. 63 – 67.

[Rubin 1994b] *Rubin, Kenneth S.:* Object Behavior Analysis- and Design. Tutorial. ParcPlace International Users Conference. Santa Clara, 31.7. – 2.8.1994, ohne Seiten.

[Rudolph 1995] *Rudolph, Ekkart; Graubmann, Peter; Grabowski, Jens:* Tutorial on Message Sequence Charts. In: Tutorials of the 7th SDL-Forum, 25 - 29.9.1995, Oslo, Norwegen. Dordrecht 1995.

[Rudolph 1995b] *Rudolph, Ekkart; Graubmann, Peter; Grabowski, Jens:* Message Sequence Chart: Composition Techniques versus OO-Techniques – "Tema con Variazioni". In: *Braek, R.; Sarma, A. (Hrsg.):* SDL '95 with MSC in CASE. Proceedings of the 7th SDL-Forum, Oslo. Amsterdam 1995.

[Rüffer 1999] *Rüffer, Thorsten:* Referenzgeschäftsprozessmodellierung eines Lebensversicherungsunternehmens. In: [MobIS 1999], S. 86 – 107.

[Rumbaugh 1993] *Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Lorensen, William:* Objektorientiertes Modellieren und Entwerfen. München 1993.

[Rumbaugh 1993b] *Rumbaugh, James:* Objects in the Constitution: Enterprise Modeling. In: Journal of Object-Oriented Programming. Januar 1993, S. 18 – 24.

[Rumbaugh 1996] *Rumbaugh, James:* A Search for Values: Attributes and Associations. In: Journal of Object-Oriented Programming, Juni 1996, S. 6 – 9.

[Rumbaugh 1996b] *Rumbaugh, James:* A State of Mind: Modeling Behavior. In: Journal of Object-Oriented Programming, Juli/August 1996, S. 6 – 12.

[Rumbaugh 1999] *Rumbaugh, James; Jacobson, Ivar; Booch, Grady:* The Unified Modeling Language Reference Manual. Reading, MA, 1999.

[Rumelhart 1975] *Rumelhart, David E.:* Notes on a Schema for Stories. In: [Bobrow 1975], S. 151 – 184.

[Rummler 1991] *Rummler, Geary; Brache, Alan:* Improving Performance. How to Manage the White Space on the Organization Chart. San Fancisco, Oxford 1991.

[Rupietta 1992] *Rupietta, Walter:* Organisatonsmodellierung zur Unterstützung kooperativer Vorgangsbearbeitung. In: Wirtschaftsinformatik, 34. Jahrgang, Heft 1, Februar 1992, 26 – 37.

[Rupp 2001] *Rupp, Chris; Sophist Group:* Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. München 2001.

[Rupper 1994] *Rupper, Peter:* Process Reengineering – eine Einführung. In: [Müller 1994], S. 9 – 11.

[Rüßmann 1994] *Rüßmann, Helmut:* Einführung in das Recht. Skript zur Vorlesung für den Fachbereich Wirtschaftswissenschaft. Universität des Saarlandes, Sommersemester 1994. (http://www.ruessmann.jura.uni-sb.de/rw20, Stand 20.5.2000)

[Saake 1993] *Saake, Gunter:* Objektorientierte Spezifikation von Informationssystemen. Habilitationsschrift, Stuttgart, Leipzig 1993.

[Schader 1998] *Schader, Martin; Korthaus, Axel (Hrsg.):* The Unified Modeling Language. Technical Aspects and Applications. Heidelberg 1998.

[Schäl 1996] *Schäl, Thomas:* Workflow Management Systems for Process Organisations. Berlin, Heidelberg, New York 1996.

[Schank 1972] *Schank, Roger C.:* Conceptual Dependency: A Theory of Natural Language Understanding. In: Cognitive Psychology, Band 3, 1972, S. 552 – 631.

[Schank 1975] *Schank, Roger C.:* Conceptual Information Processing. Amsterdam, Oxford 1975.

[Schank 1977] *Schank, Roger C.; Abelson, Robert P.:* Scripts, Plans, Goals and Understanding. An Inquiry into Human Knowledge Structures. Hillsdale, NJ, 1977.

[Schank 1981] *Schank, Roger C., Riesbeck, Christopher K.:* Inside Computer Understanding. Five Programs plus Miniatures. Hillsdale, NJ, 1981.

[Scharfenberg 1993] *Scharfenberg, Heinz (Hrsg.):* Strukturwandel in Management und Organisation. Neue Konzepte sichern die Zukunft. Baden-Baden 1993.

[Scharfenberg 1993b] *Scharfenberg, Heinz:* Von Taylor zum Team. In: [Scharfenberg 1993], S. 9 – 28.

[Scheer 1992] *Scheer, August-Wilhelm:* Architektur integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung. 2. verbesserte Auflage. Berlin, Heidelberg, New York, London u. a. 1992.

[Scheer 1995] *Scheer, A.-W.:* Wirtschaftsinformatik. Referenzmodelle für industrielle Geschäftsprozesse. Berlin, Heidelberg, New York u. a. 1995.

[Scheer 1997] *Scheer, A.-W.; Nüttgens, M.; Zimmermann, V.:* Objektorientierte Ereignisgesteuerte Prozesskette (oEPK) – Methode und Anwendung. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Iwi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes. Heft 141, Saarbrücken 1997.

[Scheer 1998] *Scheer, A. W.:* ARIS – Modellierungsmethoden, Metamodelle, Anwendungen. 3., völlig neu bearbeitete und erweiterte Auflage. Berlin, Heidelberg, New York 1998.

[Scheer 1998b] *Scheer, A. W.:* ARIS – Vom Geschäftsprozess zum Anwendungssystem. 3., völlig neu bearbeitete und erweiterte Auflage. Berlin, Heidelberg, New York 1998.

[Scherr 1993] *Scherr, Allan L.:* A New Approach to Business Processes. In: IBM Systems Journal, Band 32, Nr. 1, 1993, S. 80 – 98.

[Schienmann 1997] *Schienmann, Bruno:* Objektorientierter Fachentwurf. Ein terminologiebasierter Ansatz für die Konstruktion von Anwendungssystemen. Stuttgart 1997.

[Schlittgen 1999] *Schlittgen, Rainer:* Einführung in die Statistik. 9. Auflage, München 1999.

[Schmelzer 2002] *Schmelzer, Hermann J.; Sesselmann, Wolfgang:* Geschäftsprozessmanagement in der Praxis. 2., vollständig überarbeitete Auflage. München, Wien 2002

[Schmidt 1985] *Schmidt, Götz:* Grundlagen der Aufbauorganisation. Schriftenreihe „Der Organisator" Band 5. Gießen 1985.

[Schmidt 1985b] *Schmidt, Bernd:* Systemanalyse und Modellaufbau. Grundlagen der Simulationstechnik. Berlin, Heidelberg, New York u. a. 1985.

[Schmidt 1988] *Schmidt, Götz:* Methode und Techniken der Organisation. Schriftenreihe „Der Organisator" Band 1. 7. Auflage, Gießen 1988.

[Schmidt 1993] *Schmidt, Götz:* Organisation und EDV. In: [Scharfenberg 1993], S. 347 – 363.

[Schmidt 1997] *Schmidt, Günther:* Prozessmanagement. Modelle und Methoden. Berlin, Heidelberg, New York 1997.

[Schmitz 1992] *Schmitz, Paul:* Informationsverarbeitung. In: [Frese 1992], Spalte 958 – 967.

[Schmolke 1999] *Schmolke, Siegfried; Deitermann, Manfred:* Industrielles Rechnungswesen. 27., neu bearbeitete und erweiterte Auflage, korrigierter Nachdruck. Darmstadt 1999.

[Schneider 1998] *Schneider, Geri; Winters, Jason P.:* Applying Use Cases. A Practical Guide. Reading, MA, 1998.

[Schneider 2000] *Schneider, Uwe; Werner, Dieter:* Taschenbuch der Informatik. 3., völlig neu bearbeitete Auflage. München 2000.

[Schnieder 1993] *Schnieder, Eckehard:* Prozessinformatik. Automatisierung mit Rechensystemen, Einführung mit Petrinetzen. Braunschweig, Wiesbaden 1993.

[Scholz 1994] *Scholz, Rainer:* Geschäftsprozessoptimierung: Crossfunktionale Rationalisierung oder strukturelle Reorganisation. Dissertation, Bergisch Gladbach 1994.

[Scholz 1994b] *Scholz, Rainer; Vrohlings, Alwin:* Prozess-Struktur-Transparenz. In: [Gaitanides 1994], S. 37 – 56.

[Scholz-Reiter 1996] *Scholz-Reiter, Bernd; Stickel, Eberhard (Hrsg.):* Business Process Modelling. Berlin, Heidelberg, New York 1996.

[Schönthaler 1992] *Schönthaler, Frank; Nemeth, Tibor:* Software-Entwicklungswerkzeuge: Methodische Grundlagen. Stuttgart 1992.

[Schreyögg 1990] *Schreyögg, Georg:* Organisation III. Skript der Fernuniversität Hagen. Hagen 1990.

[Schubert 1976] *Schubert, Lenhart:* Extending the Expressive Power of Semantic Networks. In: Artificial Intelligence, Band 7, 1976, S. 163 – 198.

[Schubert 1979] *Schubert, Lenhart; Goebel, Randolph; Cercone, Nicholas J.:* The Structure and Organization of a Semantic Net for Comprehension and Inference. In: [Findler 1979], S. 121 – 175.

[Schüll 1999] *Schüll, Anke:* Ein Meta-Modell-Konzept zur Analyse von Geschäftsprozessen. Dissertation, Köln 1999.

[Schulte-Zurhausen 1999] *Schulte-Zurhausen, Manfred:* Organisation. 2., völlig überarbeitete und erweiterte Auflage. München 1999.

[Schwarzer 1994] *Schwarzer, Bettina; Krcmar, Helmut:* Business Redesign – Implikationen für das Informationsmanagement. In: [Krickl 1994], S. 79 – 92.

[Schwarzer 1995] *Schwarzer, Bettina; Krcmar, Helmut:* Zur Prozessorientierung des Informationsmanagements. In: Wirtschaftsinformatik, Band 37, 1995, S. 33 – 39.

[Schwarzer 1996] *Schwarzer, Bettina; Krcmar, Helmut:* Wirtschaftsinformatik. Grundzüge der betrieblichen Datenverarbeitung. Stuttgart 1996.

[Schwegmann 2000] *Schwegmann. Ansgar:* Management komplexer Prozessmodelle. In: HMD. Praxis der Wirtschaftsinformatik, Heft 213, 2000, S. 80 – 89.

[Schwetz 1984] *Schwetz, Roland:* Büroökonomie. In: [Witte 1984], S. 38 – 56.

[Schwetz 1993] *Schwetz, Roland:* Wandel in der Organisationsarbeit – Der Organisator wird nicht überflüssig. In: [Scharfenberg 1993], S. 293 – 321.

[Sciore 1989] *Sciore, Edward:* Object Specialization. In: ACM Transactions on Information Systems, Band 7, Nr. 2, 1989, S. 103 – 122.

[Scott-Morton 1991] *Scott-Morton, M. S. (Hrsg.):* The Corporation of the 1990s: information technology and organizational transformation. New York 1991.

[Scott-Morton 1991b] *Scott-Morton, M. S.:* Introduction. In: [Scott-Morton 1991], S. 1 – 25.

[Seiffert 1992] *Seiffert, Helmut; Radnitzky, Gerard (Hrsg.):* Handlexikon zur Wissenschaftstheorie. München 1992.

[Selic 1994] *Selic, Bran; Gullekson, Garth; Ward, Paul T.:* Real-Time Object-Oriented Modeling. New York u. a. 1994.

[Sempf 1993] *Sempf, Ulrich:* Steigerung der organisatorischen Leistungsfähigkeit durch Geschäftsprozess-Optimierung. In: [Scharfenberg 1993], S. 365 – 381.

[Shaw 1996] *Shaw, Mary; Garlan, David:* Software Architecture. Perspectives on an Emerging Discipline. Upper Saddle River, NJ, 1996.

[Sheppard 1994] *Sheppard, Deri:* An Introduction to Formal Specification with VDM and Z. London, New York, St. Louis 1994.

[Shlaer 1988] *Shlaer, Sally; Mellor, Stephen J.:* Object-Oriented Systems Analysis. Modeling the World in Data. Englewood Cliffs, NJ, 1988.

[Shlaer 1992] *Shlaer, Sally; Mellor, Stephen J.:* Object Lifecycles. Modeling the World in States. Englewood Cliffs, NJ, 1992.

[Shlaer 1995] *Shlaer, Sally; Lang, Neil:* Dependence between Attributes. In: Report on Analysis & Design, Band 2, Nr. 4, November/Dezember 1995, S. 18, 19, 23.

[Siebert 2001] *Siebert, Reiner:* Workflowmodellierung für organisationsübergreifende Prozesse bei der DaimlerChrysler AG. Vortrag auf der [IAO 2001].

[Sims 1994] *Sims, Oliver:* Business Objects. Delivering Cooperative Objects for Client-Server. London 1994.

[Sinz 1997] *Sinz, Elmar:* Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Bamberger Beiträge zur Wirtschaftsinformatik Nr. 41, Universität Bamberg, Juli 1997.

[Smith 1977] *Smith, John Miles; Smith, Diane C. P.:* Database Abstractions: Aggregation and Generalisation. In: ACM Transactions on Database Systems, Band 2, Nr. 2, Juni 1977, S. 105 – 133.

[Smith 1977b] *Smith, John Miles; Smith, Diane C. P.:* Database Abstraction: Aggregation. In: Communication of the ACM, Juni 1977, Band 20, Nr. 6, S. 405 – 413.

[Smith 1981] *Smith, E. E.; Medin, D. L.:* Categories and Concepts. Cambridge, MA, 1981.

[Sommerville 1996] *Sommerville, Ian:* Software Engineering. 5. Auflage. Harlow, England, 1996.

[Sommerville 1997] *Sommerville, Ian; Kotonya, Gerald:* Requirements Engineering. Chichester 1997.

[Sowa 1984] *Sowa, John F.:* Conceptual Structure: Information Processing in Mind and Machine. Reading, MA, 1984.

[Sowa 1991] *Sowa, John F. (Hrsg):* Principles of Semantic Networks. San Mateo, CA, 1991.

[Sowa 1992] *Sowa, John F.; Zachman, John A.:* Extending and formalizing the framework for information systems architecture. In: IBM Systems Journal, Band 31, Nr. 3, S. 590 – 616.

[Spivey 1992] *Spivey, J. M.:* The Z Notation. A Reference Manual. New York, London, Toronto 1992.

[Stachowiak 1973] *Stachowiak, Herbert:* Allgemeine Modelltheorie. Wien, New York 1973.

[Staehle 1994] *Staehle, Wolfgang H.:* Management. 7., von *Jörg Sydow* und *Peter Conrad* überarbeitete Auflage. München 1994.

[Stahlknecht 1995] *Stahlknecht, Peter:* Einführung in die Wirtschaftsinformatik. 7. Auflage. Berlin, Heidelberg, New York 1995.

[Stachowiak 1992] *Stachowiak, Herbert:* Modell. In: [Seiffert 1992], S. 219 -222.

[Staud 1999] *Staud, Josef:* Geschäftsprozessanalyse mit Ereignisgesteuerten Prozessketten. Berlin, Heidelberg, New York 1999.

[Staud 2001] *Staud, Josef:* Geschäftsprozessanalyse mit Ereignisgesteuerten Prozessketten. 2., überarbeitete und erweiterte Auflage, Berlin, Heidelberg, New York 2001.

[Stefik 1986] *Stefik, Mark; Bobrow, Daniel G.:* Object-Oriented Programming: Themes and Variations. In: The AI Magazine, Band 6, Nr. 4, S. 40 – 62.

[Stein 1997] *Stein, Wolfgang:* Objektorientierte Analysemethoden. Vergleich, Bewertung, Auswahl. 2., völlig neu bearbeitete Auflage. Heidelberg, Berlin, Oxford 1997.

[Steinbuch 1974] *Steinbuch, K.; Weber, W.:* Taschenbuch der Informatik. Band 2. Struktur und Programmierung von EDV-Systemen. Berlin, Heidelberg, New York 1974.

[Steinbuch 1995] *Steinbuch, Pitter A.:* Organisation. 9., überarb. und erweiterte Auflage. Ludwigshafen 1995.

[Steinbuch 1998] *Steinbuch, Pitter A.:* Prozessorganisation – Business Reengineering – Beispiel R/3. Ludwigshafen 1998.

[Steinmann 1991] *Steinmann, Horst; Schreyögg, Georg:* Management. Grundlagen der Unternehmensführung. 2., durchgesehene Auflage, Wiesbaden 1991.

[Stillings 1995] *Stillings, Neil A.; Weisler, Steven E.; Chase, Christopher H.; Feinstein, Mark H.; Garfield, Jay L.; Rissland, Edwina L.:* Cognitive Science. An Introduction. 2nd Ed., Cambridge, MA, London, England, 1995.

[Stork 1993] *Stork, Hans-Georg:* Methoden der imperativen Programmierung. Stuttgart 1993.

[Stoyan 1991] *Stoyan, Herbert:* Programmiermethoden der künstlichen Intelligenz. Band 2. Berlin u. a. 1991.

[Strahringer 1998] *Strahringer, Susanne:* Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips. In: [Pohl 1998].

[Striening 1988] *Striening, Hans-Dieter:* Prozess-Management. Versuch eines integrierten Konzeptes situationsadäquater Gestaltung von Verwaltungsprozessen – dargestellt am Beispiel in einem multinationalen Unternehmen – IBM Deutschland GmbH. Dissertation, Bern, New York, Paris 1988.

[Stroustrup 1992] *Stroustrup, Bjarne:* Die C++ Programmiersprache. Erweitert um Entwürfe zur ANSI-/ISO-Standardisierung. 2., überarb. Auflage. Bonn u. a. 1992.

[Stroustrup 2000] *Stroustrup, Bjarne:* Die C++ Programmiersprache. 4., akualisierte und erweiterte Auflage. Bonn u. a. 2000.

[STUD-JUR 1994] *STUD-JUR Nomos Textausgaben:* Zivilrecht, Wirtschaftsrecht. 3. Auflage, Baden-Baden 1994.

[Szyperski 1961] *Szyperski, Norbert:* Analyse der Merkmale und Formen der Büroarbeit. In: [Kosiol 1961], S. 77 – 132.

[Taivalsaari 1996] *Taivalsaari, Antero:* On the Notion of Inheritance. In: ACM Computing Surveys, Band 28, Nr. 3, September 1996. S. 438 – 479.

[Tanenbaum 1994] *Tanenbaum, Andrew S.:* Moderne Betriebssysteme. München 1994.

[Taudes 1996] *Taudes, Alfred; Cilek, Peter; Natter, Martin:* Ein Ansatz zur Optimierung von Geschäftsprozessen. In: [Vossen 1996], S. 177 – 189.

[Taylor 1995] *Taylor, David A.:* Business Engineering with Object Technology. New York 1995.

[Thom 1992] *Thom, Norbert:* Stelle, Stellenbildung und -besetzung. In: [Frese 1992], Spalte 2321 – 2333.

[Thrampoulidis 1995] *Thrampoulidis, K. X.; Agavanakis, K. N.:* Object Interaction Diagram: A New Technique in Object-Oriented Analysis and Design. In: Journal of Object-Oriented Programming. Juni 1995, S. 25 – 32.

[Turowski 1996] *Turowski, Klaus:* Prozessorientierung in der Produktionsplanung und -steuerung. In: [Vossen 1996], S. 209 – 223.

[Turowski 2001] *Turowski, Klaus:* Spezifikation und Standardisierung von Fachkomponenten. Wirtschaftsinformatik-Online-Preprint, 2001. (http://www.wirtschaftsinformatik.de, Stand 07.05.2001)

[Turner 1994] *Turner, J. G.; McCluskey, T. L.:* The Construction of Formal Specifications. An Introduction to the Model-Based and Algebraic Approaches. London u. a. 1994.

[UMLN 1997] *Rational Software Corporation*: UML Notation Guide. Version 1.1, ad/97-08-05. Santa Clara, CA, 1997.

[UMLS 1997] *Rational Software Corporation*: UML Semantics. Version 1.1, ad/97-08-04. Santa Clara, CA, 1997.

[Vester 1993] *Vester, Frederic*: Neuland des Denkens. Vom technokratischen zum kybernetischen Zeitalter. 8. Auflage, Stuttgart 1993.

[Vetter 1995] *Vetter, Max:* Objektmodellierung. Stuttgart 1995.

[V-Modell 1997] *Vorgehensmodell (V-Modell):* Allgemeiner Umdruck 250. Entwicklungsstandard für IT-Systeme des Bundes (EStdIT). Juni 1997.

[Vlissides 1999] *Vlissides, John:* Entwurfsmuster anwenden. Bonn 1999.

[Völkner 1998] *Völkner, Peer:* Modellbasierte Planung von Geschäftsprozessabläufen: Entwicklung eines Entscheidungsunterstützungssystems auf Grundlage objektorientierter Simulation. Dissertation, Wiesbaden 1998.

[von Colbe 1988] *Busse von Colbe, Walther; Laßmann, Gert:* Betriebswirtschaftstheorie. Band 1. Grundlagen, Produktions- und Kostentheorie. 4., überarb. und ergänzte Auflage. Berlin, Heidelberg, New York, u. a. 1988.

[Vossen 1996] *Vossen, Gottfried; Becker, Jörg (Hrsg.):* Geschäftsprozessmodellierung und Workflow-Management. Modelle, Methoden, Werkzeuge. Bonn, Albany, Belmont 1996.

[Waldén 1995] *Waldén, Kim; Nerson, Jean-Marc:* Seamless Object-Oriented Software Architecture. Analysis and Design of Reliable Systems. New York 1995.

[Wallrabe 1997] *Wallrabe, Arne; Oestereich, Bernd:* Smalltalk für Ein- und Umsteiger. München, Wien 1997.

[Ward 1991] *Ward, Paul T.; Mellor, Stephen J.:* Strukturierte Systemanalyse von Echtzeit-Systemen. München 1991.

[Warmer 1999] *Warmer, Jos; Kleppe, Anneke:* The Object Constraint Language. Precise Modeling with UML. Reading, MA, 1999.

[Wassermann 2001] *Wassermann, Otto:* Das intelligente Unternehmen: mit der Wassermann Supply Chain Idee den globalen Wettbewerb gewinnen. 4. Auflage. Berlin, Heidelberg, New York 2001.

[Weber 1980] *Weber, Max:* Wirtschaft und Gesellschaft. Grundriß der verstehenden Soziologie. Nachdruck der Studienausgabe der fünften, von Johannes Winkelmann revidierten Auflage von 1972. Tübingen 1990.

[Webster 1993] *Webster's New Encyclopedic Dictionary.* New York 1993.

[Wedekind 1973] *Wedekind, Hartmut:* Systemanalyse. Die Analyse von Anwendungssystemen für Datenverarbeitungsanlagen. München 1973.

[Wedekind 1992] *Wedekind, Hartmut:* Objektorientierte Schemaentwicklung. Ein kategorialer Ansatz für Datenbanken und Programmierung. Mannheim, Wien, Zürich 1992.

[Wegner 1988] *Wegner, Peter; Zdonik, Stanley B.:* Inheritance as an Incremental Modification Mechanism. In: Proceedings ECOOP 1988. Lecture Notes of Computer Science Nr. 322. Berlin, Heidelberg, New York 1988, S. 55 – 77.

[Wegner 1987] *Wegner, Peter:* Dimensions of Object-Based Language Design. In: OOPSLA '87 Proceedings. 4. – 8. Oktober 1987, S. 168 – 182.

[Weikum 1997] *Weikum, Gerhard; Wodtke, Dirk; Kotz-Dittrich, Angelika; Muth, Peter; Weißenfels, Jeanine:* Spezifikation, Verifikation und verteilte Ausführung von Workflows in Mentor. In: Informatik Forschung und Entwicklung, Band 12, 1997, S. 61 – 71.

[Weinberg 1975] *Weinberg, Gerald M.:* An Introduction to General Systems Thinking. London, Sydney, Toronto 1975.

[Weise 1999] *Weise, Thomas; Mandl, Peter; Mittasch, Christian; Bauer, Niklai; Hauptmann, Jens; Schill, Alexander:* Lightweight Workflows. In: HMD. Praxis der Wirtschaftsinformatik, Heft 210, Dezember 1999, S. 83 – 100.

[Wenzel 1997] *Wenzel, Joachim:* Entwurf einer Modellierungssprache zur Beschreibung von Geschäftsprozessen im Rahmen der Unternehmensmodellierung. Diplomarbeit, Koblenz 1997.

[Wenzel 1999] *Wenzel, Paul (Hrsg.):* Betriebswirtschaftliche Anwendungen mit SAP R/3. 3., überarbeitete Auflage, Braunschweig, Wiesbaden 1999.

[Wettstein 1993] *Wettstein, Horst:* Systemarchitektur. München 1993.

[WfMC 1996] *Workflow Management Coalition:* Workflow Management Coalition Terminology & Glossary. Document Number WFMC-TC-1011. Issue 2.0, Brüssel 1996.

[Wieczerzycki 1996] *Wieczerzycki, Waldemar:* Process Modelling and Execution in Workflow Management Systems by Event-Driven Versioning. In: [Scholz-Reiter 1996], S. 43 – 66.

[Wilkonson 1995] *Wilkinson, Nancy M.:* Using CRC Cards. New York 1995.

[Wild 1966] *Wild, Jürgen:* Grundlagen und Probleme der betriebswirtschaftlichen Organisationslehre. Berlin 1966.

[Wild 1970] *Wild, Jürgen:* Input-, Output- und Prozessanalyse von Informationssystemen. In: Zeitschrift für betriebswirtschaftliche Forschung, 22. Jahrgang, 1970, S. 50 – 72.

[Winant 1996] *Winant, Becky; Frankel, Mike:* The Event Dictionary: What your Methodolist forgets to tell you. (http://www.espritinc.com, Stand

25.03.2000) Ursprünglich erschienen im Journal of Object-Oriented Programming, Oktober 1996.

[Winant 1996b] *Winant, Becky; Frankel, Mike:* The Event Dictionary brings Precision to Use Cases. (http://www.espritinc.com, Stand 25.03.2000) Ursprünglich erschienen im Object Magazine, November 1996.

[Winant 1997] *Winant, Becky; Frankel, Mike:* Object State Model Mystery Solved Using a Key Event Dictionary. (http://www.espritinc.com, Stand 25.03.2000) Ursprünglich erschienen im Journal of Object-Oriented Programming Object Magazine, März 1997.

[Winograd 1986] *Winograd, Terry; Flores, Fernando:* Understanding Computers and Cognition. A new Foundation for Design. Reading, MA, 1986.

[Winograd 1987] *Winograd, Terry:* A Language/Action Perspective on the Design of Cooperative Work. In: Human Computer Interaction, Band 3, Nr. 1, 1987, S. 3 – 30.

[Winston 1987] *Winston, Morton E.; Chaffin, Roger; Hermann, Douglas:* A Taxonomy of Part-Whole Relations. In: Cognitive Science, Band 11, 1987, S. 417 – 444.

[Winston 1992] *Winston, Patrick Henry:* Artificial Intelligence. 3rd Edition, Reading, MA, 1992.

[Wirfs-Brock 1993] *Wirfs-Brock, Rebecca; Wilkerson, Brian; Wiener, Lauren:* Objektorientiertes Software-Design. München 1993.

[Wiswede 1992] *Wiswede, Günter:* Gruppen und Gruppenstrukturen. In: [Frese 1992], Spalte 735 – 754.

[Witte 1984] *Witte, E. (Hrsg.):* Bürokommunikation. Ein Beitrag zur Produktivitätssteigerung. Berlin, Heidelberg, New York 1984.

[WitteT 1994] *Witte, Thomas; Claus, Thorsten; Helling, Klaus:* Simulation von Produktionssystemen mit SLAM. Eine praxisorientierte Einführung. Bonn 1994.

[Wittgenstein 1994] *Wittgenstein, Ludwig:* Ein Reader. Frankfurt am Main 1994.

[Wodke 1995] *Wodke, Dirk; Kotz-Dittrich, Angelika; Muth, Peter; u. a.:* Mentor*:* Entwurf einer Workflow-Management-Umgebung basierend auf State- und Activitycharts. In: BTW 6. Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, Dresden, 1995, S. 71 – 90. (http://paris.cs.uni-sb.de/~mentor/, Stand 18.12.1999)

[Wöhe 1986] *Wöhe, Günter:* Einführung in die allgemeine Betriebswirtschaftslehre. 16., überarbeitete Auflage, München 1986.

[Wolff 1998] *Wolff, M.-R.; Linssen, Oliver:* Der innovative Gehalt des Geschäftsprozessmanagment. Arbeitsberichte zum Fachgebiet Betriebswirtschaftslehre und Wirtschaftsinformatik 98-2, Universität Wuppertal, August 1998.

[Woods 1975] *Woods, William A.:* What's in a Link: Foundation for Semantic Networks. In: [Bobrow 1975], S. 35 – 81.

[Wordsworth 1992] *Wordsworth, J. B.:* Software Development with Z. A Practical Approach to Formal Methods in Software Engineering. Wokingham, England, Reading, MA, Menlo Park, CA, 1992.

[Wordsworth 1995] *The Wordsworth Dictionary of Science & Technology.* Ware, Hertfordshire, 1995.

[Yourdon 1992] *Yourdon, Edward:* Moderne Strukturierte Analyse. Attenkirchen 1992.

[Yourdon 1994] *Yourdon, Edward:* Object-Oriented Systems Design. An Integrated Approach. Englewood Cliffs, NJ, 1994.

[Yourdon 1996] *Yourdon, Edward; Nevermann, Peter; Oppel, Karin; Whitehead, Katharine:* Mainstream Objects. München, London, Mexiko u. a. 1996.

[Zachman 1987] *Zachman, John, A.:* A framework for information systems architecture. In: IBM Systems Journal, Band 26, Nr. 3, S. 276 – 292.

[Zapf 2000] *Zapf, Michael; Heinzl, Armin:* Ansätze zur Integration von Petri-Netzen und objektorientierten Konzepten. In: Wirtschaftsinformatik, Band 42, Nr. 1, 2000, S. 36 – 46.

[Zelewski 1994] *Zelewski, Stephan:* Grundlagen. In*:* [Corsten 1994], S. 1 – 140.

[Zimmermann 1994] *Zimmermann, Hubert; Katzy, Bernhard R.; Plötz, Armin J.; Tanner, Hans-Rudolf:* Prozessorientierte Organisationsstrukturen. Integration von Prozess- und Organisationsstrukturen durch Unternehmensmodellierung. In: [Müller 1994], S. 107 – 120.

[ZimmermannV 1999] *Zimmermann, Volker:* Objektorientiertes Geschäftsprozessmanagement. Integrationsansatz – Modellierungsmethode – Anwendungsbeispiel. Wiesbaden 1999.

[Züllighoven 1998] *Züllighoven, Heinz:* Das objektorientierte Konstruktionshandbuch nach dem Werkzeug-&-Material-Ansatz. Heidelberg 1998.