UNIVERSITÄT KOBLENZ · LANDAU

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

ULRICH FRANK
MICHAEL PRASSE

EIN BEZUGSRAHMEN ZUR BEURTEILUNG OBJEKTORIENTIERTER MODELLIERUNGSSPRACHEN - VERANSCHAULICHT AM BEISPIEL VON OML UND UML

September 1997

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 6

UNIVERSITÄT KOBLENZ · LANDAU

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

ULRICH FRANK
MICHAEL PRASSE

EIN BEZUGSRAHMEN ZUR BEURTEILUNG OBJEKTORIENTIERTER MODELLIERUNGSSPRACHEN - VERANSCHAULICHT AM BEISPIEL VON OML UND UML

September 1997

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 6

Die Arbeitsberichte des Instituts für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i.d.R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The "Arbeitsberichte des Instituts für Wirtschaftsinformatik" comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Anschrift der Verfasser/ Address of the authors:

Prof. Dr. Ulrich Frank
Dipl. Inf. Michael Prasse
Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
D-56075 Koblenz

Arbeitsberichte des Instituts für Wirtschaftsinformatik Herausgegeben von / Edited by:

Prof. Dr. Ulrich Frank
Prof. Dr. J. Felix Hampe

Bezugsquelle / Source of Supply:

Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
56075 Koblenz

Tel.: 0261-9119-480
Fax: 0261-9119-487
Email: iwi@uni-koblenz.de
WWW: http://www.uni-koblenz.de/~iwi

"Sprache ist mein Instrument, aber zur gleichen Zeit auch mein Problem."

Humberto R. Maturana

Abstract

Der vorliegende Beitrag ist u.a. durch die zur Zeit laufenden Bemühungen der Object Management Group (OMG) motiviert, objektorientierte Modellierungssprachen zu standardisieren. Angesichts der großen ökonomischen Bedeutung einer derartigen Standardisierung ist eine sorgfältige Berücksichtigung der an eine Modellierungssprache zu stellenden Anforderungen zu fordern. Der Entwurf geeigneter Beurteilungskriterien für objektorientierte Modellierungssprachen ist allerdings eine delikate Aufgabe. So sind mit den verschiedenen Tätigkeiten im Rahmen der Software-Entwicklung unterschiedliche, zum Teil widersprüchliche Anforderungen an eine Modellierungssprache verbunden. Darüber hinaus ist die Qualität einer Sprache nicht unabhängig von individuellen Wahrnehmungsmustern und Konzeptualisierungspräferenzen der jeweiligen Benutzer zu sehen.

In dem vorliegenden Bericht wird zunächst ein Bezugsrahmen zur Beurteilung von Modellierungssprachen entwickelt, in dem die verschiedenen Anforderungsebenen - von eher subjektiven, wie etwa der Anschaulichkeit einer Notation, bis zu software-technischen, wie etwa Vollständigkeit und Korrektheit der Sprachbeschreibung - dargestellt werden. Anschließend wird dieser Bezugsrahmen zur vergleichenden Betrachtung von OML und UML verwendet.

Inhaltsverzeichnis

1 Einleitung

Der Entwurf von Software erfordert eine angemessene Erfassung und Beschreibung ausgewählter Eigenschaften der jeweiligen Anwendungsdomäne. Da eine solche Beschreibung i.d.R. nicht allein im Kreis von Software-Entwicklern entsteht, sondern Anwender und Domänenexperten wesentlich an ihrer Entstehung zu beteiligen sind, sind Programmiersprachen oder auch formale Spezifikationssprachen zu ihrer Erstellung kaum geeignet. Sie sind deutlich an den Randbedingungen der Implementierung orientiert und sperren sich deshalb gegen Darstellungen, die von allen Beteiligten als verständlich und authentisch empfunden werden. Auf der anderen Seite ist auch die Beschränkung auf eine natürliche Sprache, sei es in Form der Umgangssprache und/oder einer dedizierten Fachsprache, unzureichend: Schließlich sollen Beschreibungen von Anwendungsdomänen Vorgaben für die Implementierung liefern, so daß daraus erwachsende Besonderheiten auch berücksichtigt werden müssen. Vor dem Hintergrund dieses Spannungsfelds ist die *konzeptuelle Modellierung* entstanden. Als ihr vorrangiges Anliegen wird gern die "natürliche" Abbildung des jeweils betrachteten Realitätsausschnitts oder gedanklichen Entwurfs genannt: "... descriptions of a world enterprise/slice of reality which correspond directly and naturally to our own conceptualizations of the object of these descriptions." ([MyLe84], S. 11) Eine solche Kennzeichnung ist jedoch allenfalls als didaktisch motivierte Überzeichnung zu akzeptieren: Eine natürliche Beschreibung in diesem Sinn dürfte i.d.R. mit einer natürlichen Sprache besser zu leisten sein. Gegenstand und Ziel der konzeptuellen Modellierung sind deshalb besser wie folgt zu charakterisieren: Die konzeptuelle Modellierung dient einer allen Beteiligten an einem Software-Entwicklungsprozeß verständlichen Darstellung einer Anwendungsdomäne. Die Darstellung beschränkt sich dabei auf die für die Erstellung des beabsichtigten Systems wesentlichen Aspekte dieser Domäne. Zusätzlich zu dieser Abstraktion muß ein konzeptuelles Modell auch den Randbedingungen Rechnung tragen, die durch die in einer späteren Phase zu verwendenden Implementierungssprachen entstehen. Wesentliche Voraussetzung für das Umsetzen dieses Anspruchs ist eine geeignete *Modellierungssprache*.

Die *objektorientierte* konzeptuelle Modellierung - kurz: *objektorientierte Modellierung* - steht im Ruf, den skizzierten Konflikt zwischen Verständlichkeit und gleichzeitiger Berücksichtigung software-technischer Anforderungen besonders gut überwinden zu helfen. Objektorientierte Modellierung ist wesentlich durch den Einsatz objektorientierter Modellierungssprachen gekennzeichnet. Sie sind seit längerem Gegenstand intensiver Forschungsbemühungen. Dementsprechend groß ist die Zahl einschlägiger Ansätze (ein Überblick findet sich in [Fra97]), die einhergeht mit einer Fülle von - zum Teil kommerziell vertriebenen - Werkzeugen, die auf einer Reihe unterschiedlicher Modellierungssprachen beruhen. Diese Vielzahl der Ansätze fördert die Verunsicherung potentieller Anwender objektorientierter Modellierungssprachen, da ihr Einsatz die mühsame Bewertung der Alternativen voraussetzt und dessen ungeachtet erhebliche Risiken für die zu tätigenden Investitionen darstellt.

Auch aus wissenschaftlicher Sicht ist die skizzierte Situation wenig erfreulich. Die verschiedenen Modellierungsansätze stehen weitgehend isoliert nebeneinander. Zum erheblichen Teil stammen sie aus einer Grauzone zwischen Forschung und kommerzieller Verwertung. Damit verbunden ist eine wenig einheitliche und zum Teil unzureichende Terminologie (in [Fra97] wird eine Basisterminologie vorgestellt). Die Berücksichtigung einzelner Ansätze auch in der einschlägigen Forschung - hier ist vor allem an die Informatik und die Wirtschaftsinformatik zu denken - wird mehr und mehr durch deren Verbreitung und ihre daran anknüpfende wirtschaftliche Bedeutung bestimmt. Daneben fällt auf, daß sich die Forschung zum erheblichen Teil auf die Umsetzung oder Rekonstruktion einzelner Modellierungssprachen - etwa für den Entwurf korrespondierender Werkzeuge - konzentriert. Daraus ergeben sich bedenkliche Konsequenzen. Es ist für eine wissenschaftliche Disziplin kaum hinzunehmen, wenn wesentliche Entscheidungen über die Auswahl des Untersuchungsgegenstands sowie der jeweils einzusetzenden Instrumente vorrangig außerwissenschaftlich festgelegt werden. Damit zusammenhängend ist es für die wissenschaftliche Untersuchung von Modellierungssprachen und ihren Einsatzbedingungen unerläßlich, nicht bei der Betrachtung einzelner, wie auch immer entstandener Entwürfe zu verhar-

ren. Stattdessen ist es erforderlich, begründete Kriterien für die Beurteilung und damit: Weiterentwicklung von Modellierungssprachen einzuführen - nur so ist es langfristig möglich, wissenschaftlichen Erkenntnisfortschritt zu identifizieren.

Mit dem vorliegenden Arbeitsbericht unternehmen wir den Versuch, einen Beitrag zur dringend benötigten, allerdings - wie sich noch zeigen wird - äußerst diffizilen Beurteilung objektorientierter Modellierungssprachen zu leisten. Den aktuellen Anlaß für diesen Versuch bilden die gegenwärtigen Bemühungen der Object Management Group (OMG), einen Standard für objektorientierte Modellierungssprachen zu definieren. Im Juni 1996 veröffentlichte die OMG einen "Request for Proposals" ([OMG96b]), in dem Vorschläge für standardisierungsfähige Sprachen eingefordert wurden. Dabei fällt auf, daß die Ausschreibung allenfalls rudimentäre Anforderungen enthält, die zudem teilweise widersprüchlich sind (eine kritische Darstellung dieser Vorschläge findet sich in [Fra97]). Bis zum Einsendeschluß am 17. Januar 1997 sind sechs Vorschläge bei der OMG eingegangen. Mittlerweile haben sich die Einsender unter Mitwirkung der OMG darauf geeinigt, gemeinsam an einer Version zu arbeiten, die wesentlich auf der "Unified Modeling Language" (UML) der Firma Rational basiert ([Rat97n], S. 13 ff.). Angesichts des bisherigen Procedere der OMG ist letztlich ein pragmatischer, wesentlich von den wirtschaftlichen Interessen der beteiligten Unternehmen abhängender Standard zu erwarten. Dies ist ein weiterer Grund, der Frage nachzugehen, wie Anforderungen an Modellierungssprachen aussehen, die in wissenschaftlich akzeptabler Weise begründet sind. Dazu werden wir einen Bezugsrahmen entwickeln, der wichtige Beurteilungskriterien enthält, deren Anwendung allerdings zum Teil eine sorgfältige Interpretation erfordert. Es handelt sich also nicht um eine Metrik für objektorientierte Programmiersprachen, die die Qualität einer Sprache auf die Ausprägung einer Meßskala abzubilden gestattet.

Zur Veranschaulichung des Bezugsrahmens werden wir ihn beispielhaft auf zwei ausgewählte Modellierungssprachen anwenden. Die UML haben wir - in der zur Zeit verfügbaren Version - ausgewählt, weil sie eine exponierte Stellung erlangt hat. So basiert sie einerseits auf drei in der Vergangenheit besonders populären Ansätzen ([Boo94], [Rum91], [Jac92]), andererseits deutet das Standardisierungsverfahren der OMG darauf hin, daß die UML den zukünftigen Standard wesentlich prägen wird. Vergleichend betrachten wir die "OPEN Modeling Language" (OML), die von verschiedenen Forschungseinrichtungen im Rahmen der Initiative OPEN (Object-oriented Process, Environment, and Notation) entworfen wurde. Auch wenn die Mitglieder der Initiative eine Standardisierung ihres Vorschlags anstreben ([FiHe97], S. 4), sucht man ihn vergeblich in der Liste der eingereichten Entwürfe. Dies liegt allein daran, daß OPEN die Anforderungen der OMG an eine kommerzielle Umsetzung nicht erfüllen konnte: Jeder Einsender muß seine Einwilligung dazu erklären, die vorgeschlagene Technologie innerhalb von 12 Monaten nach der Annahme durch die OMG am Markt anzubieten ([OMG96], S. 24). Insofern ist die Berücksichtigung der OML auch ein Versuch, das Vorgehen der OMG zu relativieren: Es geht offensichtlich nicht um die Auswahl der per se am besten geeigneten Modellierungssprache. Vielmehr wird die Menge der betrachteten Alternativen von Beginn an auf solche eingeschränkt, die den - nachvollziehbaren - Verwertungsinteressen der OMG gerecht werden.

2 Randbedingungen der Beurteilung objektorientierter Modellierungssprachen

Anders als natürliche Sprachen sind Modellierungssprachen Kunstsprachen, die als Mittel zur Erfüllung eines bestimmten Zwecks entworfen werden. Im Unterschied zu anderen Kunstsprachen, die vorrangig maschinell interpretiert werden oder allein in einer Gruppe einschlägig spezialisierter Fachleute Verwendung finden, weisen Modellierungssprachen allerdings eine deutliche Gemeinsamkeit mit natürlichen Sprachen auf: Sie dienen u.a. der Verständigung von Menschen mit ganz unterschiedlichem beruflichem Hintergrund. Modellierungssprachen decken damit ein weites und komplexes Feld

ab. Um einen Eindruck von dieser Komplexität zu vermitteln, werden wir einen Blick auf einige Aspekte werfen, die für die wissenschaftliche Untersuchung von Sprache bedeutsam sind.

2.1 Sprache: Ein vielschichtiges Phänomen

Die menschliche Fähigkeit zu sprechen wird gemeinhin als eine "unhintergehbare Kompetenz" ([Lor96], S. 49) angesehen. Dessen ungeachtet sind wir in der Lage, über die Sprache und ihre Verwendung zu sprechen, was sich u.a. in der gängigen Unterscheidung zwischen Objektsprache und Metasprache (man spricht auch von "Schemaebene" und "Aktualisierungsebene") ausdrückt. Die Untersuchung von Sprache hat eine lange Tradition, sowohl in der Philosophie wie auch in der dedizierten Sprachwissenschaft Linguistik. Von zentraler Bedeutung dafür ist die (Re)konstruktion von Sprachstrukturen, etwa mittels einer Grammatik. Sie beschreibt die Regeln, nach denen als korrekt empfundene Sätze einer Sprache gebildet werden. Angesichts der Vielzahl natürlicher Sprachen wird seit langem die Frage diskutiert, ob es sprachliche Universalien gibt, also bestimmte Begriffe, die in allen natürlichen Sprachen enthalten sind. Deutlich darüber hinaus geht die Frage nach der Existenz und dem Aufbau der "Universalgrammatik". Die Suche nach solchen allen Menschen verfügbaren Grundregeln des Sprechens ist durch die Annahme motiviert, daß Spracherwerb auf eine angeborene Fähigkeit zurückgeht. Für Chomsky, auf den der Begriff (einer noch zu entwickelnden) Universalgrammatik wesentlich zurückgeht, stellt ein solches Regelwerk zugleich "Hypothesen" dar, "die für die angeborene Struktur des menschlichen Geistes relevant sind." ([Cho77], S. 49)

Auf einer anderen Betrachtungsebene wird mitunter - ausgelöst durch interkulturelle Sprachstudien von Whorf in den dreißiger Jahren - die Ansicht vertreten, daß Sprache einen "allgemeinen Bezugsrahmen" ([Hen75], S. 10) darstellt, der das Denken derer nachhaltig beeinflußt, die sie "gewohnheitsmäßig gebrauchen" (ibid.). Im Zusammenhang zu solchen Überlegungen ist eine verbreitete Auffassung zu sehen, wonach die Untersuchung der Sätze einer Sprache auch ihren Gebrauch und die mit ihm assoziierten Handlungen in einem lebensweltlichen Kontext zu berücksichtigen hat. Wittgenstein hat dafür den Begriff "Sprachspiel" geprägt: "Das Wort »Sprachspiel« soll hier hervorheben, daß das Sprechen der Sprache ein Teil ist einer Tätigkeit, oder einer Lebensform." ([Wit71], 23)

Unzulänglichkeiten natürlicher Sprachen haben den Entwurf zahlreicher Kunstsprachen für die sprachliche Kommunikation zwischen Menschen motiviert. Dabei ist einerseits an *Wissenschaftssprachen* zu denken, die darauf zielen, die Irreführungen natürlicher Sprachen zu vermeiden[1]. Es handelt sich dabei entweder um formale Sprachen oder um Sprachen mit normierter Syntax und Semantik, die einer formalen Rekonstruktion eher zugänglich sind als natürliche Sprachen. Daneben haben die offenkundigen Schwierigkeiten, auf die viele Menschen beim Erlernen einer fremden Sprache stoßen, Versuche motiviert, die Verständigung zwischen den Völkern durch gemeinsame, relativ leicht erlernbare Referenzsprachen zu erleichtern. Die bekannteste dieser sog. *Welthilfssprachen* ist Esperanto, die allerdings die erhoffte Funktion nicht hat erfüllen können.

Die Untersuchung der Sprache ist mehr oder weniger ausgeprägter Bestandteil vieler wissenschaftlicher Disziplinen. In der *Philosophie* und in der *Wissenschaftstheorie* nimmt die Analyse von Sprache eine zentrale Rolle ein. Dabei ist an die bereits erwähnte Frage nach der für die Wissenschaft angemessenen Sprache zu denken sowie - durchaus damit zusammenhängend - an die erkenntnistheoretische Frage nach dem Verhältnis von Sprache und Wirklichkeit. Die *Wissenssoziologie* betrachtet u.a. die Rolle der Sprache für die Entstehung und Vermittlung gesellschaftlichen Wissens. Sie geht von der Prämisse aus, daß Sprache der Schlüssel zum Verstehen sozialer Konstruktionen ist:

> "Die allgemeinen und gemeinsamen Objektivationen der Alltagswelt behaupten sich im wesentlichen durch ihre Versprachlichung. Vor allem anderen ist die Alltagswelt Leben mit und mittels der Sprache,

1. Hier ist etwa an die vom logischen Positivismus verfolgten Ziele zu denken ([Car66]).

> die ich mit den Mitmenschen gemein habe. Das Verständnis des Phänomens Sprache ist also entscheidend[BeLu80], für das Verständnis der Wirklichkeit der Alltagswelt." ([BeLu80], S. 39)

Die Informatik untersucht vor allem formale Sprachen auf ihre Eignung "Abstraktionen von Problemen der realen Welt (zu) entwickeln, die in einem Rechner dargestellt und manipuliert werden können." ([AhUl96], S. 19)

Daneben ist Sprache der konstituierende Forschungsgegenstand der bereits erwähnten Linguistik. Sie ist darauf gerichtet, "die Sprache in ihren vielfältigen Ausprägungen, die Modalitäten ihres Erwerbs und Gebrauchs, ihre Funktionen für den Menschen unter individuellen und sozialen Gesichtspunkten" ([Mey84]), S. 612) zu erforschen. Die vielfältigen Facetten von Sprache haben zu einer beachtlichen Vielfalt linguistischer Teildisziplinen und interdisziplinärer Forschungsansätze unter Beteiligung der Linguistik geführt (umfangreichere Übersichten finden sich in [GrHa87], S. 26 f. und [Mey84]). Die *Semiotik* untersucht die Zeichen der Sprache und ihre Beziehung zu anderen Zeichensystemen. Die historische Entwicklung von Sprachen, ihre Veränderungen und ihr Verschwinden im Zeitverlauf, ist Gegenstand der *diachronischen Sprachwissenschaft*. Den Einfluß der Sprache auf menschliches Denken und Handeln untersucht die *Psycholinguistik*. Demgegenüber ist die *Neurolinguistik* auf die Zusammenhänge gerichtet, die die neurophysiologischen Grundlagen sprachlicher Kompetenz bilden. In der *Computerlinguistik* versucht man u.a. Sprache mit Hilfe formaler Modelle zu rekonstruieren. Auf diese Weise können computergestützte Simulationen durchgeführt werden und Teile sprachlicher Kompetenz - mehr oder weniger überzeugend - maschinell reproduziert werden. Die *Kommunikationstheorie* ist Gegenstand interdisziplinärer Bemühungen (Linguistik, Soziologie, Philosophie), die die Eigenschaften und Bedingungen sprachlicher Kommunikation zu erhellen versuchen.

Alle Wissenschaften, die Sprache und ihre Verwendung untersuchen, oder neue "Sprachspiele", also Sprachen und daran anknüpfende Handlungen entwerfen, sehen sich einer eigentümlichen Schwierigkeit gegenüber: Der Umstand, daß man als Forscher selbst in ein nicht vollständig auflösbares Geflecht von Sprache, Denken und Handeln eingebunden ist, markiert eine latente, kaum zu überwindende Befangenheit, die in ein scheinbares Paradoxon führt: Erkenntnis ist ohne Sprache kaum möglich, sie droht allerdings gleichzeitig durch Sprache verzerrt zu werden:

> "Sprache ist das gemeinsame Element jedes Erklärungsversuchs. Sprache jedoch kann als etwas gegebenes benutzt werden, dessen Grundlage und Funktionsweise man nicht befragt. Oder sie kann als Instrument benutzt werden, das in sich selbst analysierbar ist und ein zentrales Problem aufwirft, wenn man versucht, Kommunikation als ein biologisches Phänomen zu verstehen. ... Sprache ist mein Instrument, aber zur gleichen Zeit auch mein Problem." ([Mat87], S. 90 f.)

Es bleibt allerdings der Trost, daß allein das Wissen um diese Befangenheit eine relativierende Funktion erfüllen mag.

2.2 Implikationen für die Untersuchung von Modellierungssprachen

Die Betrachtung von Modellierungssprachen, die im Rahmen der Software-Entwicklung eingesetzt werden, erfolgt üblicherweise losgelöst von sprachwissenschaftlichen oder gar philosophischen Überlegungen. Das legt die Frage nach dem Sinn unseres kleinen Exkurses nahe. Wie bereits erwähnt, ist eine Modellierungssprache eine Sprache, die auch der sprachlichen Verständigung zwischen Menschen dient. Sie wird zudem im Rahmen einer häufig anspruchsvollen intellektuellen Tätigkeit, dem Modellieren, eingesetzt. Die vielfältigen Wechselwirkungen zwischen kognitiver Wahrnehmung und Konzeptualisierung auf der einen und Sprache auf der anderen Seite sind in diesem Zusammenhang zweifelsohne von erheblicher Bedeutung. Da eine Modellierungssprache zumeist mit einer graphischen Notation einhergeht, sind desweiteren einschlägige wahrnehmungspsychologische Zusammenhänge zu berücksichtigen.

Der Entwurf und die Bewertung von Modellierungssprachen erfordert also zahlreiche fachliche Perspektiven. Das legt die Frage nahe, welche wissenschaftlichen Disziplinen zuständig sind bzw. sein sollten. Auch wenn wir hier nicht den Versuch unternehmen, die Entstehung existierender Modellierungssprachen zu rekonstruieren, so wagen wir doch die Behauptung, daß sie in aller Regel unter dem eingeschränkten Blickwinkel des Software-Engineering entworfen wurden. Aspekte, die außerhalb dieses Blickwinkels liegen, ganz gleich wie wichtig sie für eine angemessene Behandlung des Gegenstands sind, wurden dabei weitgehend übergangen. Diese Einschränkung wiegt um so schwerer als die Mehrzahl der weithin bekannten objektorientierten Modellierungssprachen aus einem Umfeld kommen, in dem das Bemühen um eine grundsätzliche Reflexion weniger geschätzt wird als die pragmatische Ausrichtung an Ergebnissen, die sich als nützlich erweisen. Zudem kommt in unserem Kontext dem Umstand, daß Sprache Denken und Wahrnehmung prägt, eine subtile Bedeutung zu: Diejenigen, die Modellierungssprachen entwerfen, verfügen in aller Regel über einschlägige Modellierungserfahrungen - was wohl unerläßlich ist, sind aber dadurch u.U. in zweierlei Hinsicht eingeschränkt. Einerseits dürfte es ihnen schwerfallen, Sprachparadigmen zu denken, die jenseits ihrer bisherigen Erfahrungen angesiedelt sind, andererseits fehlt ihnen das unmittelbare Verständnis für die Wahrnehmungsmuster derer, die ihren spezifischen Hintergrund nicht teilen und gleichzeitig potentielle Nutzer einer Modellierungssprache sind.

Angesichts der offenkundigen Vielfalt und Tiefe der Herausforderungen, die die Beurteilung einer Modellierungssprache mit sich bringt, mag man resignieren oder aber in pragmatischer Absicht wesentliche Probleme ignorieren. In einer anwendungsorientierten Wissenschaft gibt es einen gewissen Bedarf an gemeinsamen, wenn auch ggfs. vorläufigen Konstruktionen. Dies gilt ohne Zweifel für Modellierungssprachen und ihre Anwendungen im Software-Engineering und in der Wirtschaftsinformatik. Dieser Umstand sollte u.E. jedoch nicht dazu führen, wesentliche Fragestellungen leichtfertig zu übergehen - oder sie dilletantisch abzuhandeln. Der Ausweg aus diesem Dilemma kann nur darin bestehen, offene, aber außerhalb der eigenen Kompetenz angesiedelte Probleme als solche zu kennzeichnen.

3 Objektorientierte Modellierungssprachen

Die objektorientierte Modellierung ist durch eine wenig einheitliche und zum Teil unangemessene Begrifflichkeit gekennzeichnet. Dabei ist einerseits an die Mehrdeutigkeiten zu denken, die mit der Verwendung objektorientierter Konzepte bei der Modellierung einhergehen - wie etwa die je nach Kontext unterschiedliche Bedeutung des Begriffs "Objekt", andererseits an Begriffe, die das Umfeld der Modellierung strukturieren: Methode, Methodologie, Metamodell etc. Die Festlegung von Modellierungskonzepten ist Gegenstand von Modellierungssprachen. Wir werden darauf in 3.2 eingehen. Zunächst wird eine Basisterminologie skizziert, die zentrale, zum Teil bereits erwähnte Begriffe im Kontext der objektorientierten Modellierung enthält.

3.1 Terminologische Grundlagen

Eine *Methode* ist ein systematischer Ansatz zur Unterstützung der Lösung einer Klasse von Problemen. Dabei ist gewöhnlich eine Unterteilung in Teilprobleme mit jeweils assoziierten Methoden, Techniken oder Heuristiken vorgesehen. Darüber hinaus beinhaltet eine Methode eine mehr oder weniger rigide zeitliche Ordnung der einzelnen Problemlösungsaktivitäten. Im Unterschied dazu ist eine Methodologie eine Lehre vom Einsatz von Methoden. Sie enthält Richtlinien zur Bewertung und Verwendung von Methoden. Ein konzeptuelles Modell ist die Beschreibung einer tatsächlichen oder gedachten Domäne unter Vernachlässigung implementierungsrelevanter Aspekte. Eine Beschreibung erfordert eine Sprache. Falls eine Sprache speziell zum Zweck der konzeptuellen Modellierung eingeführt wird, sprechen wir von einer *Modellierungssprache*. Sie beinhaltet zumeist, aber nicht notwendigerweise, eine graphische Notation. Eine Modellierungssprache umfaßt die Festlegung und Verknüpfung von Sprachelementen zur Beschreibung von Modellen und zusätzlich eine mehr oder weniger

rigide Interpretation dieser Sprachelemente. Wenn Syntax und Semantik einer Modellierungssprache eindeutig spezifiziert sind (was vor allem für die Semantik nicht immer der Fall ist), liegt eine formale Modellierungssprache vor, ansonsten sprechen wir von einem semi-formalen Ansatz. Syntax und Semantik werden unter Rückgriff auf eine *Metasprache* beschrieben.

Eine Metasprache kann selbst wieder als Modellierungssprache ausgelegt sein. Auf diese Weise kann eine Sprachbeschreibung als Modell dargestellt werden. Man spricht dann von einem *Metamodell*. Das Metamodell definiert hauptsächlich die abstrakte Syntax und Semantik der Modellierungssprache. Jedes Metamodell definiert eine Sprache *L(M)*, die mit der Sprache *L* übereinstimmen muß, für die dieses Metamodell entwickelt wurde (*L(M)=L*). Zwei Metamodelle sind äquivalent, wenn sie dieselbe Sprache definieren. Eine Metasprache kann außerdem verwendet werden, um eine *Grammatik* zu beschreiben, die die Regeln zum Erzeugen korrekter Sätze aus einer Menge vorgegebener Symbole festlegt. Die Symbole werden in Terminal- und Nichtterminalsymbole unterteilt. Ein Satz bzw. eine *Satzform* über einem endlichen Alphabet wird in der Informatik Wort genannt. Auf diese Weise kann eine Sprache auch aufgefaßt werden als die Menge aller Wörter von Terminalsymbolen, die durch eine zugehörige Grammatik erzeugt werden. Während dabei nur die Syntax einer Sprache berücksichtigt wird, kann daneben auch die Semantik der einzelnen Wörter und Sätze einer Sprache definiert werden. Hier ist z.B. an axiomatische oder operationale Semantiken zu denken, wie sie mitunter zur Spezifikation von Programmiersprachen verwendet werden ([Kow96], S. 372 ff.). Neben der Trennung von Syntax und Semantik kann aber noch ein Schritt weitergegangen werden, indem die zugrundeliegenden Konzepte von der konkreten Darstellung abstrahiert werden. Dies führt zu einer Trennung von abstrakten Grundbegriffen und konkreten Darstellungen. Diese dritte Variante der Definition kann gut angewendet werden, wenn verschiedene Sprachen untersucht werden, die sich auf ein gemeinsames Paradigma - also etwa die Objektorientierung - stützen. Die Konzepte des gemeinsamen Paradigmas können dann als abstraktes Grundgerüst dieser Sprachen aufgefaßt werden. Wir nennen eine Modellierungssprache objektorientiert, wenn sie wesentlich auf die software-technisch zu konkretisierenden Konzepte Objekt und Klasse gestützt ist. Da es ggfs. hilfreich ist, ein komplexes System mit unterschiedlichen Abstraktionen zu beschreiben, kann eine Modellierungssprache mehrere Modell- bzw. Diagrammarten unterstützen.

Eine *Modellierungsmethode* ist eine Methode, die die Erstellung und Pflege einer Klasse von Modellen unterstützt. Sie umfaßt i.d.R. eine oder mehrere Modellierungssprachen sowie einige der folgenden Aspekte:

- Unterteilung der Gesamtaufgabe in Teilaufgaben
- temporale/kausale Ordnung der Teilaufgaben
- Verzeichnis der für die Teilaufgaben benötigten und von denselben zu erzeugenden Dokumente
- Heuristiken zur Erfassung der benötigten Informationen
- Kriterien oder Metriken zur Evaluierung von Modellen
- Zusätzliche Unterstützung des Projektmanagement
- Anwendungsbeispiele

Diese Teilaspekte beschreiben eine Vorgehensweise für den Software-Entwicklungsprozeß. Eine Modellierungsmethode umfaßt damit eine oder mehrere Modellierungssprachen und eine Vorgehensweise. Henderson-Sellers, aber auch andere Software-Entwickler argumentieren, daß für die Software-Entwicklung ein wohlbestimmter Prozeß wichtig ist und daß eine Integration von Prozeß, Modellierungssprache und Notation sich günstig auswirkt. Die Bedeutung eines Entwicklungsprozesses für die Software-Entwicklung ist unbestritten, ob aber ein Prozeß im Gegensatz zu einer Beschreibungssprache standardisiert werden sollte, ist fragwürdig. Derzeit gibt es keinen solchen Standard für den Software-Entwicklungsprozeß, allerdings verschiedene, mehr oder weniger verbreitete Referenzmodelle für solche Prozesse - wie z.B. das V-Modell ([BrDr93]). Die Verwendung einer einheitlichen

Notation mit verschiedenen Ausprägungen von Entwicklungsprozessen erscheint deshalb als die vielversprechendste Lösung. Nicht die verschiedenen Vorgehensweisen stellen häufig ein Problem dar, sondern die unterschiedlichen Dokumentarten, die bei den einzelnen Aktivitäten entstehen. Eine Vereinheitlichung dieser Dokumentarten und Notationen ermöglicht ein einheitliches Vokabular und damit eine bessere Verständigung und einen besseren Vergleich verschiedener Vorgehensweisen. Die Bedeutung des Software-Prozesses wird durch diese Festlegung nicht gemindert. Ein definierter Software-Entwicklungsprozeß ist eine notwendige, aber nicht hinreichende Bedingung für die erfolgreiche Software-Entwicklung.

Eine *Modellierungsmethodologie* ist eine Lehre von Modellierungsmethoden. Ihr Gegenstand ist die Bewertung und Anwendung von Modellierungsmethoden. Wie bereits angedeutet, werden wir uns im wesentlichen auf die Betrachtung objektorientierter Modellierungssprachen beschränken, nicht zuletzt deshalb, weil generelle Gütekriterien und damit: eine Standardisierung von Methoden oder gar Methodologien kaum vorstellbar sind (vgl. dazu [Fra97]).

3.2 Idealtypischer Aufbau objektorientierter Modellierungssprachen

Eine objektorientierte Modellierungssprache stellt eine Menge von Sprachkonstrukten und Verknüpfungen sowie deren Interpretationen zur Verfügung. Dabei ist es wesentlich, daß diese Sprachkonstrukte objektorientierte Konzepte enthalten. Dazu gehören neben der bereits erwähnten Verfügbarkeit der Modellierungskonzepte *Klasse* und *Objekt* Konstrukte zur Darstellung von Verkapselung, Generalisierung und Polymorphie. Diese Sprachkonzepte selbst bilden das abstrakte Fundament der Sprache und werden durch konkrete Sprachkonstrukte ausgedrückt. Die Sprachkonstrukte lassen sich i.d.R in textuelle und graphische unterteilen.

UML, OML und viele andere objektorientierte Modellierungssprachen weisen denselben prinzipiellen Aufbau auf und lassen sich durch zwei Teilkomponenten beschreiben (Abb. 1). Die abstrakte Syntax und Semantik definieren die Konzepte hinter den Sprachelementen und deren syntaktische Anordnung. Zudem stellen sie eine Interpretation, also eine Semantik bereit und bilden die - mehr oder weniger - formale Grundlage für die Modellierungssprache. Die abstrakte Syntax und Semantik beschreiben die Intention der Sprache und legen im wesentlichen ihre Ausdrucksmächtigkeit fest. Sie bilden die eigentliche Sprachbeschreibung. Viele Aspekte bei der Untersuchung und Bewertung von Modellierungssprachen beziehen sich deshalb auf die abstrakte Syntax und Semantik.

Die konkrete Syntax legt textuelle und graphische Sprachelemente auf der Basis der abstrakten Konzepte fest. Bei objektorientierten Modellierungssprachen dominieren graphische Darstellungsformen für die konkrete Syntax. Die Notation legt die graphische Repräsentation der Sprachkonzepte fest, wobei Verknüpfungen der einzelnen Notationselemente nur auf der Basis der abstrakten Syntax und Semantik zulässig sind. Im allgemeinen wird die Notation als Sicht oder Repräsentation aufgefaßt. Ihre Hauptaufgabe ist dann das Veranschaulichen. Sie reichert das abstrakte Modell dabei mit zusätzlichen Layout- und Darstellungsinformationen an. Zwischen den Sprachkonzepten und den graphischen Beschreibungsmitteln besteht meist eine direkte Zuordnung. Ein Modell wird somit im wesentlichen durch die abstrakten Sprachkonzepte beschrieben, während die Notation zur Veranschaulichung von Teilaspekten dient.

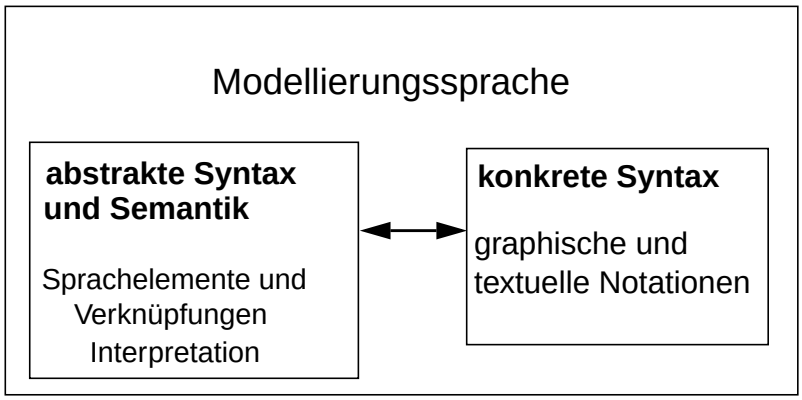

Abb. 1: Idealtypischer Aufbau einer objektorientierten Modellierungssprache

Jedes Sprachelement umfaßt im allgemeinen abstrakte und konkrete Aspekte. Die abstrakten Aspekte beinhalten das Wesen oder die logische Bedeutung des Sprachelementes unabhängig von der Darstellung. Die konkreten Aspekte beziehen sich auf eine konkrete Darstellung und beinhalten zusätzliche Layout- oder Anordnungsinformationen. Um zu betonen, daß sich eine Aussage speziell auf den abstrakten (konkreten) Aspekt eines Sprachelementes bezieht, wird von einem abstrakten (konkreten) Sprachelement gesprochen[1]. Der Vorteil der Trennung von abstrakter Syntax und Semantik (Sprachbeschreibung) und konkreter Syntax (Repräsentation) ist die Möglichkeit einer besseren formalen Darstellung der Sprache, da auf diese Weise von der konkreten Syntax und Repräsentation abstrahiert werden kann. Andererseits kann aber auch insbesondere die graphische Notation von ihrem Verwendungszweck entkoppelt und separat entwickelt werden, wobei zum Beispiel verstärkt semiotische Erfahrungen berücksichtigt werden können. Bei dieser Betrachtungsweise ist die Notation relativ leicht austausch- und anpaßbar. Defizite der Notation können daher relativ leicht behoben werden. Allerdings beeinflußt die Auswahl einer Notation auch die Sprachkonzepte, so daß beide Aspekte nicht vollständig entkoppelt sein dürfen. Die Sprachkonzepte wie auch die Notationselemente können meistens in aufgabenspezifische Teilgruppen klassifiziert werden.

3.3 Umfang objektorientierter Modellierungssprachen

Objektorientierte Modellierungssprachen dienen zur Entwicklung von Software-Systemen mit Hilfe des objektorientierten Programmierparadigmas. Sie sollen alle Phasen der Software-Entwicklung unterstützen und alle wichtigen Aspekte der Software beschreiben können. Die Unterstützung einer Modellierungssprache bezieht sich dabei im wesentlichen auf die Modellierungstätigkeiten während der Software-Entwicklung. Aufgaben, die nicht direkt mit Modellierung verbunden werden, wie Projektmanagement, aber auch Testen, werden daher nicht oder nur wenig von Modellierungssprachen berücksichtigt. Die Beschreibung der Software umfaßt die Darstellung der statischen Struktur und des dynamischen Verhaltens. Die statische Struktur umfaßt die Systemstruktur bis zur Objektstruktur der einzelnen Komponenten. Zum dynamischen Verhalten gehören die Systemfunktionalität, die Funktionalität einzelner Klassen aber auch das Zustandsverhalten von Komponenten. Zusätzlich sollten Analyse-, Entwurfs- und Implementierungstätigkeiten unterstützt werden. Die Art der verwendeten Diagramme spiegelt direkt den anvisierten Umfang der Modellierungssprache wider. Da die Informationen im Diagramm immer auch im Metamodell enthalten sein müssen, reicht es aus, sich auf die abstrakten Sprachelemente zu konzentrieren. Im wesentlichen wird die Mächtigkeit einer Modellierungssprache von ihrem Verwendungszweck bestimmt. Ihr Sprachumfang definiert, auf welche Weise Modelle beschrieben werden können. Gleichzeitig wird mit der Festlegung des Sprachumfangs auch festgelegt,

1. Zur Betonung des abstrakten Charakters eines Sprachelementes kann auch der Begriff Sprachkonzept verwendet werden.

welche Aspekte in der Sprache ausdrückbar sind und welche nicht. Die verwendete Sprache mit ihren Konzepten hat direkten Einfluß auf die Art der Modellierung.

Da im folgenden generelle objektorientierte Modellierungssprachen untersucht werden, müssen alle oben genannten Bereiche unterstützt werden. Eine Modellierungssprache muß für jeden der genannten Bereiche geeignete Beschreibungen bereitstellen.

Tabelle 1: Umfang von Modellierungssprachen

Tätigkeiten	statisch strukturelle Sicht	dynamisch funktionale Sicht
Anforderungen	Systemkomponenten	Systemanforderungen, Funktionalität
Analyse	Systemkomponenten	Anforderungen, Zustandsverhalten
Systementwurf	Systemarchitektur (verteilt, nebenläufig)	Systemfunktionalität, Objektflüsse, nebenläufige Prozesse, Zustandsverhalten
Klassenentwurf	Klassenhierarchie,	Funktionalität der Klassen, Objektflüsse, nebenläufige Prozesse, Zustandsverhalten, Spezifikationen
Implementierung	Spezifikationen, Code, Verteilungsmodell, Initialisierungen	Operationen
Testen, Verifizieren	-	Testfälle, Bedingungen

Tabelle 1 zeigt die Bereiche, die durch die objektorientierte Modellierung abzudecken sind. Dabei erfolgt eine Unterteilung in Tätigkeiten des Software-Entwicklungsprozesses und den Sichtweisen auf ein Modell. Einige Merkmale wie "Nebenläufigkeit" treten sowohl in der statischen als auch der dynamischen Sicht auf, da sie direkte Auswirkungen auf die Struktur und das Verhalten eines Systems haben. Die Sprachmittel sollten alle diese Bereiche unterstützen. Eine Beschreibungsform kann dabei durchaus mehrere Bereiche abdecken. Die Auswahl der Sprachmittel ist dabei zumeist willkürlich. Sie entwickelt sich meist im Laufe der Zeit und favorisiert solche Sprachmittel, die sich in der Praxis bewähren oder eine gute theoretische Fundierung besitzen. Die typischen Teilmodelle aktueller Modellierungssprachen zur Unterstützung dieser Bereiche sind:

- ein Use-Case Modell für die Anforderungs- und Systemanalyse
- ein System/Teilsystemmodell zur Beschreibung der Systemarchitektur
- ein Klassen/Objektmodell zum Beschreiben von Klassen, Objekten und dem Fluß von Nachrichten
- ein Zustandsmodell zum Beschreiben des Zustandsverhaltens von Objekten und Systemen
- ein Implementierungsmodell zum Angeben der Semantik von Operationen
- ein physikalisches Realisierungsmodell

Zusätzlich werden Techniken benötigt, um die Entwicklung eines Modells von der Analyse bis zur Implementierung verfolgen zu können. Diese können in der Sprache festgelegt werden oder in den Aufgabenbereich von Werkzeugherstellern fallen. Eine mögliche Variante ist die Verknüpfung der einzelnen Variationen eines Aspekts durch Querverweise oder Versionsmanagement.

Ein anderer und vielleicht besserer Ansatz ist die Festlegung einer auf die objektorientierten Konzepte beschränkten Sprache, die Schnittstellen zu anderen Beschreibungsformen hat und mit diesen integriert wird. Dies würde auch die Integration des objektorientierten Paradigmas mit anderen Paradigmen erleichtern.

4 Bewertung objektorientierter Modellierungssprachen

Objektorientierte Modellierungssprachen sind kein Selbstzweck. Vielmehr dienen sie der Unterstützung der konzeptuellen Modellierung. Modellierung ist eine komplexe Aufgabe mit vielfältigen Herausforderungen. Um mögliche Anforderungen an Modellierungssprachen zu sammeln, betrachten wir im folgenden verschiedene Blickwinkel auf die konzeptuelle Modellierung und mit ihnen einhergehende Anforderungen. Dabei ist zu berücksichtigen, daß die einzelnen Anforderungen sich überschneiden oder auch in Konkurrenz stehen können.

4.1 Strukturierung der Anforderungen

Es existiert bereits eine Vielzahl von Vergleichen zu objektorientierten Notationen und Vorgehensweisen[1]. Solche Vergleiche konzentrieren sich häufig auf eine Auflistung objektorientierter Sprachmittel. Dabei schneiden die Sprachen am günstigsten ab, die sehr umfangreich sind. Andere Merkmale wie Integration, eindeutige Definition der Sprachmittel oder die Angemessenheit für den Verwendungszweck werden dagegen nicht berücksichtigt. In dieser Untersuchung sollen auch diese Kriterien betrachtet werden. Für qualitative Merkmale ist es jedoch schwieriger, geeignete Bewertungsmaße und korrespondierende Interpretationen festzulegen. Auch gibt es häufig verschiedene Ansichten über die Wichtigkeit einzelner Merkmale (z. B. Formalisierungsgrad, Mehrfachvererbung). In vielen Fällen erfolgt daher nur eine subjektiv gefärbte Einschätzung, die eine kritische Interpretation empfiehlt.

Um eine Modellierungssprache bewerten zu können, müssen ihre zielorientierten, aber auch sprachlichen Anforderungen festgelegt werden. Die Bewertung objektorientierter Modellierungssprachen kann unter vier verschiedenen Blickwinkeln vorgenommen werden. Dies sind:

- Beschreibung
- Sprachspezifikation
- Modellierungszweck
- Generelle Kriterien

Alle diese Blickwinkel sind für die Bewertung wichtig. Die Beschreibung repräsentiert die Sprache für den Benutzer. Die Güte dieser Repräsentation ist ein wichtiges Kriterium für die Erlernbarkeit und Verständlichkeit der Sprache. Die Sprachcharakteristik umfaßt die Syntax und Semantik einer Modellierungssprache. Hier werden insbesondere die Festlegungen der Sprachdarstellung, die Exaktheit der Definition und die Integration der Sprachmittel untersucht. Ob die gewählten Sprachmittel zweckmäßig sind, wird dabei zunächst nicht betrachtet. Umfaßt die Modellierungssprache auch eine graphische Notation, so ist auch diese enthalten. Der Anwendungszweck schließlich bestimmt im wesentlichen den Umfang der Sprache. Hier erfolgt eine Bewertung der Zweckmäßigkeit der Sprachmittel. Die generellen Kriterien umfassen eine Reihe von Gütemerkmalen, die für die Bewertung von Modellierungssprachen von Vorteil sind.

1. Vgl. [ArBo91], [Big97], [CrRo92], [DeFa92], [Gra91], [FiKe91], [Fra93], [Hew91], [HoGo93], [Hsi92], [MoPu92], [Obj97], [Sch94], [Ste93] und weitere.

Die Aufteilung in diese vier Teilbereiche erscheint auf den ersten Blick willkürlich. Sie erfolgte, um die Vielzahl verschiedener Anforderungen stärker strukturieren zu können. Einzelne Anforderungen können mehreren Bereichen zugeordnet werden. Zwischen den Merkmalen der einzelnen Dimensionen existieren erwartungsgemäß Wechselwirkungen beziehungsweise Antagonismen. So besteht zum Beispiel ein Konflikt zwischen der Forderung nach Ausdrucksmächtigkeit und Einfachheit. Ausdrucksmächtigkeit verlangt spezielle Sprachkonzepte, während die Forderung nach Einfachheit möglichst wenig spezielle Sprachkonzepte impliziert.

Einen anderen Kriterienkatalog bietet die OML-Beschreibung ([FiHe96], S. 8). Die wichtigsten Ziele sind:

- Eine ähnlich exakte Beschreibung wie bei einer objektorientierten Programmiersprache. Dies erfordert insbesondere die detaillierte und vollständige Festlegung von Syntax und Semantik der Sprache.
- Die konsequente Unterstützung des objektorientierten Paradigmas.
- Die Anwendbarkeit während des ganzen Entwicklungszyklus. Dies erfordert insbesondere einen nahtlosen Übergang zwischen den einzelnen Phasen und die Unterstützung eines iterativen und schrittweisen Vorgehens.
- Die leichte Erlernbarkeit und Benutzbarkeit
- Einfachheit und Skalierbarkeit
- Die Verbesserung und Unterstützung der Kommunikation zwischen Entwicklern. Dies hat Vorrang vor der Unterstützung von Entwicklungswerkzeugen.

Das interessanteste Kriterium ist sicherlich der Vergleich mit Programmiersprachen. Die Ziele aus diesem Katalog bieten eine gute Überprüfungsmöglichkeit, inwieweit OML und UML diese Kriterien erfüllen. Allerdings fehlen unter diesen Kriterien Aussagen über Konsistenz, formale Darstellung und Eindeutigkeit. Auch werden keine Metriken zur Überprüfung und Bewertung der Kriterien angegeben.

4.2 Beschreibung von Modellierungssprachen

Unabhängig von den spezifischen Qualitäten einer Modellierungssprache ist die Qualität der Sprachbeschreibung aus der Sicht der Verwender ein wichtiges Merkmal zur Beurteilung der Sprache. Sie ist wesentlich für das Erlernen und Anwenden der Sprache sowie ggfs. für ihre Abbildung auf Werkzeuge.

4.2.1 Dokumentation

Die Dokumentation einer Modellierungssprache sollte sowohl informale als auch formale Teile enthalten. Die informale, verständliche Beschreibung erleichtert das Erfassen der Grundideen. Die exakte formale Darstellung sichert die Eindeutigkeit und Unterstützung durch Werkzeuge. Die (formale) Sprachbeschreibung sollte alle wesentlichen Sprachmerkmale eindeutig und exakt beschreiben. Die Sprachbeschreibung kann in diesem Zusammenhang als abstraktes Anforderungsdokument der Sprache verstanden werden, indem die abstrakten Konzepte unabhängig von konkreten Realisierungen dargestellt werden. Die anwenderorientierte Beschreibung sollte Prinzipien, Heuristiken und Beispiele umfassen. Sie gehört nicht direkt zur Definition der Modellierungssprache. Sie hat jedoch eine große Bedeutung, da Modellierungssprachen als Instrument zur Erfüllung bestimmter Aufgaben eingeführt werden. Die Dokumentation sollte also auch darstellen, wie die Sprache im Hinblick auf die intendierten Aufgaben sinnvoll einzusetzen ist.

4.2.2 Grammatiken

Grammatiken bieten günstige Voraussetzungen zur Syntaxüberprüfung. Dieser Beschreibungsansatz ermöglicht zudem die Nutzung von Erkenntnissen aus der Theorie der formalen Sprachen und dem Compilerbau. Eine explizit angegebene Grammatik ist also grundsätzlich positiv einzuschätzen.

4.2.3 Metamodelle

Metamodelle bieten i.d.R. eine weniger gute Grundlage zur Überprüfung der syntaktischen Korrektheit von Modellen als Grammatiken. Gleichzeitig kann aber davon ausgegangen werden, daß sie für viele Betrachter eine anschaulichere Definition einer Sprache darstellen. Zudem stellen Metamodelle eine gute Grundlage für die Implementierung von Werkzeugen dar. Ein Metamodell sollte eine vollständige Sprachbeschreibung bieten. Dazu kann eine Sprache zur Spezifikation von ergänzenden Integritätsbedingungen erforderlich sein.

4.2.4 Konsequenzen

Die Beschreibung einer Sprache sollte also eine Sprachbeschreibung, eine Sprachdefinition (Grammatik oder Metamodell) und eine anwenderorientierte Beschreibung umfassen und ist ein wichtiges Instrument zum Verständnis einer Sprache. Die Beschreibung ist in vielen Fällen die einzig zulässige Referenz bei Anwendungsproblemen der Sprache und spiegelt wichtige Spracheigenschaften direkt wider. Insbesondere sollte die Beschreibungssprache - Metasprache - einer Sprache eindeutig sein, da ansonsten eine eindeutige Interpretation der Sprache kaum gewährleistet werden kann. Dies kann durch eine Formalisierung der Metasprache oder durch Beseitigung von Mehrdeutigkeiten erreicht werden. Für Sprachen, die selbst als Metasprachen verwendet werden, bedingt dies ein hohes Maß an Exaktheit und Eindeutigkeit.

4.3 Sprachspezifikation

Die Spezifikation einer Modellierungssprache umfaßt die Darstellung der Grundkonzeption sowie die Definition von Syntax und Semantik.

4.3.1 Grundkonzeption der Sprache

Eine Modellierungssprache kann monolithisch sein oder aus verschiedenen Teilsprachen bestehen. Die Teilsprachen präsentieren sich dem Verwender der Modellierungssprache zumeist in Form spezieller graphischer Notationen. Die Modellierungssprache kann dann als Sprachsystem über diese Teilsprachen aufgefaßt werden, wobei Beziehungen zwischen den Teilsprachen die Integration sichern. Es ist jedoch nicht einfach, eine Modellierungssprache als monolithisch oder aus Teilmodellen bestehend einzuschätzen, da die Sprachmittel umfangreicher Sprachen meistens logisch gruppiert sind. Die Festlegung erfolgt dann meist aus subjektiven oder historischen Gründen.

Fast alle Modellierungssprachen für die Objektorientierung sind aus Teilsprachen aufgebaut. Dies ist zum einen durch die Übernahme bekannter Beschreibungsansätze zum anderen durch die Zerlegung des Entwicklungsprozesses in Teilaufgaben begründet.

Konsequenzen
Der Aufbau der Sprache kann als untergeordnetes Kriterium betrachtet werden. Monolithische Ansätze versprechen eine bessere Integration und Konsistenzsicherung, sind aber auch schwerer zu verstehen, anzuwenden und zu erweitern. Weiterhin erscheint eine Modularisierung auch bei Beschreibungssprachen hilfreich. Aus Teilsprachen zusammengesetzte Beschreibungssprachen unterstützen die Modularisierungsidee und sind gegebenenfalls leichter erweiterbar. In diesem Fall ist aber die Integration und Konsistenzsicherung schwieriger. Beschreibungsansätze, die auf Metamodellen beruhen, benutzen häufig den Teilsprachenansatz, da die Metamodelle in Teilmodelle untergliedert sind.

4.3.2 Abstrakte Syntax und Semantik

Welche Sprachmittel soll eine objektorientierte Modellierungssprache umfassen? Zu dieser Frage gibt es zwei grundlegende Antworten. Erstens kann man die Festlegung eines kleinen Kerns ausgewählter Sprachmittel fordern. Oder man fordert eine möglichst weite Einbeziehung derzeit bekannter Sprachmittel. Eine andere Frage ist, wie mit alternativen Sprachmitteln umgegangen werden soll. Wird nur

die am weitesten verbreitete Variante beachtet, oder werden auch die anderen Varianten berücksichtigt? Bei der Festlegung der Konzepte sind folgende Entscheidungskriterien wichtig:

- Prinzip der Sparsamkeit von Konzepten und Syntax
- gewünschter Anwendungszweck der Sprache
- gewünschte Mächtigkeit der Sprache

Zwei andere Bewertungskriterien sind der Grad der Formalisierung und die Orthogonalität einzelner Gruppen von Sprachkonzepten. Eine möglichst formale Darstellung der abstrakten Sprachelemente ist eine gute Voraussetzung, um die Bedeutung der Sprachmittel eindeutig festzulegen. Der Formalisierungsgrad stellt ein wichtiges Mittel zur Bewertung der Eindeutigkeit einer Modellierungssprache dar. Orthogonalität erlaubt die gleichartige Verwendung eines Sprachelementes unabhängig vom konkreten Anwendungsumfeld, da die Sprachmittel sich gegenseitig nicht oder nur wenig beeinflussen.

Konsequenzen für Modellierungssprachen
Die in einer Sprache definierten Konzepte stellen ein wichtiges Bewertungskriterium dar. Allerdings ist der Umfang einer Sprache nicht automatisch ein positiver Indikator. Vielmehr ist die eindeutige und exakte Definition der Sprachmittel von Bedeutung.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien
Für die verschiedenen Paradigmen und Konzepte kann überprüft werden, ob sie durch geeignete Konzepte unterstützt werden. Die einzelnen Definitionen können hinsichtlich ihrer Eindeutigkeit bewertet werden. Der Formalisierungsgrad kann hierfür ein Indikator sein. Die Anschaulichkeit und Nützlichkeit der bereitgestellten Konzepte kann dagegen nur eingeschränkt bewertet werden.

4.3.2.1 Objektorientierte Konzepte

Für objektorientierte Modellierungssprachen nehmen die objektorientierten Sprachkonzepte eine herausragende Bedeutung ein. Sie bilden das Grundgerüst der objektorientierten Modellierung. Die Unterstützung objektorientierter Konzepte durch eine Modellierungssprache ist jedoch angesichts der mangelnden Einheitlichkeit dieser Konzepte schwer zu beurteilen. Es gibt eine große Fülle möglicher Anforderungen, die im Zusammenhang mit dieser Frage zu berücksichtigen sind.

Durch die vielen verschiedenen Definitionen für objektorientierte Konzepte ist es äußerst schwierig, eine exakte und umfassende Definition auszuwählen. Eine eindeutige Festlegung der abstrakten Syntax und Semantik der objektorientierten Konzepte ist jedoch eine unabdingbare Voraussetzung, um Modelle auf der Basis der abstrakten Syntax eindeutig interpretieren zu können. Eine weitere Frage in diesem Zusammenhang ist, ob sich Modellierungssprachen bei der Definition der objektorientierten Konzepte an heutige objektorientierte Programmiersprachen anlehnen sollen. Zusätzlich wird eine einheitliche Begriffsbildung der objektorientierten Terminologie durch ihre unterschiedliche Verwendung in der Analyse, dem Entwurf, der Implementierung oder im Datenbankenbereich erschwert. Firesmith hat in [Fir95] verschiedene Definitionen für objektorientierte Begriffe zusammengestellt. Im Prinzip wird für die objektorientierte Analyse von Systemen, die Ist-Analyse und die objektorientierte Software-Entwicklung die gleiche Terminologie verwendet. Die Begriffe werden in den einzelnen Anwendungsbereichen jedoch mit unterschiedlichen Interpretationen verwendet. Tabelle 2 zeigt verschiedene Definitionen für den Begriff Objekt.

Tabelle 2: Definitionen des Begriffs Objekt

Bereich	Definition
Objektorientierte Analyse	Ein Objekt ist ein Konzept oder Gegenstand aus der Realität, der für den betrachteten Anwendungsbereich von Bedeutung ist.

Tabelle 2: Definitionen des Begriffs Objekt (Fortsetzung)

Bereich	Definition
Objektorientierter Entwurf	Ein Objekt ist ein Exemplar einer Klasse. Die Klasse definiert das Verhalten und den Zustand des Objektes.
Objektorientierte Programmierung	Objekt = Daten+Operationen+Identität. Ein Objekt ist ein Exemplar einer Klasse.

Meistens werden alle diese Teilaspekte in einer Definition zusammengefaßt. Mit dem Übergang von der Analyse zur Implementierung erfolgt eine Konkretisierung des Sprachelementes, die gleichzeitig mit einem Verlust an problemspezifischer Bedeutung verbunden ist. In der Analyse dominieren Begriffsbildungen, die es erleichtern, Ausschnitte aus der Realität mit den objektorientierten Begriffen zu identifizieren. Beim Programmieren ist die Festlegung der objektorientierten Programmiersprache maßgebend. Die unterschiedliche Interpretation der Beschreibungsmittel bei den einzelnen Tätigkeiten ist jedoch wichtig. Sie spiegelt deren Abstraktions- und Konkretisierungsgrad wider.

Weiterhin muß geklärt werden, welche objektorientierten Konzepte unterstützt werden sollen. Es ist offensichtlich, daß dies zum einen die Standardkonzepte wie Klasse (Typ) und Objekt, Ableitungsregeln wie Vererbung, Delegation und Untertypbildung sowie Polymorphie sind. Darüberhinaus gibt es jedoch eine Vielzahl von Spezialkonzepten aus verschiedenen Ansätzen. Dies sind zum Beispiel verschiedene Beziehungsarten zwischen Objekten und Klassen, Interfaces, Festlegungen für Kardinalitäten oder die Rollenmodellierung[1].

Konsequenzen für Modellierungssprachen

Eine objektorientierte Modellierungssprache kann nicht standardisiert werden, wenn nicht gleichzeitig auch die objektorientierten Konzepte standardisiert werden. Allgemein kann man von einer objektorientierten Modellierungssprache fordern, daß sie objektorientierte Konzepte wie Klasse, Objekt, Vererbung etc. in einer Weise verwendet, die weitgehend den gängigen - wenn auch nicht einheitlichen - Begriffsverwendungen entsprechen. Dazu gehören neben der Beschreibung der Konzepte ggfs. auch Anwendungsbeispiele. Im Hinblick auf die Implementierung ist es wünschenswert, daß eine Abbildung der Konzepte der Modellierungssprache auf korrespondierende Konzepte von Implementierungssprachen unterstützt wird. Daneben gibt es eine Vielzahl von Sprachmitteln, die die objektorientierte Modellierung erleichtern mögen, deren Beurteilung allerdings zum Teil ambivalent ausfällt. Mitunter ist zwischen den Diagrammen zur Erstellung von Objektmodellen und denjenigen, die der Abbildung dynamischer oder funktionaler Aspekte dienen, ein Bruch festzustellen. Eine Modellierungssprache sollte eine möglichst enge Integration von Teilmodellen ermöglichen. Die Integration der Teilmodelle nimmt tendenziell zu, wenn die Modellelemente, die in verschiedenen Teilmodellen verwendet werden, in allen Modellen in einheitlicher Weise bezeichnet und konzeptualisiert werden (also z.B. eine Operation überall als Eigenschaft einer Klasse bzw. eines Objekts angesehen wird).

Eine objektorientierte Modellierungssprache muß weiterhin Mechanismen definieren, um die Sprachmittel einer Phase auf die Sprachmittel einer anderen Phase abzubilden. Auch in diesem Fall sollte die Definition Teildefinitionen für die verschiedenen Phasen enthalten sowie Hinweise für den Bedeutungswechsel von einer Phase in die andere. Dabei dürfen aber keine semantischen Widersprüche entstehen. Problematisch ist z.B. die Abbildung des Analysekonstrukts “dynamische Klassifikation” [MaOd92] auf die Vererbungsmechanismen von objektorientierten Programmiersprachen wie Smalltalk oder Java, da diese Vererbung als statische Klassenbeziehung definieren und die generelle dynamische Änderung der Klasse eines Objektes nicht erlauben.

1. Vgl. [Ree95] und [Tas97]

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Hier ist in erster Linie an eine Liste objektorientierter Modellierungskonzepte zu denken, für deren Elemente jeweils anzugeben ist, ob sie vorhanden sind, ob ihre Semantik eindeutig spezifiziert ist und wie sich diese Semantik zu bekannten Interpretationen (auch auf der Ebene von Implementierungssprachen) verhält. Um einige Beispiele zu nennen:

- Wie werden Attribute spezifiziert?
- Welche Arten von Beziehungen zwischen Objekten gibt es? Ist ihre Semantik eindeutig festgelegt?
- Wie können Kardinalitäten von assoziierten Objekten dargestellt werden?
- Kann neben Einfachvererbung auch Mehrfachvererbung ausgedrückt werden, welche spezifische Bedeutung ist jeweils vorgegeben?
- Welche Eigenschaften von Klassen können in Unterklassen überschrieben bzw. modifiziert werden?
- Können für Operationen Vor- und Nachbedingungen angegeben werden - wenn ja, in welcher Form?
- Können Trigger bzw. Geschäftsregeln ("Business Rules") dargestellt werden?

4.3.2.2 Modularisierungskonzepte

Die Modularisierungskonzepte fördern die Abstraktion und Zerlegung in Teilaufgaben und spielen bei der Entwicklung großer Systeme eine wichtige Rolle. Sie helfen bei der Strukturierung der Systeme. Die Unterstützung von Teilsystemen, Paketen ("Packages") oder Modulen ist eine wichtige Eigenschaft für objektorientierte Modellierungssprachen. Sie erweitern die Kapselungsmöglichkeiten der objektorientierten Konzepte und sind zusätzlich wichtige Modellierungskonzepte beim Entwurf der Grobstruktur. Auch Modularisierungskonzepte können mit anderen Sprachkonzepten in Verbindung gebracht werden. Diese Beziehungen können Enthaltensein-Beziehungen, Ableitungsbeziehungen, Sichtbarkeitsbeziehungen und andere umfassen. Die Modularisierungskonzepte können relativ unabhängig von anderen Sprachmitteln untersucht werden.

Konsequenzen für Modellierungssprachen

Objektorientierte Modellierungssprachen müssen Modularisierungskonzepte für die verschiedenen Aspekte und Abstraktionsebenen unterstützen. Dies umfaßt Konzepte für Teilsysteme, Komponenten, Gruppen von Klassen bis hin zu Modulen für die Organisation von Programmcode. Da auch die einzelnen Modularisierungskonzepte untereinander nicht isoliert sind, müssen zusätzlich Beziehungen zwischen diesen berücksichtigt werden.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Die Beurteilung von Modularisierungskonzepten beschränkt sich auf die Feststellung der jeweiligen Inhalte von Modulen bzw. Komposita. Die Verwaltung von Modellen mit Hilfe von CASE-Werkzeugen wird zunächst dadurch unterstützt, daß die Beschreibung der Metamodelle in einer Form erfolgt, die prinzipiell auch für die Modellierung von CASE-Werkzeugen geeignet ist. Eine noch bessere Unterstützung ergibt sich durch die ergänzende Bereitstellung entsprechend erweiterter Modelle, in denen Sachverhalte Berücksichtigung finden, die für Werkzeuge von Bedeutung sind: Versions- und Benutzerverwaltung, Hilfesysteme etc.

4.3.2.3 Dynamische Konzepte

Objektorientierte Konzepte wie Klasse, Objekte, Vererbung, Klassenbeziehungen und Modularisierungskonzepte betonen hauptsächlich die statische Struktur eines Systems. Neben diesen Konzepten werden daher noch Konzepte benötigt, die das dynamische Verhalten eines Systems beschreiben. Dazu dienen zum einen die objektorientierten Konzepte Objekt und Botschaft, die das Interaktionsverhalten angelehnt an Programmiersprachen operational beschreiben. Zusätzlich können aber auch zustandsori-

entierte oder deklarative Beschreibungen genutzt werden. Für die Analyse ist es zudem vorteilhaft, Beschreibungen für funktionale Anforderungen und konkrete Anwendungsfälle bereitzustellen. So haben sich zum Beispiel Use-Cases[1] und Zustandsautomaten als wichtige Modellierungsinstrumente in der objektorientierten Software-Entwicklung etabliert. Eine weitere Beschreibungsform sind OBA[2]-Sätze, die viele Ähnlichkeiten zu Use-Cases besitzen.

Konsequenzen für Modellierungssprachen

Eine objektorientierte Modellierungssprache muß auf alle Fälle die Interaktionen und den Botschaftenfluß zwischen Objekten darstellen können, da dies die grundlegende Realisierung der Funktionalität im objektorientierten Paradigma ist. Zusätzlich sind strenge Integrationsregeln für die Verknüpfung von statischer und dynamischer Sicht auf ein Modell notwendig, da die statische Sicht festlegt, welches dynamische Verhalten realisierbar ist. Die Unterstützung weiterer dynamischer Beschreibungsmittel verbessert die Detaildarstellung des Anwendermodells. Sie führen in vielen Fällen aber zu nichtobjektorientierten Sichtweisen. Eine Integration mit dem objektorientierten Paradigma ist daher erforderlich.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Hier ist einerseits zu prüfen, welche Diagrammarten zur Darstellung dynamischer Konzepte verfügbar sind. Andererseits ist zu klären, wie präzise ihre Semantik festgelegt ist und welche Integritätsbedingungen für die Integration mit anderen Teilmodellen sich daraus ergeben. Weiterhin ist darauf zu achten, ob bei Vorliegen verschiedener Diagrammarten für die dynamische Modellierung vermeidbare Redundanzen entstehen.

4.3.2.4 Prozeßorientierte und nebenläufige Konzepte

Nebenläufigkeit und Verteilung spielen bei umfangreichen Systemen eine immer größere Rolle. Verschiedene objektorientierte Sprachen unterstützen aktive Objekte und asynchrone Botschaften, zumal nachrichtengekoppelte parallele Systeme viele Gemeinsamkeiten zum objektorientierten Paradigma aufweisen. Es gibt daher Forschungen, wie Prozesse und Objekte miteinander verschmolzen werden können. Aber auch die Verteilung objektorientierter Systeme in Netzwerken gewinnt immer größere Bedeutung. Prozeßorientierte Beschreibungen von Systemen sind ein anderes Paradigma, das in vielen Bereichen eine große Bedeutung hat (Business Process Engineering, Workflows) und häufig zusammen mit objektorientierten Techniken eingesetzt wird. Objekt-Prozeß-Modelle[3] versuchen ebenfalls, Prozeßmodellierung und objektorientierte Konzepte miteinander zu verbinden. In diesem Zusammenhang sind auch *Use Cases* zu erwähnen, die verschiedentlich für die Modellierung sogenannter Geschäftsprozesse eingesetzt werden. Im Hinblick auf die Entwicklung verteilter und paralleler Software-Systeme ist jedoch die Frage aufzuwerfen, ob es nicht angemessenere Konzepte gibt, die unterstützt werden sollten. Dies könnten zum Beispiel Petrinetze oder Prozeßkalküle sein.

Konsequenzen

Längerfristig wird eine Verknüpfung von nebenläufigen, prozeßorientierten und objektorientierten Konzepten erfolgen. Objektorientierte Modellierungssprachen müssen diese Entwicklung berücksichtigen und Integrierungsmöglichkeiten dieser Konzepte in ihr Gesamtkonzept erlauben.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Die Überprüfung ist darauf zu richten, welche Sprachkonzepte/Diagrammarten zur Abbildung nebenläufiger Prozesse vorhanden sind. Ein weiteres Evaluationskriterium ergibt sich durch die Feststellung, ob spezielle Diagrammtechniken zur Modellierung von Prozessen in bestimmten Anwendungsbereichen (also z.B. von Geschäftsprozessen im Büro- und Verwaltungsbereich) angeboten werden.

---

1. Vgl. z.B. [Jac92].
2. OBA steht für Object Behavior Analysis [GoRu92].
3. Vgl. z.B. [Bur94].

4.3.2.5 Integration mit anderen Paradigmen

Neben den objektorientierten Konzepten gibt es noch weitere Konzepte, die aus anderen Vorgehensweisen übernommen wurden. Häufig entwickeln diese Konzepte aber mit dem objektorientierten Paradigma in Konflikt stehende oder konkurrierende Sichtweisen. Von ihrer Intention sind diese Konzepte meist unabhängig von der Objektorientierung. Ungewiß ist auch, ob diese Paradigmen in das objektorientierte Paradigma eingeordnet werden sollen oder ob das objektorientierte Paradigma als Teilbestand anderer Paradigmen anzusehen ist.

Konsequenzen

Inwieweit andere Konzepte in eine objektorientierte Modellierungssprache aufgenommen werden sollten, ist fraglich. Grundsätzlich ist es aber als positiv anzusehen, wenn eine Modellierungssprache Schnittstellen zu alternativen Konzepten aufweist.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Bei der Bewertung der externen Konzepte wird nicht deren Umfang, sondern nur die Eindeutigkeit ihrer Definition und die Integration dieser Konzepte in das Objektmodell berücksichtigt.

4.3.3 Konkrete Syntax

Für die zwischenmenschliche Kommunikation und als Repräsentationsmittel hat die graphische Notation eine große Bedeutung. Die Entscheidung, ob eine Notation richtig gewählt wurde, hängt von vielen Kriterien ab und umfaßt kognitive, biologische, physische und technische Aspekte (s. 2.1). Die Einschätzung und Bewertung der Notationselemente ist zudem stark subjektiv und von den Präferenzen des jeweiligen Betrachters abhängig. So kann die Frage, ob Symbole oder textuelle Beschriftungen an Beziehungskanten verwendet werden sollen, unterschiedlich beantwortet werden. Symbole sind stellenweise übersichtlicher und verständlicher. Sie müssen aber erinnert werden. Text kann auch für den gelegentlichen Anwender gut verständlich sein, aber es ist schwieriger, mehrere Aspekte parallel zu erfassen.

Die Anschaulichkeit einer graphischen Notation wird tendenziell durch folgende Kriterien gefördert (vgl. dazu auch [Rum96a]):

- Eindeutige Zuordnung zwischen Sprachelement und Notation
- Kein Überladen (Mehrfachverwenden) von Symbolen
- Einheitliche Zuordnungsregeln
- von Hand zeichenbar.
- gut druckbar: auf Papier repräsentierbar (zweidimensional), Auflösung (z.B. Fax), Schwarz-Weiß/Farbe
- in sich schlüssig und konsistent
- keine zu große Ähnlichkeit zwischen einzelnen Notationselementen
- leicht erlernbar und merkbar
- einfache Notation für die am häufigsten benutzten Sprachelemente

Einige dieser Anforderungen stehen im Widerspruch zueinander. Sollen Notationselmente nicht mehrfach verwendet werden, dann bedingt dies eine recht große Anzahl von Notationselmenten. Viele Notationselemente verletzen aber die Forderung nach Einfachheit. Für die Darstellung der Notationselemente stehen verschiedene graphische Darstellungsmittel wie Farbe, Form, Schriftart, Linienart und -breite, Anordnung und Markierungen zur Verfügung. Die obigen Regeln schränken die Verwendung der Gestaltungsmittel ein. Monochrome Darstellungen bewirken den Verzicht auf Farbe. Komplizierte Symbole lassen sich nur schwierig per Hand zeichnen. Die zu große Ähnlichkeit zwischen Diagrammelementen kann zu Verwechslungen oder Mißinterpretationen führen. Es kann aber auch durch fehlerhaftes Zeichnen oder Weglassen von Zeichendetails schnell ein neues Symbol entstehen. Die Notation

sollte so gewählt sein, daß auch bei geringfügigen Zeichenfehlern ein richtiges Erkennen des Notationselementes noch möglich ist. Beim Anwenden von Werkzeugen sinkt die Verwechslungsgefahr, da Zeichenfehler reduziert werden. Grundsätzlich ist zu berücksichtigen, daß die genannten Anforderungen keine hinreichenden Bedingungen für eine anschauliche Notation darstellen.

Die Anschaulichkeit einer Modellierungssprache wird nicht zuletzt auch durch die jeweils angebotenen speziellen Diagrammarten geprägt. Spezielle Diagrammarten erlauben die Einschränkung der verschiedenen Verknüpfungen zwischen den Diagrammen und können dadurch die Notation wesentlich vereinfachen. Auch erlauben verschiedene Diagramme eine differenziertere Betrachtung und Betonung von Sachverhalten. Objekte mögen z.B. in verschiedenen Diagrammen mit verschiedenen Darstellungsformen einhergehen. Die obige Frage kann unter diesem Blickwinkel also auch anders beantwortet werden. Die verschiedenen Diagramme sollen helfen, die statisch-strukturellen und die dynamisch-funktionalen Aspekte eines Modells zu beschreiben. Sie sollen auch in den verschiedenen Phasen der Software-Entwicklung einsetzbar sein. Sie dürfen aber nicht zu kompliziert und speziell sein, da sie sonst den Anwender einschränken. Die Diagramme können in Gruppen klassifiziert werden, wobei ein Diagramm in verschiedene dieser Gruppen gleichzeitig eingeordnet werden kann. Die wichtigsten Diagrammgruppen sind:

- Diagramme für die Anforderungen
- Diagramme für die Systemarchitektur
- Diagramme für die Objektstruktur
- Diagramme für die Klassenstruktur
- Diagramme für das Zustandsverhalten
- Diagramme für Interaktionen und Nachrichtenaustausch

Vielfach ist es zu aufwendig, alle Informationen eines Modells durch Diagramme auszudrücken. Die Verwendung von Annotationen für die einzelnen Sprachelemente ist ein geeignetes Mittel, alle Informationen zu einem Sprachmittel zentral darzustellen. Gleichzeitig können diese Annotationen die Funktion eines *Data Dictionary* übernehmen.

Konsequenzen

Die graphische Notation einer Modellierungssprache sollte den genannten Anforderungen genügen. Hinsichtlich des unzureichenden Wissens über die Wahrnehmung und Interpretation graphischer Darstellungen ist es zudem wünschenswert, wenn die jeweils vorgeschlagene Notation empirisch evaluiert wurde. Eine Modellierungssprache sollte für alle ihre Sprachkonzepte Annotationen bereitstellen. Diese Annotationen sollten alle wichtigen Informationen für ein Konzept enthalten. Die Annotationen können formale aber auch informale Informationen umfassen.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Hier sind einerseits offenkundige Verstöße gegen die gelisteten Anforderungen zu identifizieren, andererseits ist zu prüfen, ob textuelle Annotationen zur Anreicherung von Modellen möglich sind. Bei der Untersuchung der Diagrammarten ist auf Redundanz, richtige Auswahl und zweckgemäße Anwendung der Diagramme zu achten.

4.3.4 Benutzungssichten auf Modellierungssprachen

Die Anwendersicht ist die ursprüngliche Betrachtungsweise einer Modellierungssprache. Der Benutzer verwendet die Sprache, um seine Modelle aus dem Anwendungsbereich darzustellen. Das Hauptanliegen von Modellierungssprachen ist die Unterstützung der Teamkommunikation und - im Hinblick auf die Software-Entwicklung - die exakte Beschreibung von Modellen. Die Unterstützung für Werkzeuge steht dabei nicht im Vordergrund. Die Metasicht betrachtet die Modellierungssprache selbst und dient der Erweiterung, Anpassung und Präzisierung von Sprachmitteln. Bei der Betrachtung von Modellie-

rungssprachen und der ständigen Weiterentwicklung im Bereich Software-Entwicklung gewinnt diese Komponente zusätzliche Bedeutung.

4.3.5 Modellierungszweck

Die Beurteilung des Sprachumfangs kann nicht unabhängig vom Anwendungszweck gesehen werden. Der wesentliche Zweck der konzeptuellen Modellierung ist die Unterstützung der Software-Entwicklung. Im Rahmen dieser Gesamtaufgabe sind, je nach Phase und Anwendungsdomäne, unterschiedliche Ziele mit der Erstellung von Modellen verbunden. Die verschiedenen Modelle sollten den gesamten Software-Lebenszyklus abdecken. Eine Modellierungssprache für alle Entwicklungsphasen bei der Software-Erstellung muß korrespondierende Konstrukte für die Analyse, den Entwurf, die Implementierung und das Testen bereitstellen. Die Anforderungen an die Detaillierung und Formalisierung eines Modells von Phase zu Phase variieren. Dessen ungeachtet sollte der Übergang zwischen Modellen, die aufeinanderfolgenden Phasen zugeordnet sind, möglichst ohne Verlust von Semantik möglich sein. Außerdem sollte es möglich sein, die Darstellung eines Modellaspekts in vorhergehenden Phasen zurückzuverfolgen ("traceability"). Schließlich sollten Modelle geeignet sein, die Komplexität der Systementwicklung zu reduzieren. Im Hinblick auf die Software-Entwicklung ist an dieser Stelle noch eine weitere Forderung zu stellen: Die erstellten Modelle sollten dem objektorientierten Paradigma entsprechen.

Im Zusammenhang mit der Gestaltung betrieblicher Informationssysteme kommt Modellen häufig die zusätzliche Funktion zu, den betrachteten Wirklichkeitsausschnitt vor dem Hintergrund betriebswirtschaftlicher Anforderungen zu analysieren und neu zu gestalten. Dazu sollte ein Modell die für solche Aufgaben benötigten Angaben in angemessener Abstraktion beinhalten. Unter gewissen Voraussetzungen kann es zudem wünschenswert sein, Simulationen durchzuführen, um so die Bewertung alternativer Gestaltungsoptionen zu unterstützen.

Konsequenzen für Modellierungssprachen

Grundsätzlich sollte eine Modellierungssprache die Sprachmittel bereitstellen, die ein Modellierer benötigt, um den Detaillierungs- und Formalisierungsgrad zu wählen, den er für angemessen hält. Daraus folgt z.B. die Anforderung, in allen Phasen (genauer: den mit ihnen korrespondierenden Diagrammarten), vor allem aber in der späten Entwurfsphase, Integritätsbedingungen in einem sinnvoll erscheinenden Maß formal spezifizieren zu können. Gleichzeitig sollte allerdings eine formale Spezifikation nicht erzwungen werden. Die Berücksichtigung des objektorientierten Paradigmas erfordert einerseits die Bereitstellung von Sprachkonzepten, die gemeinhin als konstituierend für die Objektorientierung angesehen werden (s.u.). Komplexe Modelle sind ohne Werkzeugunterstützung kaum zu pflegen. Es ist deshalb wünschenswert, daß die Sprachbeschreibung den Entwurf von CASE-Werkzeugen möglichst gut anleitet. Daneben ist hier an weitere Konzepte wie abstrakte Klassen oder Metaklassen zu denken. Um die Reduktion der Komplexität von Modellen zu unterstützen, sind geeignete Modularisierungs- bzw. Kompositions-/Dekompositionskonzepte zu fordern.

Die Analyse und Gestaltung der Wirklichkeit wird u.a. unterstützt durch Sprachkonzepte, die die Erfassung dazu geeigneter Informationen erleichtern - z.B. zur Erfassung organisatorischer Stellen und der ihnen zugeordneten Kosten. Andererseits ist hier an spezielle Diagrammarten, etwa zur Abbildung von Geschäftsprozessen, zu denken. Simulation erfordert u.a. die Möglichkeit, neben Klassen auch (prototypische) Instanzen sowie die für eine Simulation wichtigen Zielfunktionen und Nebenbedingungen darstellen zu können.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Im Hinblick auf das Spezifikationspotential einer Modellierungssprache ist zu prüfen, ob - ergänzend zu graphisch repräsentierten Modellierungskonzepten - eine Spezifikationssprache angeboten wird. Zur Beurteilung der Mächtigkeit einer solchen Sprache bietet sich der Vergleich mit den bekannten Klassen formaler Sprachen an. Anwendungsspezifische Diagramme können von unterschiedlicher

Qualität sein. Dennoch werden wir hier lediglich feststellen, ob es derartige Diagramme gibt und von ihrer Evaluierung absehen. Ähnlich soll die Unterstützung von Simulationen betrachtet werden: Sie gilt nur dann als gegeben, wenn entsprechende Konzepte explizit vorgesehen sind.

Bei der Betrachtung von Modellierungssprachen ist es wichtig, ihren Verwendungszweck zu berücksichtigen. Es ist unsachlich, einer Modellierungssprache fehlende Implementierungskonzepte vorzuwerfen, wenn der Verwendungszweck auf die Analyse zugeschnitten ist.

4.3.6 Besonderheiten, die aus dem Software-Entwicklungsprozeß erwachsen

Aus Sicht der Software-Entwicklung ist es wünschenswert, alle Phasen der Software-Entwicklung mit einer objektorientierten Modellierungssprache zu unterstützen. Allerdings gibt es während der Software-Entwicklung Tätigkeiten, die sich nur wenig auf die Modellierung beziehen. Ein Ansatz für eine gute Unterstützung des Software-Entwicklungsprozesses ist daher:

- die Bereitstellung phasenspezifischer Konzepte
- die Transformationen zwischen den phasenspezifischen Konzepten
- die Berücksichtigung der unterschiedlichen Zielsetzungen bei den einzelnen Tätigkeiten

Zusätzlich müssen Navigationsmöglichkeiten integriert sein, die es erlauben, die Entwicklung und die Ausdrucksweise eines Anwendungsbereiches nachvollziehen zu können. Dies kann mit normalen textuelle Anmerkungen, Hypertext oder Versionsverwaltung unterstützt werden. Offen ist hier jedoch die Frage, wieviel Unterstützung eine Modellierungssprache hier bieten soll oder ob dies nicht eher in den Aufgabenbereich von Werkzeugen fällt.

Die Software-Entwicklung umfaßt viel mehr, als durch objektorientierte Techniken abgedeckt werden kann. Dies gilt sowohl für Teilphasen der Software-Entwicklung wie Testen, Organisation des Software-Entwicklungsprozesses, aber auch für spezielle Anwendungsbereiche, wie Datenbankanwendungen, wo Sicherheits- und Zugriffsaspekte eine wichtige Rolle spielen. Es muß daher abgewägt werden, ob Modellierungssprachen sich auf die Modellbildung konzentrieren sollen oder als Dokumentationsvehikel für alle Entwicklungsaufgaben dienen sollen.

Zur Software-Entwicklung zählen auch *Planung und Projektmanagement*. Die Aufgaben für Planung und Projektmanagement sind jedoch stark verschieden von den typischen Entwicklungsaufgaben. Mit der Forderung, auch das Projektmanagement in eine objektorientierte Modellierungssprache zu integrieren, wird ein Teil aufgenommen, der nur wenig Bezug zum objektorientierten Modellieren hat. Im allgemeinen erfolgt eine Aufteilung und Zerlegung einer Aufgabe in Teilaufgaben, um besser abstrahieren und organisieren zu können. Die zusätzliche Integration des Projektmanagements widerspricht dieser Regel. Statt dessen sollten im Projektmanagement Vorgehensweisen für die objektorientierte Software-Entwicklung bereitgestellt werden. Diese Trennung erlaubt die Entwicklung differenzierter Verfahren (vgl. Abschnitt 3.1, S. 17). Für eine objektorientierte Modellierungssprache ist es günstiger, sich auf ihr zentrales Anliegen, das objektorientierte Entwickeln und Beschreiben von (Software-) Systemen und die Einordnung ihrer Artefakte in den allgemeinen Entwicklungsprozeß zu konzentrieren. Deshalb gehört die Unterstützung für Planung und Projektmanagement nicht zu den Anforderungen an eine Modellierungssprache.

Unterstützung für Analysetätigkeiten

Für die frühe Phase der Software-Entwicklung haben sich verschiedene Techniken und Verfahren etabliert. Dies sind Techniken zur Ist-Analyse eines Systems, zur Anforderungsanalyse und -spezifikation. Das objektorientierte Paradigma ermöglicht die Benutzung objektorientierter Begriffe während der Analyse. Es ist jedoch wichtig zu beachten, daß diese Begriffe meistens mit einer informaleren Bedeutung als später im Entwurf oder der Implementierung benutzt werden. In der Analyse ist Informalität in vielen Fällen erwünscht, da sich Begriffe und Konzepte erst schrittweise manifestieren müssen. Aus Sicht des Modellierers ist es wichtig, verschiedene Alternativen eines Sachverhaltes veran-

schaulichen zu können, inkonsistente, unexakte und mehrdeutige Modelle zu erlauben und die Sprechweisen des Anwendungsbereiches zu unterstützen. So sollte zum Beispiel eine gleichartige Behandlung von Objekten und Subsystemen möglich sein. Neben dieser Informalität werden aber auch formalere Ansätze für System- oder Anforderungsspezifikationen benötigt.

Unterstützung für Entwurfstätigkeiten
Das Modellieren hat eine enge Beziehungen zum Analysieren und Entwerfen. Im Rahmen dieser Tätigkeiten hat der Einsatz von Modellen eine lange Tradition, was wohl darauf zurückzuführen ist, daß Modell durch adäquate Abstraktionen geeignet sind, die Komplexität von Analyse- und Entwurfsaufgaben zu verringern und gleichzeitig eine (mehr oder weniger) anschauliche Dokumentation darstellen. Wie schon für die Analysetätigkeiten sollte es auch für die Entwurfstätigkeiten möglich sein, den Formalisierungsgrad der Darstellung an die Erfordernisse anpassen und schrittweise erhöhen zu können.

Unterstützung für Implementierungstätigkeiten
Da Modelle möglichst unabhängig von der gewählten Implementierungssprache sein sollen, müssen Modellierungssprachen keine speziellen Programmiersprachenkonzepte umfassen. Es werden aber Regeln benötigt, wie Entwurfskonzepte auf Programmiersprachenkonzepte abgebildet werden. Dies bezieht sich meistens auf Deklarationsangaben. Zusätzlich ist es aber hilfreich, Entwürfe für die Realisierung mit Implementierungsdetails anreichern zu können. Für die Implementierung ist daher eine eindeutige und meist wesentlich detailliertere Darstellung als für die Analyse und den Entwurf notwendig.

Unterstützung für Verifikation und Validation
Im Bereich Verifikation und Validation bieten heutige objektorientierte Modellierungssprachen wenig Unterstützung. Zum einen können hier Navigationsmöglichkeiten zur Analyse der Entwicklung eines Modells Abhilfe schaffen. Diese erlauben es zum Beispiel, die Umsetzung von Analyseanforderungen zu verfolgen. Eine andere Möglichkeit ist die Integration von Testfällen in die Modellierungssprache, die mit verschiedenen Sprachkonzepten verknüpft werden können. Diese können jedoch auch als externe Dokumentation gewertet werden.

Konsequenzen
Für die Analyse werden formale und informale Ausdrucksmöglichkeiten benötigt. Die informalen Beschreibungen werden zum Experimentieren, Ausprobieren und Sammeln von Informationen benötigt und die formalen Beschreibungen zum Festhalten der Ergebnisse. Gleichzeitig muß es möglich sein, informale Aussagen nachträglich formalisieren zu können. Die Modellierungssprache muß Beschreibungsformen für die einzelnen Phasen bereitstellen und deren Integration zu einer einheitlichen Gesamtsicht sichern (vgl. dazu auch [Dod96]).

4.4 Generelle Kriterien

Im folgenden werden Kriterien vorgestellt, die helfen, eine Modellierungssprache zu bewerten. Dabei handelt es sich um Eigenschaften und deren Ausprägung in einer Modellierungssprache. Sie können auf die Modellierungssprache in ihrer Gesamtheit, aber auch auf einzelne Bestandteile oder Sprachmittel angewendet werden. In vielen Fällen bestehen zwischen den einzelnen Eigenschaften enge Beziehungen. Je nachdem für welchen Sprachbestandteil die Bewertung vorgenommen wird, können andere Maße für die Bewertung verwendet werden. Für viele der hier vorgestellten Merkmale gibt es jedoch keine exakten Bewertungsverfahren, inwieweit das Merkmal erfüllt ist oder nicht.

4.4.1 Anwenderbezogene Kriterien

Modellierungssprachen bieten die Möglichkeit, Ideen und Gedanken mittels Modelle austauschen zu können. Modellierungssprachen sind Sprachen, die primär für menschliche Anwender als Mittel zur

Kommunikation bestimmt sind. Die anwenderbezogenen Kriterien beleuchten dieses Verhältnis zwischen Modellierungssprache und Anwender.

4.4.1.1 Anschaulichkeit und Verständlichkeit

Hier geht es einerseits um eine dem Modellbegriff schon fast inhärente Forderung: Ein Modell sollte *anschaulich* sein. Andererseits ist hier an den Umstand zu denken, daß Modelle der Abbildung faktischer oder geplanter Realität dienen. Solche Abbildungen können grundsätzlich fehlbar sein. Ein Modell sollte seine Überprüfung an der Realität unterstützen.

Ein Modell ist dann anschaulich, wenn es möglichst direkt mit Wahrnehmungsmustern und Konzeptualisierungen des Betrachters korrespondiert. Wahrnehmungsmuster und Konzeptualisierungen streuen aber bekanntlich interpersonell und intrapersonell (der Mensch ist lernfähig). Anschaulichkeit in diesem Sinn hängt einerseits den jeweils gewählten Abstraktionen und Bezeichnern ab, andererseits von der jeweiligen Modellierungssprache (worauf noch einzugehen sein wird). Die Beurteilung der Beziehung zwischen Modell und Realität liegt i.d.R. allein bei den Betrachtern. Die Anschaulichkeit eines Modells ist also grundsätzlich geeignet, seine Überprüfbarkeit zu fördern. Dabei ist es wesentlich, daß das Verhältnis des Modells zur Realität durch möglichst genaue Abbildungsvorschriften erläutert wird.

Konsequenzen für Modellierungssprachen

Da Wahrnehmungsmuster und Konzeptualisierungen weit und in subtiler Weise streuen, sind generelle Aussagen über die Wirkung einer Modellierungssprache auf die Anschaulichkeit eines Modells kaum möglich. Allgemein läßt sich vermuten, daß Syntax und Semantik einer Modellierungssprache tendenziell dann als anschaulich empfunden werden, wenn sie eine deutliche Nähe zu Sprachmitteln aufweisen, die der jeweilige Betrachter aus ihm vertrauten Sprachen kennt - also beispielsweise das generelle Muster "Subjekt, Prädikat, (Objekt)". Der Stand der Forschung in diesem Bereich muß allerdings als unbefriedigend angesehen werden. Es gibt nur wenige empirische Untersuchungen, die die Anschaulichkeit von konzeptuellen Modellen aus der Sicht verschiedener Betrachter zum Gegenstand haben (so etwa [GoSt90], [Hit95]). Dabei hat sich gezeigt, daß viele Betrachter erhebliche Schwierigkeiten hatten, die ihnen vorgelegten Modelle nachzuvollziehen. Ein solches Ergebnis kann kaum überraschen: Schließlich wurden Modellierungssprachen i.d.R. nicht nach den Wünschen möglicher Anwender entworfen. Wir können an dieser Stelle lediglich zweierlei festhalten: Je mehr Diagrammarten geboten werden, desto größer die Wahrscheinlichkeit, daß eine Diagrammart im Kreis der Betrachter intuitiv verstanden wird. Je stärker eine Diagrammart die spezifischen Besonderheiten einer Domäne berücksichtigt, desto größer ist die Wahrscheinlichkeit, daß die Kenner dieser Domäne entsprechende Diagramme für anschaulich halten. Anschaulichkeit impliziert sinnvollerweise das Verstehen der verwendeten Modellierungssprache. Das empfiehlt die Forderung nach leichter Erlernbarkeit: Je geringer die Zahl der Modellierungskonzepte und je einfacher die zu berücksichtigenden Verknüpfungsregeln, desto leichter ist eine Sprache zu erlernen. Dabei muß allerdings berücksichtigt werden, daß diese Anforderung ggfs. im Konflikt mit dem Bemühen um Anschaulichkeit steht.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Auch wenn Modellierungssprachen unzweifelhaft einen erheblichen Einfluß auf die Anschaulichkeit von Modellen haben, ist uns die Erfassung und Beurteilung dieses Einflusses in befriedigender Weise nicht möglich. Wir können lediglich auf einige Indikatoren hinweisen. Die verschiedenen Diagrammarten verbessern tendenziell die Chance, individuell als anschaulich empfundene Sichten zu befriedigen. Diesem Vorteil steht allerdings ggfs. der Nachteil redundanter Sprachelemente gegenüber, wodurch die Erlernbarkeit der gesamten Sprache verschlechtert wird. Weiterhin sind *domänenspezifische Diagrammarten* geeignet, Anschaulichkeit für bestimmte Betrachtergruppen zu fördern. Als Beispiel sei eine Diagrammart zur Abbildung von Geschäftsprozessen im Rahmen der Entwicklung betrieblicher Informationssysteme genannt.

Darüber hinaus sind Symbole zur Kennzeichnung ergänzender Kommentare wünschenswert. Die Erlernbarkeit einer Sprache wird durch die *Dokumentation* erleichtert. Neben der verständlichen Beschreibung der Sprachkonzepte ist hier an die Erläuterung der Verwendung der Sprache zu denken - ggfs. durch *Beispiele* ergänzt.

Auch die Verständlichkeit der Sprache wirkt sich direkt auf die Anschaulichkeit der Sprache aus. Die Verwendung der einzelnen Sprachmittel sollte klar festgelegt sein, und es sollten Entscheidungshilfen bei Alternativen gegeben werden. Auch die Eindeutigkeit der Semantik der Sprachmittel ist eine wichtige Voraussetzung für eine verständliche Sprache.

4.4.1.2 Angemessenheit

Das Kriterium der Angemessenheit steht in direktem Zusammenhang mit der Mächtigkeit und dem Anwendungszweck einer Sprache. Die Angemessenheit ist immer relativ zum Anwendungszweck zu sehen. Sie bezieht sich sowohl auf die Sprache in ihrer Gesamtheit als auch auf die einzelnen Sprachkonzepte. Angemessenheit und Mächtigkeit bieten zusammen die Möglichkeit, zu untersuchen, ob eine Sprache für den Anwendungszweck ausreichende Konzepte bereitstellt, gleichzeitig aber keine überflüssigen Konzepte enthält.

Konsequenzen für Modellierungssprachen

Insbesondere bei sehr speziellen Sprachkonstrukten muß untersucht werden, ob diese Konstrukte nötig sind oder ob sie nur zu einer Erhöhung der Sprachkomplexität führen. Die Entscheidung, ob ein Sprachmittel angemessen ist oder nicht, ist jedoch häufig nicht objektiv vornehmbar, sondern hängt stark von den Präferenzen der jeweiligen Betrachter ab. Im allgemeinen gibt es immer Vor- und Nachteile für einzelne Sprachmittel, so daß ein sorgsames Abwägen notwendig ist.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Generelle formale Maße zur Bestimmung der Angemessenheit von Modellierungssprachen gibt es nicht. Die Untersuchung der Angemessenheit eines Sprachelementes kann daher nur subjektiv erfolgen. Dabei ist zu untersuchen, ob ein Sprachelement bedeutsam für den Anwendungszweck ist oder nicht.

4.4.1.3 Überprüfbarkeit

Modelle dienen der Abbildung faktischer oder geplanter Realität. Solche Abbildungen können grundsätzlich fehlbar sein. Ein Modell sollte seine Überprüfung an der Realität unterstützen. Der Einfluß der Modellierungssprache auf die Überprüfbarkeit eines Modells ist ambivalent. So geht es einerseits darum, eine möglichst genaue, intersubjektiv nachvollziehbare Überprüfung an der Wirklichkeit anzustreben. Andererseits ist zu berücksichtigen, daß solche objektivierten Verfahren nicht für alle Facetten eines Modells anwendbar sind und sich Modelle oft nicht auf faktische, sondern geplante Realität beziehen. In diesen Fällen erfolgt die Überprüfung allein durch die Einschätzung der Betrachter. Idealtypisch ist ein Modell dann als realistisch (im Hinblick auf Erwartungen) anzusehen, wenn im Kreise gutwilliger und sachkundiger Betrachter ein Konsens darüber erzielt wurde. Während die objektivierte Überprüfung eines Modells an der Realität tendenziell zur Forderung nach formalen Modellierungssprachen führt, ergeben sich aus dem Bemühen um Konsensbildung andere Konsequenzen: Bekanntlich fördert Mehrdeutigkeit die Chance, einen Konsens zu erzielen. Ein Modellierungsansatz wie die "Soft System Methodology" ([Che81]) läßt den Betrachtern eben wesentlich mehr individuelle Interpretationsspielräume als eine Sprache zur formalen Systemspezifikation. Wegen dieser widersprüchlichen Konsequenzen ist das Kriterium "Überprüfbarkeit" nicht zur Diskriminierung zwischen Modellierungssprachen geeignet.

Ein Modell ist eine vereinfachte und idealisierte Abbildung eines Problembereichs. Im Modell werden nur die für die Betrachtung wesentlichen Eigenschaften berücksichtigt und unwesentliche vernachlässigt. In Abb. 2 ist die Beziehung zwischen Modell und zu modellierendem Sachverhalt dargestellt.

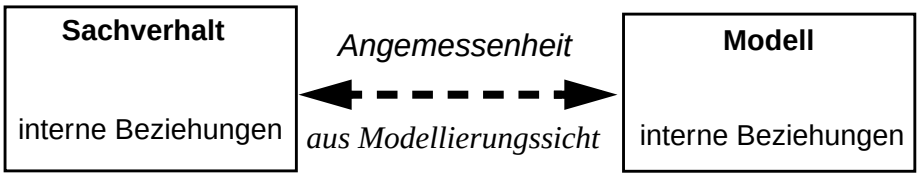

Abb. 2: Beziehung zwischen Modell und zu modellierendem Sachverhalt

Modelle bedürfen immer der Überprüfung mit dem zu modellierenden Sachverhalt. Zwischen Sachverhalt und Modell existiert immer eine Beziehung, die ein Maß für die Angemessenheit des Modells ist. Die Bewertung der Angemessenheit eines Modells kann nur relativ zum Modellierungszweck erfolgen, da Modelle von unwesentlichen Details abstrahieren. Im allgemeinen kann diese Beziehung nicht direkt angeben werden, da der Sachverhalt meistens nur unklar spezifiziert ist[1]. Aus diesem Grund ist es auch nicht möglich, immer automatisch zu überprüfen, ob ein Modell den gewünschten Sachverhalt modelliert. Nur wenn der Sachverhalt selbst als exakte Beschreibung[2] vorliegt und auch das Modell exakt beschrieben ist, wird ein formales Überprüfen erst möglich, und die Angemessenheit eines Modells kann formal bewertet werden. Meistens ist der Sachverhalt jedoch nicht klar spezifiziert.

Die einzige Möglichkeit ist dann, den Inhalt des Modells zu erfassen und zu überprüfen, ob dieser mit dem zu modellierenden Sachverhalt übereinstimmt. Je exakter und präziser ein Modell interpretiert werden kann, desto leichter kann es mit dem zu modellierenden Sachverhalt verglichen werden. Obwohl es nicht möglich ist, die Adäquanz zwischen einer eventuell unpräzisen Darstellung (Sachverhalt) und einer möglichst präzisen Darstellung (Modell) prinzipiell zu entscheiden, hilft eine präzise Darstellung und eindeutige Interpretation des Modells beim Vergleich mit dem zu modellierenden Sachverhalt. Eine Modellierungssprache, die es ermöglicht, Modelle exakt und eindeutig zu beschreiben und zu interpretieren, erleichtert daher wesentlich die Überprüfung des Modells.

Nur wenn Modelle eindeutig sind, ist eine korrekte Verständigung möglich. Dies verlangt eine eindeutige Interpretation der einzelnen Sprachmittel und Regeln zur syntaktischen Verknüpfung dieser Sprachmittel. Andernfalls bieten sich Interpretationsfreiräume, die zu Mißverständnissen führen können. Die exakte Festlegung der abstrakten Syntax und Semantik der Sprache ist somit Voraussetzung für die eindeutige Interpretierbarkeit der Modelle. Beim Betrachten eines Artefakts der Modellierungssprache hat man im allgemeinen nicht wie bei Programmen die Möglichkeit, durch Inspizieren oder Ausprobieren des Codes die Aufgabe der Software herauszubekommen. Vielmehr ist die Beschreibung der Metasprache die einzige Referenz.

Ein anderer Aspekt der Überprüfung betrifft die Nachvollziehbarkeit und Rückverfolgung von Modellierungsentscheidungen.

4.4.1.4 Mächtigkeit

Die Mächtigkeit einer Modellierungssprache bezieht sich unter anderem auf die Ausdrucksmächtigkeit der bereitgestellten Konzepte. Eine Modellierungssprache sollte sich auf die Modellierung konzentrieren. So können zu viele Konzepte zu einer unangemessen mächtigen Sprache führen. Zwischen Mächtigkeit und Angemessenheit muß daher ein ausgewogenes Verhältnis existieren.

Mächtigkeit kann aber auch relativ zur Mächtigkeit von Turingmaschinen betrachtet werden und ist dann ein Maß für die Berechnungsfähigkeit einer Modellierungssprache. Für Modellierungssprachen ist diese Betrachtung jedoch nicht ganz korrekt, da sie im allgemeinen keine Formalisierungen von Algorithmen sind.

---

1. Gerade ein Modell kann ja dazu dienen, den Sachverhalt klarer auszudrücken und zu spezifizieren.
2. Dies ist z. B. der Fall, wenn eine formale Spezifikation in eine andere überführt wird.

Konsequenzen
Mächtigkeit und Angemessenheit bedingen einander. Die Mächtigkeit einer Sprache muß daher relativ zum Anwendungsbereich betrachtet werden. Geht man grundsätzlich davon aus, daß die hier betrachteten Modellierungssprachen der konzeptuellen Modellierung dienen, mißt sich die Mächtigkeit einerseits an der Möglichkeit, als wesentlich erkannte Sachverhalte des abzubildenden Realitätsbereichs natürlich, also ohne aufwendige Rekonstruktionen, beschreiben zu können. Andererseits sollte die Modellierungssprache auch Möglichkeiten bieten, Informationen darzustellen, die für die Implementierung benötigt werden.

4.4.2 Modellinterne Kriterien

Nach seiner Erstellung sollte ein Modell nicht gegen vorhandene syntaktische Regeln verstoßen. Es sollte vollständig und widerspruchsfrei sein. Dabei handelt es sich an dieser Stelle um Kriterien, die sich allein auf das Modell, nicht den jeweils abgebildeten Realitätsausschnitt beziehen. Vollständigkeit bezieht sich in diesem Fall also allein auf die Frage, ob alle im Modell benutzten Modellierungskonzepte definiert sind oder ob Konzepte verwendet werden, für die es keine Definitionen gibt. Desweiteren sollte ein Modell keine Redundanzen enthalten. Wenn ein Modell verschiedene Diagrammarten enthält, sollten diese orthogonal sein: Änderungen in einem Diagramm sollten die Konsistenz anderer Diagramme nicht berühren (ausgenommen davon ist das Löschen von Modellelementen, die in mehreren Diagrammen vorkommen und für die Integration der Diagramme wesentlich sind). Weitere Anforderungen an die interne Struktur eines Modells ergeben sich aus dem Modellierungszweck. Sie lassen sich zum Teil in Metriken ausdrücken (s.u.).

Konsequenzen für Modellierungssprachen
Die genannten Anforderungen können gewiß nicht allein von einer Modellierungssprache garantiert werden. Sie kann allerdings dazu beitragen, indem sie die folgenden Prinzipien erfüllt (vgl. [SüEb97], S. 2 f.): Die Beschreibung einer Modellierungssprache mittels eines Metamodells sollte ein *korrektes* Modell der Sprache liefern: Die Sprachbeschreibung sollte die Identifikation unzulässiger Modelle gewährleisten und gleichzeitig erlauben, die Menge aller zulässigen Modelle zu generieren. Die Sprachbeschreibung sollte zudem *vollständig* sein, d.h. alle in der Sprache verwendeten Konzepte sowie die Bedingungen ihrer Verwendung sollten definiert sein. Eine Metamodell sollte auch dem *Prinzip der Klarheit/Einheitlichkeit* genügen. Das umfaßt allgemein die Forderung nach einer verständlichen Darstellung des Metamodells, konkret die Forderung danach, daß ähnliche Konzepte auch in ähnlicher Weise im Metamodell repräsentiert werden. Schließlich sollte ein Metamodell so *einfach* wie möglich dargestellt werden, was u.a. die Vermeidung redundanter Informationen empfiehlt.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien
Es kann festgestellt werden, ob das Prinzip der Korrektheit erfüllt ist. Auch wenn diese Eigenschaft damit mit einem einfachen Prädikat gekennzeichnet werden könnte, ist zu berücksichtigen, daß das Prinzip der Korrektheit mehr oder weniger knapp verfehlt werden kann. Ähnlich verhält es sich mit dem Prinzip der Vollständigkeit. Das Prinzip der Klarheit/Einheitlichkeit kann - wie auch das Prinzip der Einfachheit - in seiner Gesamtheit ohne Verzicht auf subjektive Beurteilungen kaum überprüft werden. Um die Nachvollziehbarkeit entsprechender Beurteilungen zu fördern, sind offenkundige Verstöße gegen die Prinzipien explizit zu benennen.

4.4.2.1 Eindeutigkeit

Ein Modell sollte möglichst wenig Interpretationsspielräume lassen. Die Eindeutigkeit eines Modells bezieht sich immer auf die abstrakte Syntax und Semantik. Die abstrakte Syntax und Semantik wird damit Grundlage eines gemeinsamen Verständnisses. Zusätzliche Informationen, wie sie zum Beispiel in Diagramme hineininterpretiert werden, bleiben unberücksichtigt. Die Notwendigkeit nach Eindeutigkeit ergibt sich für objektorientierte Modellierungssprachen noch aus einem anderen Grund. Die objektorientierte Modellierung erfolgt meist im Hinblick auf die spätere Realisierung der modellierten

Systeme mit Hilfe von Programmiersprachen. Programmiersprachen erfordern aber die Darstellung des Sachverhaltes in Programmcode, der nur noch eine Interpretation erlaubt.

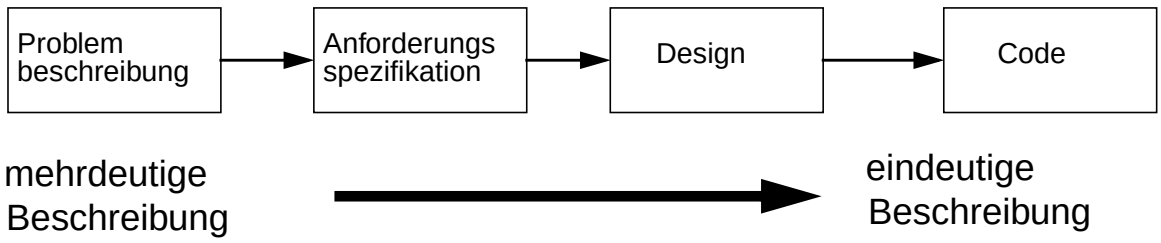

Abb. 3: Reduzierung von Mehrdeutigkeit während der Software-Entwicklung

Betrachtet man die verschiedenen Dokumente in Abb. 3, die während der Software-Entwicklung entstehen, so stellt man fest, daß in diesen Dokumenten eine schrittweise Reduzierung von Mehrdeutigkeit erfolgt. Am Ende dieser Kette stehen Programme, welche eindeutig interpretierbar sind. Bei den Übergängen müssen Entscheidungen getroffen werden, wie die Mehrdeutigkeiten aufgelöst werden sollen. Soll teilweise automatisch aus dem Entwurf Code generiert werden, so müssen die betroffenen Entwurfsfragmente schon eindeutig interpretierbar sein. Unterstützt eine Modellierungssprache die Erzeugung eindeutig interpretierbarer Dokumente nicht, so erfolgt der Übergang von der mehrdeutigen zur eindeutigen Beschreibung zwangsweise nach der Modellbildung.

Ähnliche Aussagen zur Bedeutung der Eindeutigkeit von Beschreibungssprachen findet man auch bei Schienmann. Er verweist auf die ungenügende formale Semantik vieler objektorientierter Notationen und schlägt die Einführung einer Normsprache vor, die auf der natürlichen Sprache basiert, aber einen eingeschränkten Wortschatz und eingeschränkte Grammatikregeln hat, um Mehrdeutigkeiten zu vermeiden ([Sch97], S. 74 ff.).

Die Forderung nach einer eindeutigen Interpretation der Sprachelemente soll noch differenzierter betrachtet werden. Sie verlangt die Festlegung der abstrakten Syntax und Semantik für ein Sprachelement. Die Festlegung erfolgt dabei auf mehr oder weniger formaler Basis. Häufig wird jedoch nicht die exakte Bedeutung der Konzepte[1] definiert. Die Sprachelemente gleichen dann Schablonen, deren wirkliche Bedeutung erst bei der jeweiligen Verwendung festgelegt wird. Problematisch bei dieser Vorgehensweise ist, daß zwar die Sprachelemente und ihre Darstellung bekannt sind, nicht aber die wirkliche Bedeutung der Konzepte, deren Auslegung von Betrachter zu Betrachter verschieden sein kann. Die spätere Interpretation der Sprachkonzepte kann in der Sprachbeschreibung definierte Beziehungen zwischen den Sprachelementen ad absurdum führen. Eine unpräzise Festlegung der Bedeutung der Sprachkonzepte ermöglicht somit verschiedene Interpretationen und Anwendungen, die zu Mißverständnissen führen können. Aus diesen Gründen sollte die Bedeutung der Sprachelemente in der Sprachbeschreibung exakt festgelegt werden. Der Vorteil ist, daß man nicht nur die gleiche Sprache spricht, sondern auch versteht, was der andere sagt, da man die gleiche in der Sprachbeschreibung festgelegte Interpretation benutzt.

Konsequenzen

Die abstrakte Syntax und Semantik sollte eindeutig festgelegt werden. Das Ziel ist, daß ein Satz in der Beschreibungssprache (Folge von Beschreibungsmitteln) bezüglich der abstrakten Semantik eindeutig interpretierbar sein soll. Die Überprüfung der Semantik kann analog zu Programmiersprachen erfolgen. Durch die eindeutige Interpretation der Sprachmittel und Verknüpfungsregeln ist der Benutzer in der Lage zu überprüfen, ob sein Modell den Anforderungen entspricht. Die Sprache kann prinzipiell

1. Dies findet man häufig bei objektorientierten Modellierungssprachen, wo objektorientierte Begriffe verwendet, aber unzureichend definiert werden.

nicht sichern, daß der Anwender ein zu seinem Problem relevantes Modell erstellt. Sie kann aber durch die Eindeutigkeit sichern, daß das Modell eindeutig interpretiert werden kann.

4.4.2.2 Konsistenz

Konsistenz bezieht sich auf das Modell und nicht auf das Verhältnis zwischen Modell und modellierter Realität. Die Konsistenz ist damit eine technische Eigenschaft, die die Widerspruchsfreiheit zwischen verschiedenen Konzepten behandelt. Konsistenz ist eng mit Integration verbunden.

4.4.2.3 Formalisierung

Eine Möglichkeit, um Eindeutigkeit und Konsistenz zu erreichen, ist eine formale Grundlage. Der Formalisierungsgrad einer Sprache kann unterschiedlich sein. Teile der Sprache können informal sein und andere wiederum formal. Bei der Formalisierung können verschiedene Stufen erreicht werden (formale Definition der eigenen Ansätze, Integration mit existierenden formalen Theorien, axiomatische Definition des Untersuchungsbereichs).

Konsequenzen

Eine Modellierungssprache sollte formale Sprachelemente enthalten, um konsistente und eindeutige Modelle zu erlauben. Um auch die Benutzerfreundlichkeit zu gewährleisten, sollten zusätzlich informale Beschreibungen erlaubt sein und eine spätere Formalisierung möglich sein.

4.4.2.4 Integrationsgrad

Modellierungssprachen umfassen meist eine Vielzahl recht verschiedener Konzepte. Diese Konzepte müssen zu einem einheitlichen Ganzen integriert werden. Der Integrationsgrad ist ein Maß für die Integration der einzelnen Teilsprachen. Die Integration erfolgt über geeignete Sprachmittel. Das gemeinsame Sprachmittel wird in verschiedenen Teilsprachen verwendet und dient somit als Brücke zwischen diesen.

Ein anderer wichtiger Aspekt ist die Integrationsmöglichkeit der Modellierungssprache als Teilsprache in andere Modellierungssprachen. Modellierungssprachen können unter diesem Blickwinkel als Komponenten betrachtet werden, die die Bildung neuer Modellierungssprachen erlauben. Aus dieser Perspektive ist insbesondere die Integration des objektorientierten Paradigmas mit anderen Paradigmen, Beschreibungsmitteln und Bereichen der Software-Entwicklung interessant. Dies können Echtzeit, Parallelität, HCI oder graphentheoretische Ansätze sein und bei den Beschreibungsmitteln verschiedene Formen von Automaten, *Rich Pictures* [Che81], Petrinetze oder formale Beschreibungsmittel.

4.4.3 Ökonomische Faktoren

Die Wirtschaftlichkeit der Modellierung hängt von vielen Faktoren ab. Sie umfassen sowohl Faktoren, die direkt mit der Modellierungssprache in Zusammenhang stehen, aber auch externe Faktoren, wie Werkzeugverfügbarkeit, Prozeßunterstützung und Akzeptanz der Sprache. In dieser Untersuchung sind nur die Faktoren von Interesse, die in direktem Zusammenhang mit der Modellierungssprache stehen.

4.4.3.1 Investitionsschutz

Wirtschaftlichkeit legt zudem das Bemühen um den Schutz von Investitionen nahe - sowohl vergangener als auch zukünftiger. Der Schutz zurückliegender Investitionen wird dadurch gefördert, daß eine Modellierungssprache durch Regeln für die Rekonstruktion von Modellen bzw. Verzeichnissen ergänzt wird, die in der Vergangenheit verbreitet waren. Der Schutz zukünftiger Investitionen hängt maßgeblich von der Verbreitung einer Modellierungssprache ab und ist bei einer (wirksamen) Standardisierung besonders hoch. In diesem Zusammenhang ist es auch von Bedeutung, ob am Markt Werkzeuge verfügbar sind, die eine Modellierung mit der jeweiligen Sprache unterstützen. Neben der Feststellung, ob eine Sprache standardisiert bzw. weit verbreitet ist und ggfs. durch Werkzeuge unterstützt wird, ist nach der Verfügbarkeit einschlägig qualifizierter Mitarbeiter zu fragen.

Für die Bewertung einer Modellierungssprache sind dies jedoch eher externe Faktoren, die auf die fachliche Untersuchung wenig Einfluß haben.

4.4.3.2 Wiederverwendung

Die Wirtschaftlichkeit kann durch die *Wiederverwendung* einmal geschaffener Artefakte erhöht werden. Dabei sollte sich Wiederverwendbarkeit gewiß nicht auf einzelne Konzepte (Klassen) beschränken. Wiederverwendbarkeit kann sowohl auf die Modellierungssprache selbst als auch auf die Modelle angewendet werden. Im ersten Fall bezieht sich die Wiederverwendung auf die Beschreibungsebene. Bei der Definition der Modellierungssprache werden generelle Prinzipien oder Konzepte benutzt. Dies vereinfacht die Sprache. Im anderen Fall stellt die Modellierungssprache Konzepte bereit, um innerhalb der Modelle Komponenten wiederverwenden zu können.

Konsequenzen für Modellierungssprachen

Wiederverwendung kann zum einen durch die Wiederverwendung existierender Komponenten und zum anderen durch die Ableitung neuer Komponenten auf der Basis existierender Komponenten erfolgen. Wiederverwendbarkeit wird grundsätzlich gefördert, wenn die zur Modellierung verwendbaren Konzepte Generalisierungs-, bzw. Spezialisierungsmöglichkeiten und Modularisierungskonzepte zur Komponentenbildung bieten. Daneben ist an die Darstellbarkeit abstrakter Entwurfsartefakte zu denken, wie sie etwa unter dem Schlagwort "Design Pattern" diskutiert werden. Auch die bereits erwähnten Modularisierungskonzepte können zur Wiederverwendbarkeit beitragen.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Im Hinblick auf die Wiederverwendung vorhandener Modellteile ist zu klären, für welche Konzepte eindeutige Generalisierungs-/Spezialisierungsvorschriften vorliegen. Weiterhin ist zu untersuchen, welche Sprachkonzepte Komponenten beschreiben, die in verschiedenen Situationen eingesetzt werden können.

4.4.3.3 Erweiterbarkeit

Die Erweiterbarkeit bezieht sich auf die Modellierungssprache und ist ein Maß für die Möglichkeit, eine Sprache um zusätzliche Konzepte anzureichern. Wiederverwendung und Erweiterung sind miteinander verbunden, wenn schon existierende Komponenten erweitert werden. Gleichzeitig ist mit Erweiterbarkeit auch Flexibilität und Anpaßbarkeit verbunden. Erweiterungsmechanismen erlauben die Definition neuer Sprachmittel. Dies sollte nach formalen Regeln erfolgen. Erweiterung ist damit ein Teil der Metasicht auf Modellierungssprachen.

Konsequenzen für Modellierungssprachen

Als Konsequenz aus der Erweiterbarkeit ergibt sich die Forderung nach Flexibilität: Modellierungssprachen reflektieren notwendigerweise bestimmte Vorstellungen über die in den verschiedenen Tätigkeiten, wie etwa Analyse und Entwurf, zu erhebenden bzw. festzulegenden Angaben. Da sich solche Vorstellungen im Zeitverlauf wie auch mit der je betrachteten Domäne ändern mögen, sollte eine Modellierungssprache erweiterbar sein. Im Hinblick auf die Austauschbarkeit von Modellen sollte eine solche Erweiterung die Semantik ursprünglicher Sprachmittel nicht berühren. Dazu ist es wesentlich, daß zulässige Erweiterungen im Metamodell festgelegt sind.

Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Im Hinblick auf die Flexibilität und Erweiterbarkeit sind die ggfs. vorhandenen Konzepte zur Sprecherweiterung zu benennen, die Bandbreite der Erweiterungsmöglichkeiten abzugrenzen. Ferner ist der Aufwand ihrer Verwendung zu charakterisieren.

4.5 Bezugsrahmen zur Bewertung objektorientierter Modellierungssprachen

Der hier vorgestellte Bezugsrahmen liefert eine überblicksartige Orientierung für die Bewertung objektorientierter Modellierungssprachen. Er faßt die im vorherigen Abschnitt untersuchten Anforderungen und Eigenschaften von objektorientierten Modellierungssprachen tabellenartig zusammen. Je nach Bedarf können einzelne Bewertungsmerkmale weiter verfeinert werden, falls diesem Merkmal bei der Untersuchung besondere Bedeutung zukommt.

Mit diesem Rahmen ist jedoch nicht das Ziel verbunden, objektorientierte Modellierungssprachen quantitativ zu vergleichen. Es ist also keine Art von "Algorithmus", der aus einer Auswahl von Modellierungssprachen mechanisch zur "optimalen" Modellierungssprache führt. Wie schon erklärt wurde, existieren viel zu viele Einflußgrößen aus den verschiedensten Forschungsrichtungen, die eine solche Möglichkeit derzeit - und vielleicht generell - ausschließen. Vielmehr ist es das Ziel, eine möglichst umfassende und doch kurze Charakteristik einer Sprache zu erhalten und so die Auswahl und Untersuchung von objektorientierten Modellierungssprachen zu unterstützen.

Der Rahmen ist keinesfalls als vollständig zu verstehen. Insbesondere die Bereitstellung geeigneter Bewertungsmaße und Kriterien bedarf weiterer Untersuchungen.

Tabelle 3: Bewertungsrahmen

		Merkmal	Ausprägung	Bewertung
Beschreibung	Definition	Grammatik	Vorhandensein einer Grammatik Grammatiktyp	Das Vorhandensein einer Grammatik ermöglicht die Nutzung von Erkenntnissen aus der Theorie von formalen Sprachen (Wortproblem).
		Metamodell	Art des Metamodells	Ein objektorientiertes Metamodell erlaubt die Darstellung und Umsetzung der Modellierungssprache mit objektorientierten Konzepten.
	Dokumentation	Sprach-spezifikation	Strukturierung Eindeutigkeit/Exaktheit Vollständigkeit	Die Sprachdefinition beschreibt die abstrakte Syntax und Semantik, aber auch die konkrete Syntax. Ihr kommt eine große Bedeutung zu.
		Anwender-dokumentation	Erlernbarkeit Anschaulichkeit	Die Anwenderdokumentation zeigt die Sprache aus Anwendersicht und enthält Beispiele für die Benutzung der einzelnen Sprachelemente.
	Aufbau	Grundkonzeption	monolithisch / Teilsprachen	Der Grundaufbau einer Sprache hat Einfluß auf die Erweiterungsmöglichkeiten.
		Objektorientierte Konzepte	Klasse, Objekt, Vererbung, Polymorphismus, Rollen	Eine objektorientierte Modellierungssprache muß alle wesentlichen objektorientierten Konzepte unterstützen. (vgl. Abschnitt 4.3.2.1, S. 25)
		Modularisierungskonzepte	Subsysteme, Module Muster, Frameworks	Für die Entwicklung großer Systeme sind Modularisierungskonzepte erforderlich.

Tabelle 3: Bewertungsrahmen (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Sprache	Konzepte (abstrakte Syntax und Semantik)	Dynamische Konzepte	Zustandsautomaten, Interaktionsbeschreibungen, Ablaufpläne	Neben der Beschreibung der Struktur eines Systems ist auch die Beschreibung des dynamischen Verhaltens erforderlich.
		Nebenläufige Konzepte	Parallelität, verteilte Systeme, Prozesse	Nebenläufigkeit und Verteilung spielen bei umfangreichen Systemen eine immer größere Rolle.
		Prozeßorientierte Konzepte	CSP, Workflows, Prozeßmodellierung, Objekt-Prozeß-Modell	Prozeßorientierte Beschreibungen von Systemen sind ein anderes Paradigma, das in vielen Bereichen eine große Bedeutung hat (*Business Process Engineering*) und häufig zusammen mit objektorientierten Techniken eingesetzt wird.
		Weitere Konzepte	Auflistung dieser Konzepte (Petrinetze, funktionale oder logische Ansätze)	Für viele Anwendungsbereiche existieren zusätzlich spezielle Konzepte, die sich in diesem Bereich bewährt haben.
		Integration	starke/schwache Verknüpfung der verschiedenen Paradigmen und Konzepte	Die einzelnen Konzepte beschreiben Spezialansichten auf ein Modell. Für die Gesamtsicht und die Sicherung der Konsistenz ist es notwendig, die verschiedenen Konzepte miteinander zu integrieren. Ein multiparadigmatischer Ansatz ermöglicht es zudem, die Vorteile verschiedener Ansätze gemeinsam zu nutzen.
	Notation (konkrete Syntax)	Diagrammarten	Art und Anzahl der verschiedenen Diagramme	Die Diagramme sollten möglichst viele Aspekte des Modells veranschaulichen.
		Diagrammelemente	Art und Auswahl der Diagrammelemente	Die Auswahl der Diagrammelemente hat direkten Einfluß auf die Anschaulichkeit und Verständlichkeit der Sprache. Insbesondere ist eine Bewertung der Diagrammelemente aus ergonomischer Sicht notwendig.
		Annotationen	Art und Auswahl	In vielen Fällen sind Annotationen eine gute Ergänzung zu Diagrammen.
		Integration	Integration zwischen den einzelnen textuellen und graphischen Repräsentationen	Da textuelle und graphische Repräsentationen häufig gemeinsam benutzt werden, muß die Verbindung zwischen beiden geklärt sein.

Tabelle 3: Bewertungsrahmen (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Anwendung	Benutzersichten	Anwendersicht	zwischenmenschliche Kommunikation Werkzeugunterstützung	Dies ist die dominierende Sicht auf die Modellierungssprache. Die Unterstützung der Zwischenmenschlichen Kommunikation und die Beschreibung von Modellen ist das Hauptziel der Modellierungssprache. Fast alle Kriterien in diesem Bezugsrahmen beziehen sich auf die Anwendungssicht.
		Metasicht	Formen der Erweiterung und Konkretisierung des Metamodells	Gute Erweiterungsmöglichkeiten erlauben die Ableitung spezieller Modellierungssprachen auf der Basis der Ausgangssprache, aber auch die Weiterentwicklung der Ausgangssprache selbst.
	Zweck	Anwendungszweck	Sprache zur Systembeschreibung und Software-Entwicklung	Der Anwendungszweck bestimmt den Umfang einer Modellierungssprache. Er ist Grundlage für die Einschätzungen, ob eine Sprache angemessen und vollständig bezüglich des Modellierungszwecks ist.
	Entwicklungstätigkeiten	Analysetätigkeiten	spezielle Konstrukte für die Analyse	Für die Analyse sind spezielle Konzepte oder Sichtweisen erforderlich, die die besonderen Zielstellungen bei der Analyse berücksichtigen.
		Entwurfstätigkeiten	spezielle Konstrukte für den Entwurf	Für den Entwurf sind spezielle Konzepte oder Sichtweisen erforderlich, die die besonderen Zielstellungen beim Entwurf berücksichtigen. Kann dies durch Anreicherung oder direkter Überarbeitung von Analyseergebnissen erfolgen, dann ist zusätzlich auch die Integration gesichert.
		Implementierungstätigkeiten	spezielle Konstrukte zur Darstellung von Implementierungsdetails	Für die Implementierung sind spezielle Konzepte oder Sichtweisen erforderlich, die die besonderen Zielstellungen bei der Implementierung berücksichtigen. Meist erfolgt dies durch die Anreicherung der Entwürfe mit Implementierungsdetails.
		Verification and Validation (V&V)	spezielle Konstrukte für V&V, Navigationsmechanismen	Für V&V müssen spezielle Mechanismen bereitgestellt werden, die es erlauben, zwischen den einzelnen Phasenmodellen zu navigieren. Zusätzlich kann auch untersucht werden, ob die Modellierungssprache eine Validierung der Modelle unterstützt.
		Integration	Verknüpfung der Sprachkonzepte und Ziele in den einzelnen Phasen Regeln	Neben der Bereitstellung von Konzepten für die einzelnen Tätigkeiten muß auch ihre Integration gesichert werden. Die Verwendung gleicher Konzepte unter Berücksichtigung der verschiedenen Zielsetzungen in allen Phasen sichert eine gute Integration.

Tabelle 3: Bewertungsrahmen (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Kriterien (Gesamtsicht)	anwenderbezogen	Anwendbarkeit	bezüglich Konzepte und Notation	Anwendbarkeit bewertet die prinzipielle Eignung der Konzepte für den Modellierungszweck und den Benutzer.
		Anschaulichkeit Verständlichkeit	bezüglich Konzepte bezüglich Notation bezüglich Beschreibung	Anschaulichkeit ist eine wesentliche Anforderung an Modellierungssprachen. Sie ist ein Maß für die Benutzerfreundlichkeit der Sprache.
		Angemessenheit	bezüglich Modellierungszweck	Angemessenheit sichert, daß die Sprache den Anforderungen der Modellierung gerecht wird.
		Überprüfbarkeit	zwischen Modell und modellierter Realität innerhalb des Modells	Überprüfbarkeit ist eine wesentliche Anforderung an Modellierungssprachen und gleichzeitig ein wichtiger Teil von V&V.
		Mächtigkeit	Umfang und Vollständigkeit bezüglich Modellierungszweck	Die Mächtigkeit ist ein Maß, ob die Modellierungssprache den Anforderungen der Modellierung gerecht wird.
	modellbezogen	Eindeutigkeit	Eindeutigkeit der abstrakten Syntax und Semantik	Eindeutigkeit ist eine Grundvoraussetzung für das gemeinsame Verständnis eines Modells. Sie bedeutet eine Reduzierung von Mehrdeutigkeiten und Interpretationsfreiräumen.
		Konsistenz	zwischen Modell und modellierter Realität innerhalb des Modells auf der Basis der abstrakten Syntax und Semantik	Widerspruchsfreie Modelle sind ein wichtiges Ziel in der Modellierung. Eine Modellierungssprache muß den Bau widerspruchsfreier Modelle unterstützen.
		Formalisierungsgrad	informale Konzepte formale Konzepte Stufe der Formalisierung	Die Formalisierung erfolgt zur Gewährleistung der Konsistenz und Eindeutigkeit.
		Integrationsgrad	Integration der Bestandteile der Sprache Integrationsfähigkeit der Sprache selbst	Die Integration der einzelnen Teilkonzepte zu einem Ganzen ist eine wesentliche Forderung für die Konsistenz und Überprüfbarkeit.
	ökonomisch	Wiederverwendbarkeit	bezogen auf die Beschreibungsebene bezogen auf die Modelle	Wiederverwendung erlaubt die Benutzung oder Erweiterung existierender Konzepte oder Komponenten. Sie steigert damit die Effizienz und führt zu einer Vereinheitlichung.
		Erweiterbarkeit	bezogen auf die Beschreibungsebene bezogen auf die Modelle	Die Erweiterbarkeit spielt aus Metasicht eine wichtige Rolle, da sie in gewissen Umfang die Anpassung einer Modellierungssprache an geänderte Modellierungsbedingungen erlaubt.

5 Vorstellung von UML und OML

UML und OML sind zwei objektorientierte Modellierungssprachen. Sie basieren auf älteren objektorientierten Modellierungssprachen. Im Hinblick auf die derzeitig angestrebte Vereinheitlichung von objektorientierten Modellierungssprachen spielen beide eine große Rolle. Beide Sprachen sollen nun kurz vorgestellt werden. Anschließend erfolgt dann der Vergleich.

5.1 UML

5.1.1 Überblick

Die UML (Unified Modeling Language) ist das Ergebnis der Standardisierungsbemühungen von Booch, Rumbaugh und Jacobson. Bis vor kurzem war die Version 1.0 aktuell. Seit September liegt die erweiterte und überarbeitete Version 1.1[1] vor. In ihr sind verschiedene Ideen und Ansätze aus den anderen, bei der OMG eingereichten Standardisierungsvorschlägen[2], integriert. Insbesondere haben die Entwickler dieser Standardisierungsvorschläge sich der UML-Entwicklungsgruppe angeschlossen. Die auffälligsten Änderungen sind die Definition einer Object Constraint Language (OCL)[3] und eine stärkere Formalisierung und Überarbeitung des Metamodells. Das Ziel bei der Entwicklung der UML ist die Schaffung einer Sprache zur Spezifikation, Konstruktion, Dokumentation und Veranschaulichung von Software-Systemen beliebiger Komplexität aus allen Bereichen ([Rat97b], S. 6).

Während ihrer Untersuchungen für die Entwicklung einer "Unified Method" kamen Booch, Rumbaugh und Jacobson zu dem Ergebnis, daß eine Vereinheitlichung des Entwicklungsprozesses problematisch ist. Sie konzentrierten sich deshalb auf die Entwicklung einer einheitlichen Modellierungssprache. Die UML selbst ist in einem Metamodell und einem separaten Notationsteil beschrieben.

Die Dokumentation zu UML besteht aus einer großen Anzahl von Dokumenten ([Rat97a] bis [Rat97m] für die Version 1.0 und [Rat97n] bis [Rat97q] für die Version 1.1), die sowohl UML als auch das Verhältnis von UML zu verschiedenen anderen Standards und Austauschformaten beschreiben. Die wichtigsten Dokumente zum Verständnis der UML Version 1.0 sind der *Semantics Guide* [Rat97c], der *Notation Guide* [Rat97i] und das Glossar [Rat97d]. Für die Version 1.1 sind dies der *Semantics Guide* mit integriertem Glossar [Rat97q], der *Notation Guide* [Rat97p] und zusätzlich die OCL Spezifikation [Rat97o].

5.1.2 Metamodell

Die Sprachelemente der UML bieten die Möglichkeit, strukturelle, dynamische und zustandsorientierte Aspekte objektorientierter Modelle zu modellieren. Zusätzlich gibt es Erweiterungsmechanismen, um die UML an spezielle Anwendungsbereiche anzupassen. Das Metamodell der UML definiert die abstrakte Syntax und Semantik der Sprachelemente der UML ([Rat97b], S. 7). Die einzelnen Elemente der UML werden unabhängig von ihrer möglichen Repräsentation beschrieben. Für die wichtigsten Sprachmittel werden informale Definitionen ihrer Semantik angegeben. Eine getrennte Darstellung von Sprachspezifikation und Metamodell erfolgt nicht. Das Metamodell von UML ist selbst in UML modelliert. Die einzelnen Sprachkonzepte werden dabei hauptsächlich auf Klassen abgebildet. Zusätzlich enthält das Modell aber auch Klassen, die nicht direkt Sprachkonzepte der UML modellieren. Zur Modellierung des Metamodells der UML werden überwiegend die statischen klassenbezogenen Sprachkonzepte der UML verwendet. Das Metamodell der UML zeigt im wesentlichen eine Realisierung der UML. Im Metamodell werden objektorientierte Techniken verwendet, um Gemeinsamkeiten

1. Dieser Arbeitsbericht betrachtet noch die Version 1.0. Aussagen zur Version 1.1 sind speziell gekennzeichnet.
2. Es handelt sich hierbei um 5 weitere Vorschläge, die bei der OMG eingereicht wurden.
3. Vgl. hierzu auch [IBM97]. IBM und ObjectTime Limited spezifizieren in ihren Vorschlag für die OMG ebenfalls eine OCL, die nun in die UML integriert wird.

bei den einzelnen Sprachelementen zu finden und die Sprache durch einheitliche Mechanismen und Sprachmuster zu vereinfachen. Vereinheitlichende Prinzipien sind z. B. die Typ/Exemplar-Dichotomie, die Klassifikation von Sprachelementen durch gemeinsame abstrakte Klassen und die Verwendung von Stereotypen zur Erweiterung der UML. Die Typ/Exemplar-Dichotomie wird für Klassen/Objekte, für Klassenbeziehungen/Objektverweise und für Muster/Musterinstanzen verwendet. Die abstrakte Klasse *ModelElement* dient im Metamodell zur Festlegung der Semantik, die allen Modellierungselementen der UML gemeinsam ist.

Der *Semantics Guide* Version 1.0 [Rat97c] unterteilt die Elemente des Metamodells in die fünf Gruppen Grundelemente, strukturelle Elemente, Verhaltenselemente, Ansichtenelemente und vordefinierte Werte. Die Grundelemente sind überwiegend abstrakter und organisatorischer Natur. Sie umfassen die Klassen *Model, System, Package, Element, ModelElement* und *ViewElement* zum Aufbau des Grundgerüsts des Metamodells und die Klassen für die Grundmechanismen wie Bedingungen, Stereotypen, Abhängigkeiten und Beziehungen zur Verknüpfung, Einschränkung oder Erweiterung von Sprachmitteln. Drittens zählen zu den Grundelementen noch die Typen *String, Enumeration, List* und weitere, die zur Formulierung der Sprachmittel benötigt werden. Die Grundelemente sind hauptsächlich Ausdruck der Beschreibung und Realisierung der Sprache durch das Metamodell. Sie definieren aber auch die Modularisierungskonzepte *Package* und *System* der UML, die zur Organisation innerhalb der Anwendermodelle verwendet werden. Die strukturellen Elemente im UML-Metamodell beschreiben die Klassen für die Sprachkonstrukte der UML, die zur Beschreibung der strukturellen Aspekte eines Anwendermodells benötigt werden. Dies sind Typen, Klassen, Instanzen, Operationen, Attribute, Rollen, Kardinalitäten, Mechanismen und Beziehungen wie Assoziation und Vererbung. Die Verhaltenselemente dienen zum Beschreiben der dynamischen Aspekte objektorientierter Modelle. Sie sind in zwei große Gruppen unterteilt. Die erste Gruppe dient zur Modellierung von endlichen Automaten, die Erweiterungen geschachtelter Zustandsautomaten sind. Die zweite ermöglicht die Modellierung von Botschaften und anderen Interaktionsformen. Die Ansichtselemente[1] integrieren und definieren die Diagramme. Dies ist hauptsächlich eine Entscheidung der Metaebene, da die Diagramme keine abstrakten Sprachmittel, sondern Repräsentationsformen der graphischen Notation sind. Die vordefinierten Werte fassen die vordefinierten Stereotypen, Bedingungen und Name-Wert-Paare der UML zusammen.

Der *Semantics Guide* Version 1.1 [Rat97q] beschreibt das überarbeitete Modell des Metamodells. Die Unterteilung erfolgt nun in die Pakete *Foundation, Behavioral Elements* und *Model Management*. Das Paket Foundation unterteilt sich in die Pakete *Core, Auxiliary Elements, Extension Mechanisms* und *Data Types* und das Paket *Behavioral Elements* in *Common Behavior, Collaborations, Use Cases* und *Statemachines*. Trotz verschiedener Änderungen, wie Hinzufügen und Entfernen von Klassen (*Namespaces, System*) und der Umorganisation des Modells, entspricht das Metamodell der Version 1.1 dem der Version 1.0. Die auffälligste Änderung zur Version 1.0 ist die Verwendung der OCL zur Beschreibung der statischen Semantik in der Version 1.1 sowie eine bessere und übersichtlichere Strukturierung.

5.1.3 Notation

Die Notation ist separat vom Metamodell im *Notation Guide* [Rat97i] beschrieben. In ihr wird die graphische Darstellung für die einzelnen Sprachelemente festgelegt und mit Beispielen erklärt. UML definiert im wesentlichen eine graphische Notation als konkrete Syntax. Zusätzlich erfolgt eine kurze semantische Beschreibung des Sprachmittels aus Anwendersicht. Sie kann als Kurzbeschreibung der abstrakten Semantik der Sprachkonzepte aufgefaßt werden. Diese anwenderorientierte Beschreibung ermöglicht eine bessere Zuordnung der Sprachmittel zu den einzelnen Teilmodellen der UML. Ähnlich wie bei der Festlegung der Sprachmittel wird versucht, gleiche Strukturen der Sprachmittel durch entsprechende graphische Notationen auszudrücken. Dadurch wird die Notation einfacher und leichter

1. In Version 1.1 sind die Klassen für die Diagramme nicht mehr im Metamodell enthalten.

anwendbar. Die Notation benutzt als Grundlage die Symbolik aus der Booch-Notation, der OMT und OOSE. Zwischen Version 1.0 und Version 1.1 gibt es keine wesentlichen Differenzen. Die Notation von Version 1.1 entspricht der der Version von 1.0.

Die Begriffe Diagramm, Diagrammelement und die Bedeutung topologischer Beziehungen zwischen Diagrammelementen wie Enthaltensein oder Zusammengehörigkeit werden im Notation Guide erklärt. UML stellt jedoch keine tabellenartigen textuellen Annotationen wie die *Class Templates* der Booch-Notation für die einzelnen Modellierungselemente bereit.

5.1.4 Diagrammarten

Im *Notation Guide* werden die Diagrammarten der UML vorgestellt. Die Diagramme bieten verschiedene Blickwinkel auf ein Modell. Die Anzahl verschiedener Diagrammtypen sollte in einer graphischen Notation klein sein. Jedes Diagramm sollte das Modell dabei aus einer speziellen Perspektive darstellen. Es sollte nicht mehrere verschiedene Diagrammtypen für dieselbe Aufgabe geben.

UML definiert neun Diagrammarten zur Veranschaulichung des Modells ([Rat97c], S. 93). Diese Diagramme können strukturelle, dynamische und physikalische Teilaspekte des Modells darstellen. In Tabelle 4 sind die einzelnen Diagramme der UML mit einer kurzen Beschreibung aufgelistet.

Tabelle 4: Diagramme der UML

	Diagramm[a]	Aufgabe/Beschreibung	Anwendung
Static Structure	Class Diagram	Beschreibt Klassen, Typen, aber auch Packages, Objekte und die Beziehungen zwischen diesen.[b]	In allen Phasen der Software-Entwicklung zur Beschreibung der logisch statischen Struktur der Software.
	Object Diagram[c]	Beschreibt die statische und dynamische Struktur von Objekten.	In allen Phasen der Software-Entwicklung zum Beschreiben der Objektstruktur.

Tabelle 4: Diagramme der UML (Fortsetzung)

	Diagramm[a]	Aufgabe/Beschreibung	Anwendung
Behavior	Use Case Diagram	Zeigt die Beziehungen zwischen Benutzern und Use-Cases in einem System.	Anforderungsanalyse, Systemmodellierung.
	Interaction Diagram	Beschreiben Mechanismen (Interaktionen) zwischen Objekten.	In allen Phasen der Software-Entwicklung zur Beschreibung von Interaktion und Kommunikation.
	Sequence Diagram	Veranschaulicht den zeitlichen Verlauf der Interaktion (Botschaftenfluß) unter Vernachlässigung der Beziehungen zwischen den Objekten.	
	Collaboration Diagram[d]	Veranschaulicht die strukturellen Beziehungen zwischen den Objekten und dient zum Beschreiben von Entwurfsmustern. Es zeigt nicht den zeitlichen Verlauf der Interaktion.	
	State Diagram	Beschreibt den endlichen Automaten für ein Objekt (Klasse) oder ein Teilsystem. Es basiert auf den Zustandsdiagrammen von D. Harel.	System-, Klassenentwurf und Implementierung. Zur Beschreibung von Zustandsänderungen.
	Activity Diagram	Ist ein spezielles Zustandsdiagramm, welches eine Operation (Algorithmus) beschreibt. Es ist vergleichbar mit Ablaufplänen und kann auch zur Beschreibung von Geschäftsprozessen und Work-Flows verwendet werden.	Analyse, Entwurf und Implementierung. Zur Beschreibung von Abläufen.
Implementation	Component Diagram	Zeigt die Organisation und Abhängigkeiten wie Compiler-, Schnittstellen oder Aufrufabhängigkeiten zwischen Systemkomponenten.	In der Implementierung zum Aufzeigen von Programmcodeabhängigkeiten.
	Deployment Diagram	Beschreibt die Konfiguration des Laufzeitsystems und die physikalische Verteilung.	Implementierung und Systemkonfiguration des Laufzeitsystems.

a. Für die Diagramme werden die englischen Originalbezeichnungen verwendet.
b. Der Bezeichner *Class Diagram* ist für Diagramme mit dieser Verwendung verwirrend. UML gibt selbst zu, das *Static Structural Diagram* ein besserer Bezeichner wäre ([Rat97i], S. 18).
c. vgl. *Collaboration Diagram*. Beide Diagrammarten haben die gleiche Aufgabe.
d. vgl. *Object Diagram*. Beide Diagrammarten haben die gleiche Aufgabe.

5.1.5 Erweiterungsmechanismen

Die UML enthält drei Mechanismen zur Erweiterung der Modellierungssprache ([Rat97c], S. 15 ff.). Diese sind Stereotypen, Name-Wert-Paare (*tagged values*) und Bedingungen. Die Erweiterungsmechanismen sind wichtig, um die UML an neue Anforderungen anpassen zu können.

Stereotypen erlauben die Ableitung eines neuen Sprachelements auf der Basis eines schon existierenden. Stereotypen können dabei zur Definition anwendungsspezifischer Sprachelemente auf der Grundlage existierender Sprachelemente[1] der UML benutzt werden. UML selbst benutzt Stereotypen zur

1. Stereotypen können nur von Typen und Klassen des Metamodells abgeleitet werden.

Definition einfacher Erweiterung von Modellierungsmitteln. <<Interface>> und <<Powertype>> sind z.B. vordefinierte Stereotypen für das Sprachelement Type. Stereotypen bieten damit ein Klassifikationsschema zur Unterscheidung von Konzepten mit ähnlicher Semantik. Die Struktur des Elements, von dem der Stereotype abgeleitet wird, kann nicht geändert werden. Prinzipiell könnten durch Stereotypen eingeführte Elemente auch durch Vererbungsbeziehungen in das Metamodell integriert werden. Name-Wert-Paare erlauben es, zu einer Klasse aus dem UML Metamodell zusätzliche Attribute hinzuzufügen. Dies erlaubt das dynamische Hinzufügen von Attributen zu Klassen[1]. Der Name und der Typ der neuen Variable werden durch das Name-Wert-Paar festgelegt. Bedingungen ermöglichen die Änderung der Semantik existierender Sprachelemente. Name-Wert-Paare ändern die Struktur eines Sprachelements. Bedingung ist das Analogon zu Name-Wert-Paar bezüglich des definierten Verhaltens der Sprachelemente.

Diese Erweiterungsmechanismen erlauben keine Hinzunahme vollständig neuer Sprachkonzepte. Sie erlauben es jedoch, neue Sprachmittel von existierenden abzuleiten und die existierenden strukturell und semantisch zu erweitern. Die Erweiterungsmechanismen sind Metamodellierungskonzepte.

5.1.6 Allgemeine Bemerkungen

UML ist eine sehr umfangreiche und komplexe objektorientierte Modellierungssprache. Sie ist ein wichtiger Sprachvorschlag, der bekannte und neue Modellierungskonzepte verbindet und ein großes Standardisierungspotential hat. Sie hat jedoch auch Schwächen, die ihre Anwendbarkeit erschweren. Die Erweiterungsmechanismen erlauben z.B. eine leichte Änderung der Sprache, bergen aber gleichzeitig auch die Gefahr des Mißbrauchs in sich.

UML enthält Sprachmittel zur Modellierung verteilter und nebenläufiger Systeme, zum Ausdrücken von Entwurfsmustern und zur Verfeinerung von Modellierungsentscheidungen in den verschiedenen Phasen der Software-Entwicklung.

Für den Anwender ist der *Notation Guide* [Rat97i] informativer. Er beschreibt neben der Notation auch die Bedeutung der wichtigsten Sprachelemente aus Anwendersicht. Dies kann aber eine Sprachbeschreibung nicht ersetzen, in der die einzelnen Sprachelemente und ihre Semantik unabhängig von Metamodellierungsrestriktionen definiert sind. Ob die Klasse *StateMachine* eine Unterklasse von *Behavior* ist, ist für den Anwender relativ unbedeutend, solange eine exakte Definition von Zustandsautomaten existiert, die bei der Anwendung dieses Konstrukts hilft. Der *Semantics Guide* [Rat97c] beschreibt das Metamodell von UML, ist aber keine Sprachbeschreibung von UML. Zwar können aus dem Metamodell Aussagen über die UML abgeleitet werden, die Vermischung von Meta- und Sprachebene verhindert aber die eindeutige Darstellung der Sprachelemente. Da zudem keine Hinweise gegeben werden, warum das Metamodell auf diese Weise und nicht anders modelliert wurde, sind die Modelle im allgemeinen schwer nachvollziehbar. Die gleichzeitige Verwendung objektorientierter Begriffe auf Meta-und Sprachebene erschweren das Verständnis zusätzlich.

5.2 OML

5.2.1 Überblick

OML (Open Modelling Language) ist ein Teil von OPEN (Object-oriented Process, Environment, and Notation) [FiHe96]. OPEN ist der Versuch, für die objektorientierte Software- und Systementwicklung ein einheitliches Vorgehensmodell zu entwickeln, welches zu einer Standardisierung von Vorgehensweise und Notation führt. OPEN ist aus dem COMMA-Projekt (Common object meta modelling architecture) hervorgegangen, in dem Henderson-Sellers verschiedene objektorientierte Notationen und

1. Dieser Ansatz ist vergleichbar mit der Implementierung von Klassen mit dynamischem Zustand, bei der die Variablen in einem Verzeichnis verwaltet werden.

Vorgehensweisen untersucht hat. In [HeEd94] erfolgt eine umfassende Analyse der Anwendungssituation objektorientierter Techniken auf der Basis des COMMA-Projekts. Auch Graham und Firesmith, zwei weitere Autoren der OML, besitzen langjährige Erfahrungen mit objektorientierten Techniken ([Fir92] und [Gra91]). Das Grundgerüst von OPEN ist in Abb. 4 gezeigt.

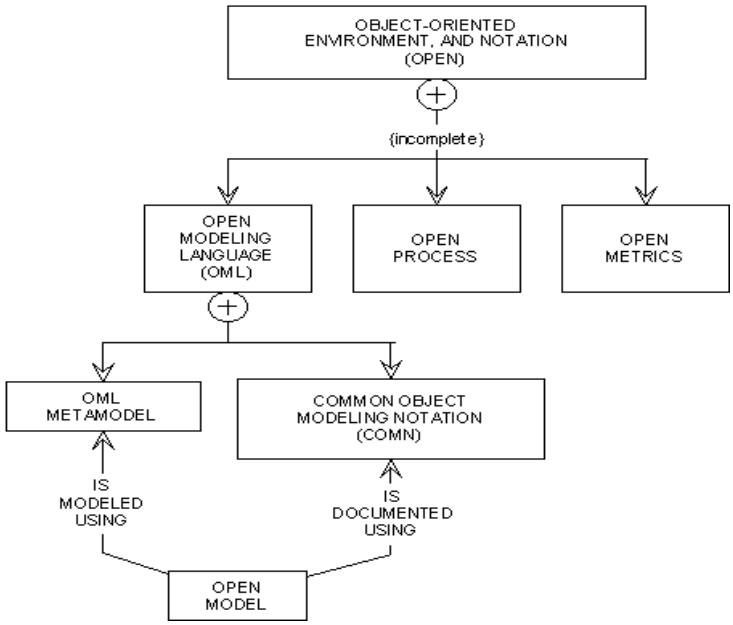

Abb. 4: OPEN ([FiHe96], S. 5)

OML selbst besteht aus COMN (Common Object Modelling Notation) und einem Metamodell, das die einzelnen Sprachmittel und ihre Beziehungen beschreibt. UML und OML benutzen dieselbe Grundstruktur zur Sprachbeschreibung, die durch ein Metamodell sowie die Trennung von Sprachbeschreibung und graphischer Darstellung charakterisiert ist. OML ist in einem Dokument beschrieben, in dem die einzelnen Sprachmittel mit ihrer Bedeutung und Darstellung eingeführt werden.

5.2.2 Metamodell

Auch OML benutzt ein Metamodell zur Festlegung der Sprachkonstrukte. Die Sprachkonstrukte sind nach ihrer graphischen Repräsentation in Knoten und Beziehungen untergliedert. Zusätzliche Elemente wie Kardinalitäten oder Namenskonventionen werden separat betrachtet. OML versucht ähnlich wie UML, die Sprachmittel durch Standardmechanismen zu vereinfachen. So wird die Typ/Exemplar-Dichotomie für Klassen/Objekte, *Cluster*/*Cluster*-Instanzen und Beziehungen angewendet. Andere generelle Mechanismen sind das Rollenkonzepts oder die allgemeine Verwendung von *Drop-Down*-Boxen zur Annotierung. Wie bei UML modelliert OML das Metamodell mit Hilfe von OML. Bei der graphischen Darstellung der Metamodelldiagramme werden für noch nicht bekannte OML-Symbole Ersatzsymbole verwendet. Die OML-Symbole werden schrittweise eingeführt. Ersatzsymbole sind zum Beispiel normale Linien und Rechtecke. Diese Vermischung von COMN und Ersatzsymbolen ist vorteilhaft für die Vorstellung der Konstrukte. Sie erschwert aber durch die uneinheitliche Notation das Untersuchen des Metamodells in seiner Gesamtheit.

Die Knoten unterteilen sich in sechs Gruppen, wobei *Drop-Down*-Boxen nur Repräsentationsknoten sind. Die Zustandsknoten werden in einem separaten Kapitel über Zustandsdiagramme vorgestellt. Die weiteren Knotengruppen sind Objektknoten, *Cluster*-Knoten, *Szenario*-Knoten und nicht objektorientierte Knoten. Für Objektknoten und *Cluster*-Knoten werden die Begriffe *Class*, *Instance*, *Role*, *Type*, *Class implementation* und *Characteristic* definiert. Man spricht dann beispielsweise von *Object Class* beziehungsweise *Cluster Class*. Zusätzlich können Objekte in interne, externe, persistente, *Actor*-

Objekte und andere klassifiziert werden. Die Objektknoten definieren die Sprachmittel, die zur strukturellen Beschreibung objektorientierter Systeme notwendig sind. *Cluster*-Knoten dienen zum Beschreiben von Teilsystemen und von Mengen zusammengehörender Komponenten. *Cluster* ist in OML der Sammelbegriff für Modul, Layer, Muster, Einheit oder Teilsystem. UML verwendet hierfür den Begriff *Package*. *Cluster* können bezüglich differenzierter Eigenschaften unterschieden werden. Wie schon erwähnt gibt es auch für *Cluster* Klassen, Typen, Implementierungen, Rollen und Eigenschaften. Aus dem Blickwinkel der OML werden *Cluster* ähnlich wie Objekte behandelt. Szenarien beschreiben Interaktionen. Auch für Szenarien gibt es Klassen, Typen und Implementierungen. Die nicht-objektorientierten Knoten beschreiben globale Daten, Operationen und andere nicht-objektorientierter Konzepte. Dies ist hilfreich bei der Modellierung von Systemen mit alten nicht-objektorientierten Teilsystemen.

Die Beziehungen sind ähnlich wie die Knoten organisiert. Sie sind in definierende und referenzielle Beziehungen unterteilt. Definierende Beziehungen geben an, daß ein Element auf die Definition eines anderen Elements aufbaut. Die wichtigsten definierenden Beziehungen sind Klassifizierungs-, Implementierungs- und Vererbungsbeziehungen. Diese Beziehungen sind immer binär und unidirektional. OML unterstützt keine n-nären Beziehungen. Referenzielle Beziehungen oder Verweise veranschaulichen die Sichtbarkeit eines Elementes auf ein anderes. Auch diese Beziehungen sind binär. Typische referenzielle Beziehungen in OML sind Verbindung, Assoziation, Aggregation und *Containment* (Enthaltensein-Beziehung). Im Gegensatz zu definierenden Beziehungen können Verweise auch bidirektional sein. Dies gilt aber nicht für alle Verweise. Zusätzlich gibt es noch Szenario-Beziehungen und Transformationsbeziehungen.

OML stellt ähnlich wie UML umfangreiche Sprachmittel für die Modellierung zur Verfügung. Sie unterstützen den Systementwurf, die objektorientierte Modellierung und die Modellierung der strukturellen und dynamischen Aspekte der Software.

5.2.3 Notation

Die Notation für die einzelnen Sprachelemente wird gleichzeitig mit den entsprechenden Sprachmitteln eingeführt. Es gibt keine separate Notationsbeschreibung. Die verwendete Notation orientiert sich an der Symbolik aus den Notationen von Firesmith, Odell, und Henderson-Sellers ([Fir92], [HeEd94], [Hen92] und [MaOd92]). Die Symbole sind so gewählt, daß es wenig Gemeinsamkeiten mit strukturellen Notationen gibt. So benutzt OML z.B. für Klassen keine Rechtecke, weil Entity-Relationship-Diagramme Rechtecke zur Darstellung von *Entities* benutzen. Die Verwendung von Rechtecken für Klassen könnte nach Henderson-Sellers dazu führen, daß datenorientiert und nicht objektorientiert modelliert wird und Klassendiagramme als spezielle Entity-Relationship-Diagramme aufgefaßt werden. Ähnlich wie UML benutzt auch die OML einheitliche Grundprinzipien bei der Festlegung einzelner Notationselemente. Dies gewährleistet eine einheitliche Darstellung ähnlicher Komponenten.

Neben Diagrammen stellt OML auch textuelle Spezifikationen für einzelne Modellierungselemente bereit. Dies sind z. B. die bekannten CRC-Cards[1] für Klassen, Objekte und Rollen.

5.2.4 Diagramme

OML unterstützt verschiedene Diagrammarten, die die statische und dynamische Modellierung unterstützen. Die Diagramme[2] sind in vier große Gruppen unterteilt; semantische Netze, Szenario-Klassen-Diagramme, Interaktionsdiagramme und Zustandsdiagramme.

Semantische Netze beschreiben statische Zusammenhänge. Die OML unterscheidet 6 verschiedene Arten von semantischen Netzen. Dies sind *Context Diagrams*, *Layer Diagrams*, *Configuration Dia-*

1. CRC-Cards sind Class-Responsibility-Collaborator-Cards. Vgl auch [CuBe89] und [WiWi90].
2. Für die einzelnen Diagrammarten werden die englischen Originalbezeichnungen verwendet

grams, *Cluster Diagrams*, *Inheritance Diagrams* und *Deployment Diagrams*, wobei sich die *Context Diagrams* noch einmal untergliedern in *Software* und *System Context Diagrams*. Szenario-Klassen-Diagramme beschreiben Szenarien und andere Zusammenhänge auf Klassen- und Objektebene. Sie beschreiben funktionale Zusammenhänge und basieren auf funktionaler Dekomposition. Es gibt *Mechanism Diagrams*, *Task Script Diagrams* und *Use Case Diagrams*. Zustandsdiagramme sind die graphische Repräsentation endlicher Automaten. Endliche Automaten werden zur Beschreibung von Objekten (Klassen) und *Cluster* angewendet, um deren Zustandsverhalten zu charakterisieren. Interaktionsdiagramme beschreiben Botschaftenflüsse, Mechanismen und Abläufe. Sie untergliedern sich in *Collaboration Diagrams* und *Sequence Diagrams*.

Insgesamt umfaßt die OML damit über 17 verschiedene Diagrammarten. Diese große Anzahl und die stellenweise überdeckenden Aufgaben erschweren die richtige Auswahl der Diagramme. Die speziellen Diagramme erschweren weiterhin die Integration verschiedener zusammengehöriger Teilaspekte in einem Diagramm. Vererbungsbeziehungen und normale Beziehungen für Klassen sollen zum Beispiel laut OML nicht gemeinsam, sondern in separaten Diagrammen dargestellt werden.

In Tabelle 5 sind die einzelnen Diagramme der OML mit einer kurzen Beschreibung aufgelistet.

Tabelle 5: Diagramme der OML

	Diagramm	Aufgabe/Beschreibung	Anwendung
Semantic Nets	Context Diagram[a]	Beschreibt die Wechselwirkungen zwischen einem System und seiner Umgebung aus externer Sicht. Es dient zum Abgrenzen des Systems.	Analyse des Problembereichs, System- und Grobentwurf und Systemmodellierung
	System Context Diagram	Betrachtet Systeme, bestehend aus Software, Hardware, Personen, etc.	Systementwurf, Anforderungs- und Systemanalyse
	Software Context Diagram	Betrachtet Software-Komponenten.	Systementwurf, Anforderungs- und Systemanalyse
	Layer Diagram	Beschreibt die Software-Architektur einer Anwendung. Dies umfaßt die Zerlegung in einzelne Schichten und die Zuordnung der Software auf die einzelne Hardware. Externe Sicht auf *Cluster*.	Systemanalyse, System- und Grobentwurf
	Configuration Diagram	Beschreibt die Software-Architektur einer Anwendung mit Hilfe von *Clustern* und *Cluster*-Instanzen[b].	Systemanalyse, System- und Grobentwurf
	Cluster Diagram	Beschreibt die interne statische Struktur innerhalb eines *Clusters*.	*Cluster*-Entwurf und Implementierung
	Inheritance Diagram	Beschreibt die Vererbungsbeziehungen, aber auch andere Beziehungen zwischen Klassen[c].	Analyse, Entwurf und Implementierung
	Deployment Diagram	Beschreibt die Hardware und die Verteilung der Software auf die Hardware.	Systementwurf, Realisierung

Tabelle 5: Diagramme der OML (Fortsetzung)

	Diagramm	Aufgabe/Beschreibung	Anwendung
Scenario Class Diagrams	Mechanism Diagram	Beschreibt Mechanismen zwischen Objekten sowohl auf Klassen- als auch auf Objektebene.	Analyse, Entwurf und Implementierung
	Task Script Diagram	Beschreibt *Task Scripts*.	Analyse und Entwurf
	Use Case Diagram	Beschreibt *Use Cases*.	Analyse und Entwurf
Interaction Diagrams	Collaboration Diagramm	Dokumentiert das Zusammenwirken von Klassen oder Objekten. Es erlaubt das Darstellen des dynamischen Verhaltens durch Botschaften und Ausnahmen. (*Message Flows*).	Analyse, Entwurf und Implementierung Modellierung des dynamischen Verhaltens
	Cluster Collaboration Diagram	Dokumentiert das dynamische Verhalten von Klassen und Objekten aus externer Sicht.	Analyse, Entwurf
	Scenario Collaboration Diagram	Dokumentiert das dynamische Verhalten der Modellierungselemente, die in einem Szenario teilnehmen.	Feinentwurf und Implementierung
	Internal Collaboration Diagram	Dokumentiert das dynamische Verhalten innerhalb einer Klasse oder eines Objekts.	Klassenentwurf, Feinentwurf und Implementierung
	Sequence Diagram	Veranschaulicht den zeitlichen Verlauf der Interaktion (Botschaftenfluß) unter Vernachlässigung der Beziehungen zwischen den Objekten.	Analyse, Entwurf, und Implementierung Beschreibt Anforderungen und dient als Grundlage für Testfälle.
	Blackbox Sequence Diagram	Dokumentiert den zeitlichen Verlauf der Interaktion einer Komponente mit der Systemumgebung.	
	Whitebox Sequence Diagram	Dokumentiert den zeitlichen Botschaftenfluß innerhalb eines Mechanismus, Muster oder *Use Case*.	
State Diagrams	State Transition Diagram	Beschreibt das Zustandsverhalten von Objekten als endliche Automaten.	Klassenentwurf

a. Vgl. hierzu auch Context Diagrams bei Datenflußdiagrammen.

b. *Layer Diagrams* beschreiben laut OML *Cluster*; *Configuration Diagrams* beschreiben eine konkrete Instanzierung. Vgl. dazu auch die Typ/Exemplar-Dichotomie.

c. Dies umfaßt Klassen, Typen und Klassenimplementierungen.

5.2.5 Erweiterungsmechanismen

Im Gegensatz zur UML sind in der OML keine expliziten Erweiterungsmechanismen vorgesehen. Zwar werden auch Stereotypen und Constraints benutzt, diese dienen aber mehr als Zusatzinformation. Insbesondere werden auf ihrer Basis keine neuen Modellierungselemente definiert. Ihre Bedeutung liegt auf der Modellierungsebene und weniger auf der Metaebene. So ist ein Stereotype in OML "any character string of enumeration type that is used to conceptually classify a modeling element independently from inheritance or provide some other important information about the modeling element" ([FiHe96], S. 229).

5.2.6 Abschließende Bemerkungen

Die OML ist eine sehr umfangreiche Sprache. Sie unterstützt wie UML insbesondere die Analyse- und Entwurfstätigkeiten der Software-Entwicklung und bietet Möglichkeiten, nicht-objektorientierte Bestandteile auszudrücken. Neben den objektorientierten Sprachelementen umfaßt sie ein reiches Repertoire für die Systemmodellierung. Die Beschreibung der OML ist anwenderorientiert. Das Metamodell wird zum Aufzeigen der Sprachelemente verwendet. Es erfolgt aber keine explizite Darstellung des Metamodells.

OML wurde aus organisatorischen Gründen nicht als Standardisierungvorschlag bei der OMG eingereicht, da einreichende Organisationen "contributing member" der OMG sein müssen[1]. Trotzdem ist auch OML ein äußerst interessanter Sprachvorschlag, der Aufmerksamkeit verdient und mit einem definierten Entwicklungsprozeß verknüpft ist.

6 Vergleich von UML und OML

Der Vergleich setzt sich aus zwei unterschiedlichen Teilen zusammen. Im ersten Teil erfolgt eine direkte Gegenüberstellung und Untersuchung beider Sprachen. Dabei werden spezielle Spracheigenschaften und ihre Umsetzung untersucht. Im zweiten Teil wird für beide Sprachen der Bewertungsrahmen aus Abschnitt 4.5, S. 41 aufgestellt.

6.1 Direkter Vergleich

In diesem Direktvergleich erfolgt eine Analyse beider Sprachen. Dabei werden ihre Gemeinsamkeiten und Unterschiede herausgearbeitet. UML und OML adressieren den gleichen Anwendungsbereich. Beide sind Modellierungssprachen für die objektorientierte Modellierung von Software-Systemen. Damit beziehen sie sich auf die gleiche abstrakte Syntax und Semantik. Beide basieren auf bekannten älteren objektorientierten Modellierungssprachen, wobei die Wurzeln jedoch verschieden sind. Sie sind bezüglich Umfang und Art der bereitgestellten Sprachkonstrukte ungefähr gleichmächtig. So definieren beide Sprachen eine Reihe ähnlicher Diagramme und unterstützen die dynamische und statische Betrachtung von Modellen. Sie haben jedoch auch auffällige Unterschiede. Dies betrifft insbesondere die Festlegung einzelner Sprachelemente. Trotz der gleichen abstrakten semantischen Grundlage, die durch die objektorientierten Konzepte gebildet wird, unterscheiden sie sich schon in der Festlegung der objektorientierten Sprachelemente. Auch die Modellierungskonzeption beider Sprachen ist unterschiedlich. OML mißt dem Rollenkonzept zum Beispiel eine wesentliche größere Bedeutung zu als UML. UML dagegen betont die Modellierung der strukturellen Aspekte von Objekten stärker. Beide Modellierungssprachen definieren jeweils ihre eigene Notation. Auch diese sind sehr verschieden. So ist die Auswahl der Diagrammsymbole auf beiden Seiten unterschiedlich und vielfach eine Folge der Übernahme bekannter Symbole älterer Notationen.

1. Vgl. dazu [Fra97] und [OMG96].

Die Entwickler von UML und OML betonen die Eignung beider Modellierungssprachen für alle Tätigkeiten der Software- und Systementwicklung. In diesem Zusammenhang muß erneut gefragt werden, ob es nicht Tätigkeiten während der Software-Entwicklung gibt, bei denen es vorteilhafter ist, von den Konzepten der Objektorientierung zu abstrahieren. Es gibt zusätzlich eine Vielzahl von Anwendungen, wo neben dem objektorientierten Paradigma weitere Paradigmen angewendet werden.

6.1.1 Abstrakte Syntax und Semantik

UML und OML unterscheiden sich hinsichtlich ihrer abstrakten Syntax und Semantik nur unwesentlich, da sie durch weitgehend übereinstimmende Konzepte geprägt sind. Beide Sprachen zählen zur Gruppe der objektorientierten Modellierungssprachen. Die gemeinsame Grundlage beider Sprachen bedingt gemeinsame Eigenschaften. Jedoch definieren und interpretieren beide Sprachen die abstrakten Konzepte auf verschiedene Weise, so daß schon bei den grundlegenden Sprachkonstrukten auseinandergehende Vereinbarungen festzustellen sind. Beide sind sehr umfangreiche Modellierungssprachen und enthalten viele Sprachkonzepte, die aus anderen Ansätzen übernommen wurden, wobei allerdings keine Integration aller dieser Sprachkonzepte zu einem einheitlichen Gesamtkonzept erfolgt.

Die abstrakte Syntax und Semantik bilden den Kern einer Modellierungssprache. Auf ihrer Basis kann eine verbindliche Interpretation der Modelle gesichert werden. Dies erfordert jedoch eine genaue Definition. Anstatt für die einzelnen Sprachmittel eine feste Semantik festzulegen, erlaubt UML jedoch benutzerdefinierte Interpretationen: "There are different possible ways to interpret the semantics of generalization (as with other constructs). Although there is a standard UML interpretation consistent with the operation of the major object-oriented languages, there are purposes and languages that require a different interpretation. Different semantics can be permitted by identifying semantic variation points and giving them names, so that different users and tools could understand the variation being used (it is not assumed that all tools will support this concepts)" ([Rat97i], S. 53). Diese Aussage kann nicht akzeptiert werden. Hinter jedem Sprachkonstrukt steht eine gewisse Bedeutung. Es kann daher verlangt werden, daß diese Bedeutung exakt angegeben wird und Alternativen kenntlich gemacht werden. Auch OML enthält keine exakten Definitionen für die abstrakten Sprachelemente. UML und OML konzentrieren sich stärker auf die abstrakte Syntax. Dies ist jedoch problematisch, weil die Semantik eines Sprachelements direkten Einfluß auf die möglichen Verknüpfungen mit anderen Sprachmitteln (Syntax) hat. Eine Änderung in der Semantik kann daher auch eine Änderung in der Syntax bewirken. Erlaubt UML beispielsweise die Erweiterung der Semantik des Sprachkonstrukts *generalization (inheritance)* um selektives Erben, wie es Eiffel unterstützt, dann muß gesichert werden, daß auf diese Form der Vererbung die polymorphe Typkompatibilität[1] nicht zutrifft.

UML und OML beziehen sich auf das klassenorientierte objektorientierte Programmierparadigma. Klassenlose Entwicklungsansätze und klassenlose Sprachen wie Self werden nicht direkt unterstützt. Allerdings erlaubt OML durch das Rollenkonzept, daß Instanzen Klassenrollen übernehmen können. Abb. 5 zeigt dies am Beispiel von Vererbung in OML.

1. Hier wird davon ausgegangen, daß Vererbung gleichzeitig auch *Subtyping* bedeutet, wie es in vielen objektorientierten Programmiersprachen der Fall ist.

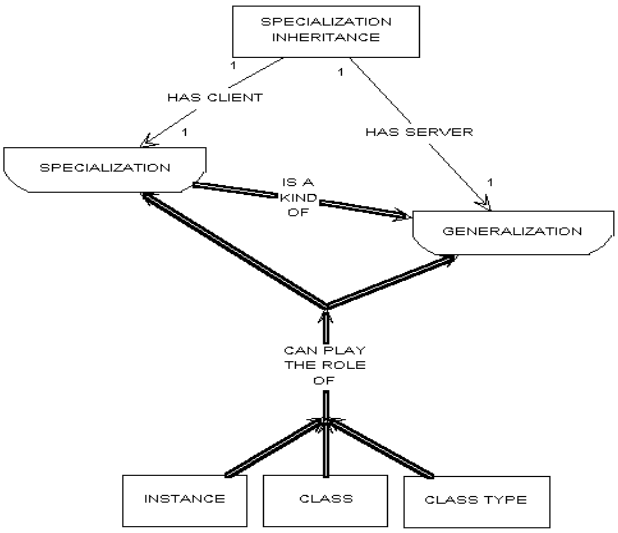

Abb. 5: Rollenkonzept für Vererbung in OML ([FiHe96], S. 85)

UML und OML legen die abstrakte Syntax und Semantik ihrer Sprachmittel nicht exakt fest. Sie weisen hier die gleichen Schwächen auf wie ältere objektorientierte Modellierungssprachen. Eine Abhilfe kann durch eine stärkere Verbindung mit formalen Ansätzen erreicht werden. Diesen Weg geht UML in der Version 1.1 mit OCL. Allerdings wird mit OCL derzeit nur die statische Semantik formalisiert, die Ausdruck der abstrakten Syntax ist. Die dynamische Semantik wird dagegen natürlichsprachlich formuliert.

6.1.1.1 Konzepte

Bei UML und OML werden eine Reihe allgemeiner Konzepte angewendet, die zu einer Vereinheitlichung und Vereinfachung der Sprache führen.

6.1.1.1.1 Typ/Exemplar-Dichotomie

OML benutzt die Typ/Exemplar-Dichotomie für *Class/Object*, *Cluster class/Cluster instance*, *Scenario class/Szenario instance* und *Association/Link*. Und auch in UML ist die Typ/Exemplar Dichotomie ein Grundmechanismus: "The Separation of Type and Instance constitutes the essence/manifestation dichotomy of the UML" ([Rat97c], S. 29). Es ist vorteilhaft, Typisierung nicht nur im Zusammenhang mit abstrakten Datentypen zu benutzen, sondern auch für Systeme, Funktionen und Beziehungen. Gerade Systeme und Objekte sind nach ihrer Charakteristik gleichartige Komponenten. In Analogie zu Klassen können daher auch Systemtypen eingeführt werden. Dies ermöglicht es, für die oben genannten Komponenten, charakteristische Eigenschaften festzulegen und ähnliche Objekte zusammenzufassen. Ein weiterer Vorteil ist, daß Typbeziehungen wie Vererbung nun auch für diese Komponenten anwendbar sind. Da Typen und Instanzen für Systeme oder Szenarien jedoch keine Entsprechung auf der Programmiersprachenebene haben, muß spezifiziert werden, wie diese Konstrukte in den einzelnen Phasen benutzt werden. Hierzu fehlen sowohl in OML als auch in UML konkrete Vereinbarungen.

6.1.1.1.2 Typ

Durch die erweiterte Anwendung der Typ/Exemplar-Dichotomie erhalten die Begriffe Typ und Exemplar eine neue Dimension. UML und OML verwenden den Begriff Typ auch für Systeme, Szenarien und Beziehungen. Für alle diese Typbegriffe ist charakteristisch, daß sowohl strukturelle als auch dynamische Eigenschaften beschrieben werden müssen. Typen erlauben die Gruppierung nach speziellen Kriterien (Eigenschaften). Die Festlegung des Typbegriffs für Systeme, Szenarien oder Beziehungen kann daher auf der Basis des klassischen Typbegriffs (abstrakter Datentyp) erfolgen. Weder UML noch OML definieren den Typbegriff und seine Erweiterungen vollständig. Insbesondere wird nicht

explizit zwischen den einzelnen Ausprägungen unterschieden. So beziehen sich Aussagen einmal auf alle Typen gemeinsam und dann wieder nur auf den Zusammenhang zwischen Klasse und Typ.

In OML ist ein Typ "any declaration of visible characteristics that form all or part of the interface of a single kind of instance that conforms to the type" ([FiHe96], S. 230). Und in UML heißt es: "A type is a description of a set of instances that share the same operations, abstract attributes and relationships, and semantics. A type may define an operation specification but not an operation implementation" ([Rat97d], S. 15); bzw.: "A type is the specification of a domain together with behavior applicable to that domain" ([Rat97c], S. 25). Beide Modellierungssprachen machen damit keine Aussage, ob ein Exemplar gleichzeitig mehrere Typen haben darf und ob sich seine Typisierung zeitlich ändern darf[1]. Die Definition in UML umfaßt nicht nur die Schnittstelle (Signatur), sondern die Spezifikation (Signatur und Semantik). Allerdings wird dies durch "Type instances specify interfaces" ([Rat97c], S. 57) wieder in Frage gestellt. Typen beschreiben in UML und OML damit im wesentlichen externe Schnittstellen.

Ein abstrakter Datentyp ist ein Analogon zu einer mathematischen Theorie oder Algebra mit einer ausgezeichneten Trägermenge, die durch ein Quadrupel $ADT = [S, \Omega, X, A]$ beschrieben ist. Dabei ist $[S, \Omega]$ die Signatur Σ des abstrakten Datentyps, bestehend aus S, der Menge der Sorten, und Ω, der Menge von Operationen. *X* ist ein System von Variablen und *A* eine Menge von Axiomen. Ein abstrakter Datentyp beschreibt eine Menge konkreter Datentypen mit der gleichen Signatur Σ, die die Axiome erfüllen. Ein konkreter Datentyp ist damit ein Modell oder eine Realisierung des abstrakten Datentyps [OtWi90]. Der abstrakte Datentyp beschreibt das *Black-Box*-Verhalten des Typs. Meyer definiert Klassen als Modelle abstrakter Datentypen, wobei auf die Bedeutung der Axiome zur Beschreibung der Semantik hingewiesen wird [Mey97].

Während lange Zeit die Begriffe Klasse und Typ synonym verwendet wurden, trennen UML und OML beide Begriffe jetzt. Allerdings fehlt eine einheitliche und konsistente Abgrenzung beider Begriffe. In UML ist eine Klasse die Realisierung eines Typs ([Rat97c], S. 24). Auf Metaebene stellt sich der Unterschied folgendermaßen dar: "Class is a subtype of Type, and therefore instances of Class have the same property as instances of Type. The fundamental difference being that Type instances specify interfaces, whereas Class instances specify the realization of these interfaces" ([Rat97c], S. 57). Und im *Notation Guide* heißt es: "UML uses the word type in a more general way, roughly corresponding to the concept of abstract data type in computer science. In most programming languages the word type corresponds most closely to the UML concept of class, as the programming-language types include both data structure and operation structure" ([Rat97i], S. 15). Diese Definition entspricht zwar der gebräuchlichen Verwendung des Begriffs Typ in der Software-Entwicklung, sie steht aber im Widerspruch zu den Definitionen im *Semantics Guide*, wo Typen Schnittstellen sind. Problematisch ist auch die Definition primitiver Typen: "Primitive Type: A predefined basic type, such as an integer or a string" ([Rat97d], S. 11), da unklar ist, wie diese Definition in Relation zu den anderen Typ-Definitionen steht.

In Version 1.1 der UML wurde die neue Klasse *Classifier* in das Metamodell eingefügt. Diese Klasse übernimmt im wesentlichen die Bedeutung der Klasse *Type* aus Version 1.0: "A classifier is an element that describes behavioral and structural features" ([Rat97q], S. 21). *Type* selbst ist als *Stereotype* von *Class* realisiert. Damit ist nun im Gegensatz zu Version 1.0 *Type* von *Class* abgeleitet. Im *Notation Guide* Version 1.1 ist die Bedeutung von *Type* nun wie folgt erklärt: "A Type characterizes a changeable role that an object may adopt and later abandon. ... . An object may have multiple Types (which may change dynamically) but only one ImplementationClass (which is fixed)" ([Rat97p], S. 35). Auch der *Semantics Guide* Version 1.1 enthält diese Festlegung: "An object may have multiple classes, i.e. it may originate from several classes. ... Moreover, the set of classes (i.e., the set of features that the object conforms to) may vary over time. New classes may be added to the object and old ones may be

1. Vgl. UML Version 1.1, die einen anderen Typbegriff verwendet als UML Version 1.0.

detached" ([Rat97q], S. 77). Diese Festlegung steht im Widerspruch zur gängigen Praxis der meisten klassenbasierten Programmiersprachen, wo jedes Objekt genau Instanz einer Klasse ist. Interessanterweise findet sich diese Festlegung im Glossar vom *Semantics Guide*: "An object is an instance of a class" ([Rat97q], S. 155). Welche Festlegung ist nun verbindlich? Nach der neuen Festlegung werden Typen jetzt im Sinne von Rollen verwendet.

Abb. 6 zeigt einen Ausschnitt aus dem Metamodell der OML für die Beziehung zwischen Typ, Klasse und Klassenimplementierung. Eine Klasse wird eigentlich nicht durch einen Typ implementiert, sondern die Klasse ist die Implementation eines Typs, d.h., ein Typ "is implemented by" einer Klasse. Offen sind dagegen Fragen, wie:

- Kann ein Typ durch verschiedene Implementierungen realisiert werden?[1]
- Muß eine Klassenimplementierung einen Typ vollständig implementieren?
- Kann sie umfangreicher sein, als für den Typ nötig?
- Semantik der *Implements*-Beziehung?

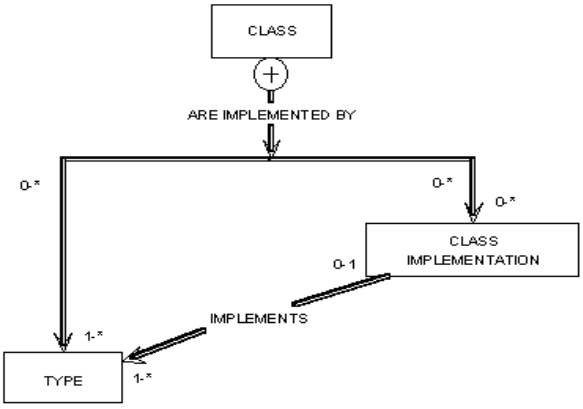

Abb. 6: Klassen, Typen und Klassenimplementierung in OML ([FiHe96], S. 14)

Werden Typen auf Signaturen oder Schnittstellen reduziert, dann fehlt jegliche semantische Beschreibung der Funktionalität. Die konkrete Implementierung einer Klasse kann ja gerade als operationale Semantik aufgefaßt werden. Die Idee, eine Klasse in ihre Signatur und ihre Implementierung zu trennen, ist zu begrüßen. Die Signatur entspricht der statischen Semantik. Die Verwendung des Begriffs Typ für Schnittstellen, ohne Berücksichtigung der dynamischen Semantik, führt jedoch zu einer erneuten Begriffsverwirrung. Vielmehr muß eine Unterscheidung in Spezifikation und Implementierung erfolgen. Die Spezifikation setzt sich aus Signatur und dynamischer Semantik zusammen. Dies entspricht der Trennung in abstrakten und konkreten Datentyp. Der Begriff *Type* in Abb. 7 entspricht damit der *Class specification* und nicht der externen Schnittstelle. Die Klassenspezifikation kann als Typ aufgefaßt werden, da sie die abstrakte Bedeutung der Klasse definiert.

1. Die Kardinalität der "implements"-Beziehung zwischen *Type* und *ClassImplementation* schließt dies derzeit aus. Es gibt aber für Datentypen zeit- oder speicheroptimierte Implementierungen.

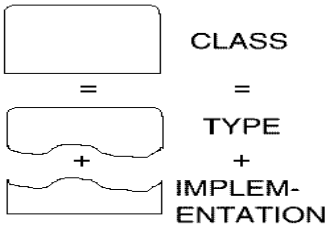

Abb. 7: Zusammenhang zwischen Class, Type und Implementation

Auch der Begriff *Interface* wird ähnlich widersprüchlich wie *Type* oder *Class* benutzt. So heißt es: "An interface is the use of a type to describe the externally-visible behavior of a class, component, or other entity (including summarization units such as packages)" und weiter: "An interface is a type and may also be shown using the full rectangle symbol with compartments" ([Rat97i], S. 25). Oder: "An interface is a stereotyped type" ([Rat97c], S. 31). Bei diesen verschiedenen Aussagen bleibt nur die Frage, was ein *Interface* nun wirklich ist?

Die zusätzliche Betrachtung von *Interfaces*, wie Java sie verwendet, ist hilfreich, um z. B. gemeinsame Methodenprotokolle zu definieren. Diese *Interfaces* beschreiben jedoch nur die Signatur und nicht die dynamische Semantik. Die Reduzierung der Spezifikation auf die Signatur erfolgt hauptsächlich, da viele Programmiersprachen nur die statische Semantik sichern können. Für die Modellierung sind die Spezifikationen jedoch wesentlich wichtiger als die Signatur, zumal die Signatur ein Teil der Spezifikation ist. So betrachtet man Klassen während der Analyse und im Entwurf meist bezüglich ihrer öffentlichen Spezifikation und abstrahiert von der Implementierung.

UML und OML liefern durch die Verflechtung von Typ, Schnittstelle und Klasse sowie ihren widersprüchlichen Vereinbarungen keine Aufklärung über die Beziehungen zwischen Typ, Klasse und Schnittstelle. Insbesondere verpassen sie eine einheitliche Festlegung des Typbegriffs, der sich am Begriff abstrakter Datentyp orientiert und eine vollständige Semantik umfaßt. Meyers Festlegung, eine Klasse als konkreten Datentyp und Modell eines abstrakten Datentyps zu interpretieren, ist deutlich konsistenter mit der Verwendung von Klassen in Programmiersprachen. Für eine Modellierungssprache bleibt festzuhalten, daß nicht Schnittstellen, sondern Spezifikationen unterstützt werden müssen.

6.1.1.1.3 Rollen/Delegationskonzept

OML definiert das Sprachkonstrukt Rolle als "any partial declaration of a kind of object, the declared characteristics of which are required to fulfill a cohesive set of responsibilities. Such an object is said to play the role" ([FiHe96], S. 228). Rollen dienen als Platzhalter für andere Objekte, die die Aufgabe der Rolle übernehmen können. Rollen erlauben damit die Modellierung generischer Lösungen[1]. Rollen legen eine Spezifikation fest, die ein Objekt erfüllen muß, wenn es die Rolle übernehmen will. Die Definition von Rolle zeigt sehr große Ähnlichkeit zur Definition von Typ (vgl. Abschnitt 6.1.1.1.2, S. 56). Im Prinzip sind Rollen auch Typen. Diese Idee greift UML in der Version 1.1 auf, ohne jedoch spezielle Angaben über die Anwendung zu machen. Die Trennung zwischen Typ und Rolle erfolgt meist nur, weil die Typisierung eines Exemplars als statisch angesehen wird, während Rollen eine dynamische Klassifikation erlauben.

Ein Beispiel des Rollenkonzept aus OML ist: "The Customer and Employee delegate common responsibilities to the Person class, thereby allowing the same person object to be both a customer and an employee"[2] ([FiHe96], S. 36).

1. Verschiedene Modellierungsansätze basieren sehr stark auf diesem Konzept. Vgl. dazu auch [Ree95].
2. In [FrHa97] wird mit dem Delegationskonzept ein vergleichbarer Rollenbegriff vorgestellt.

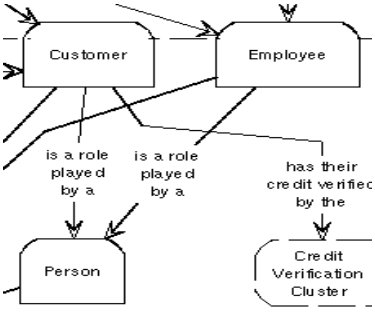

Abb. 8: Rollenkonzept in OML ([FiHe96], S. 36)

Das Rollenkonzept wird intensiv im Metamodell von OML verwendet (vgl. Abb. 5, S. 56). Allerdings fehlt eine präzise Definition der verwendeten Sprachkonstrukte. So ist die Bedeutung der Beziehung "is a role played by" aus Abb. 8 nicht klar spezifiziert. Auch die Eigenschaften, die ein Objekt erfüllen muß, um eine bestimmte Rolle spielen zu können, sind nicht festgelegt. Ein weiteres Problem ist das Anwenden und Erkennen von Rollen. Während Klassen, Subsysteme und Objekte auch in den späten Entwicklungsphasen eingesetzt werden können, ist das Umsetzen von Rollen in den späten Phasen nicht so offensichtlich, da Rollen keine entsprechenden Implementierungskonstrukte haben und häufig nur als Platzhalter in der Analyse und im Entwurf eingesetzt werden. Dies ist auch ein Grund für die prinzipielle Frage, wann Rollensymbole verwendet werden sollten und welche Objekte eine Rolle spielen dürfen.

Abb. 28, S. 74 zeigt die Darstellung eines Entwurfsmusters mit Rollensymbolen. UML benutzt dafür normale Objektsymbole. Dabei stehen diese Objekte exemplarisch für die tatsächlichen Objekte und sind nach ihren Rollen benannt: "The names of the objects represent their roles within the collaboration. A collaboration is a prototype, so the objects in its context are also prototypes; in each execution of the collaboration they are bound to actual objects" ([Rat97i], S. 28). Die Darstellung der OML betont den generischen Charakter von Entwurfsmustern. Die direkte Verwendung von Objekten in UML zeigt die beteiligten Objekte besser.

Auch UML benutzt Rollen. Es wird jedoch kein eigenes Sprachkonstrukt für Rollen definiert. Zum einen dienen Rollen als Zusatz bei Beziehungen, um die Aufgaben der an einer Beziehung beteiligten Objekte zu kennzeichnen. Zum anderen werden sie ähnlich wie in OML verwendet: "Roles is a shared aggregation of an Instance instance to a collection of Type instances. The responsibility of roles is to specify the role that the given Instance instance is playing at a moment in time/space, where role in this context means the face or faces that the Instance instance is presenting to its clients. Whereas an Instance instance is always the instance of exactly one Type instance, the roles of an Instance instance may change" ([Rat97c], S. 28). Diese Festlegung aus dem Metamodell (siehe auch Abb. 38, S. 85) zeigt jedoch einige Schwächen. Eine *Instance instance* kann als Rollen alle Instanzen der Klasse *Type* und ihre Unterklassen also auch *UseCase*, *Class* und *PrimitiveType* haben. Wie eine *Instance instance* typkompatibel zu Typen sein soll, deren Exemplar sie nicht ist, ist nicht definiert und steht im Widerspruch zur Typisierung.

Zusammenfassend läßt sich sagen, das sowohl OML als auch UML das Rollenkonzept unterstützen. Die Einführung von Rollen ist jedoch hauptsächlich ein Tribut an objektorientierte Programmiersprachen, wie Eiffel, Java, C++ oder Smalltalk, in denen Typisierung eine statische Beziehung auf Klassenbasis ist und die keine Mehrfachtypisierung erlauben. Rollen erweitern damit Typen um dynamische Aspekte.

6.1.1.1.4 Stereotypen

Beide Modellierungssprachen definieren Stereotypen. Stereotypen sind im wesentlichen ein Konzept der Metasicht. Sie können aber sowohl aus Anwenderperspektive als auch aus Metamodellperspektive

betrachtet werden. Aus Anwenderperspektive können sie verwendet werden, um bekannte Symbole mit selbst definierten Stereotypen an eigene Bedürfnisse anzupassen oder um bekannten Stereotypen benutzerdefinierte Symbole zuzuordnen. Dies ermöglicht das Anpassen der Notation an spezielle Modellierungs- und Problembereiche. Dies bedeutet aber einen Verlust an Vereinheitlichung, da benutzerdefinierte Symbole und Bedeutungen erlaubt sind. Sie erlauben es dem Anwender auch, Sprachkonzepte in ihrer Bedeutung zu verändern oder neue Konzepte einzuführen. Im Metamodell erlauben Stereotypen die Erweiterung der Sprache, ohne die Grundstruktur des Metamodells zu ändern.

In OML ist ein Stereotyp "any character string of enumeration type that is used to conceptually classify a modeling element independently from inheritance or provide some other important information about the modeling element. Stereotyps provide a facility for metaclassification" ([FiHe96], S. 19).

In UML ist ein Stereotyp einer von drei Erweiterungsmechanismen: "Stereotypes are one of the extension mechanisms and extend the semantics of the metamodel. User-defined icons can be associated with given stereotypes for tailoring the UML to specific processes" ([Rat97b], S. 9). In UML sind die Stereotypen vollständig in das Metamodell integriert und werden auch im Metamodell intensiv angewendet. Stereotypen sind in diesem Fall ein Erweiterungsmechanismus für die Klassen im UML-Metamodell: "Specifically, an Element instance E classified by Stereotype instance S is semantically equivalent to a new metamodel class with the same name as S and whose supertype is the Element instance E" ([Rat97b], S. 15). Stereotypen sind damit eine Form des Metamodell-Subtyping. Sie werden in UML hauptsächlich verwendet, um die UML erweitern zu können, ohne die Grundstruktur des Metamodells ändern zu müssen. Dies ist z.B. für Werkzeughersteller vorteilhaft, die bei neuen Versionen von UML die alten *Repositories* weiter benutzen können, so fern die Grundstruktur nicht wesentlich verändert wurde. Werden Stereotypen im Metamodell zur Ableitung neuer Konzepte benutzt, so ist eine exakte Definition für Stereotyp notwendig, da ansonsten die Semantik der abgeleiteten Konzepte ungewiß ist. Durch die Verflechtung von Meta- und Sprachebene entstehen im Zusammenhang mit Stereotypen recht unverständliche Aussagen: "Each stereotype is a stereotype «stereotype» of a class (yes, this is a self-referential usage!)" ([Rat97b], S. 17).

Die Verwendung von Stereotypen bewirkt einige Vorteile. Sie können die Anzahl benötigter Symbole erheblich reduzieren, indem ähnliche Sprachelemente das gleiche graphische Symbol zugeordnet bekommen und die Unterscheidung durch textuelle Zusatzinformationen erfolgt. Stereotypen sind dann ein Teil der Notation, und die zugehörigen Sprachelemente sind in der Sprachbeschreibung definiert. Eine zweite Möglichkeit ist die kommentarartige Verwendung von Stereotypen zur Gruppierung von Sprachmitteln. Diese beiden Möglichkeiten sind unproblematisch, da sie keine Auswirkung auf die Sprachbeschreibung haben. Etwas anderes ist es jedoch, wenn Stereotypen zur Definition abgeleiteter Sprachelemente und zur Erweiterung der Sprachbeschreibung dienen. Stereotypen sind dann wie Vererbung ein Ableitungsmechanismus und müssen dementsprechend exakt spezifiziert werden, da es ansonsten Probleme bei der Überprüfung der Modelle gibt.

Zusammenfassend läßt sich sagen, daß Stereotypen ein sehr mächtiges und flexibles Mittel zur Erweiterung der Sprache sind, aber auch eine potentielle Gefahr für Verständlichkeit, Wohldefiniertheit und Einheitlichkeit in sich bergen.

6.1.1.1.5 Generische Klasse

OML kennzeichnet generische Klassen nur mit dem Stereotyp <<parameterized>> ([FiHe96], S. 27). UML benutzt zur Darstellung generischer Klassen ein gestricheltes Rechteck mit den generischen Parametern am rechten oberen Symbolrand. UML schränkt die Verwendung generischer Klassen stark ein: "A template cannot be used directly in an ordinary relationship such as generalization or association, because it has a free parameter that is not meaningful outside of a scope that declares the parameter. To be used, a template's parameters must be bound to actual values" ([Rat97i], S. 27). In OML konnten hierzu keine Aussagen gefunden werden. Die Festlegung von UML ist jedoch zu strikt und widerspricht der Praxis in Programmiersprachen, die Generizität unterstützen. So erlaubt Eiffel das

Ableiten generischer Klassen von generischen Klassen: "class BI_LINKABLE [G] inherit LINKABLE [G]" ([Mey97], S. 598). Dabei ist die neue Klasse *BI_LINKABLE [G]* wieder eine generische Klasse. Eine normale Klasse kann dagegen nur von einer konkreten Instanzierung einer generischen Klasse abgeleitet werden: "class WINDOW inherit TREE [WINDOW] RECTANGLE ... Note that class TREE will be generic, so we need to specify an actual generic parameter, here WINDOW itself" ([Mey97], S. 525). Die Entscheidung von UML stimmt nicht mit der Festlegung in Programmiersprachen (Eiffel) überein. Auch die generische Baumhierarchie aus Abb. 9 wäre mit der Festlegung der UML nicht erlaubt.

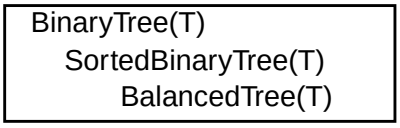

Abb. 9: Generische Klassen

6.1.1.1.6 Gruppierungsmechanismen

Cluster bzw. *Package* sind die Gruppierungsmechanismen von OML und UML. Der Begriff *Cluster* ist in OML sehr weitläufig definiert und steht für Unit, Layer, System, Modul, Entwurfsmuster und Kategorie ([FiHe96], S. 38). Fraglich ist, ob ein Gruppierungsmechanismus für alle diese Speziallfälle ausreicht. OML definiert *Cluster*-Klassen und *Cluster*-Exemplare. Auch für *Cluster* gibt es in OML keine exakte Definition. UML definiert als generellen Gruppierungsmechanismus *Packages*, die im wesentlichen den *Clustern* von OML entsprechen. Allerdings wendet UML nicht die Typ/Exemplar-Dichotomie auf *Packages* an. Es gibt keine *Package class* und somit keine Möglichkeit, für mehrere Pakete eine einheitliche Beschreibung festzulegen. OML bietet hier bessere Möglichkeiten als UML. Für *Packages* sind in UML eine Reihe von Sichtbarkeitsregeln für Importbeziehungen festgelegt. Diese sind jedoch im Zusammenhang mit Aliasen für *Package*-Komponenten stellenweise widersprüchlich. Auch ist der Unterschied zwischen Importbeziehung und Generalisierung im Zusammenhang mit Pakkages nicht offensichtlich. Generalisierung scheint eher eine Importbeziehung mit aufgehobenen Sichtbarkeitsregeln zu sein, als eine Ableitung oder Verfeinerung. Wie OML erlaubt auch UML die Verfeinerung von Paketen durch Generalisierungsbeziehungen. Zusätzlich gibt es noch *Collaborations*. Pakete gruppieren hauptsächlich strukturelle und statische Sprachmittel. *Collaborations* beschreiben Mechanismen und Interaktionen.

6.1.1.2 Beziehungen

Beziehungen sind wichtige Sprachmittel in objektorientierten Modellierungssprachen. Sie setzen ein oder mehrere Sprachelemente in Relation. Auch für Beziehungen ist eine Festlegung der Semantik wichtig. Beziehungen werden in allen Teilsichten objektorientierter Modellierungssprachen verwendet.

6.1.1.2.1 Generalisierung

UML benutzt einen sehr weit gefaßten Vererbungsbegriff: "Generalization is a unidirectional inheritance relationship, uniting two more generalizable elements in a supertype/subtype hierarchy, wherein an instance of the subtype is substitutable for an instance of the supertype. A supertype generalizes a subtype" ([Rat97c], S. 35). Insbesondere setzt diese Definition Generalisierung mit *Subtyping* gleich. Durch die Typ/Exemplar-Dichotomie ist es möglich, Generalisierung auch für andere Komponenten als Klassen (Typen) zu definieren. Generalisierung ist: "A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed" ([Rat97d], S. 7). Diese Festlegung bedingt die strikte Substituierbarkeit des allgemeinen Elements durch das spezielle Element. Gilt dies auch für die Stereotypen der Generalisierungsbeziehung? Ähnlich wie durch Vererbung eine neue Klasse von bestehenden abgeleitet werden kann, können auch Teilsysteme oder Szenarien die Definition schon

existierender Teilsysteme oder Szenarien benutzen. Voraussetzung ist nur eine exakte Definition dieser neuen Vererbungsbeziehungen.

In Abb. 10 ist ein Beispiel für *Cluster*-Vererbung in OML gezeigt. Gerade für Subsysteme ist es naheliegend, Vererbung zu benutzen. Neben einer Kompositionshierarchie können so zusätzlich Ableitungshierarchien aufgebaut werden. In der Analysephase werden oft Objekte identifiziert und klassifiziert, die sich später als eigenständige Teilsysteme erweisen. Die Typ/Exemplar-Dichotomie gewährleistet die einheitliche Behandlung dieser Komponenten. Problematisch ist allerdings, daß sowohl UML als auch OML die vollständige Semantik dieser Beziehungen nicht definieren. Dies sind nur einige Fragen, die unbeantwortet bleiben.

- Ist die Vererbung mit der Übernahme aller Komponenten verbunden?
- Welche Bedeutung hat die Vererbung?
- Was passiert, wenn Komponenten selbst abgeleitet werden?

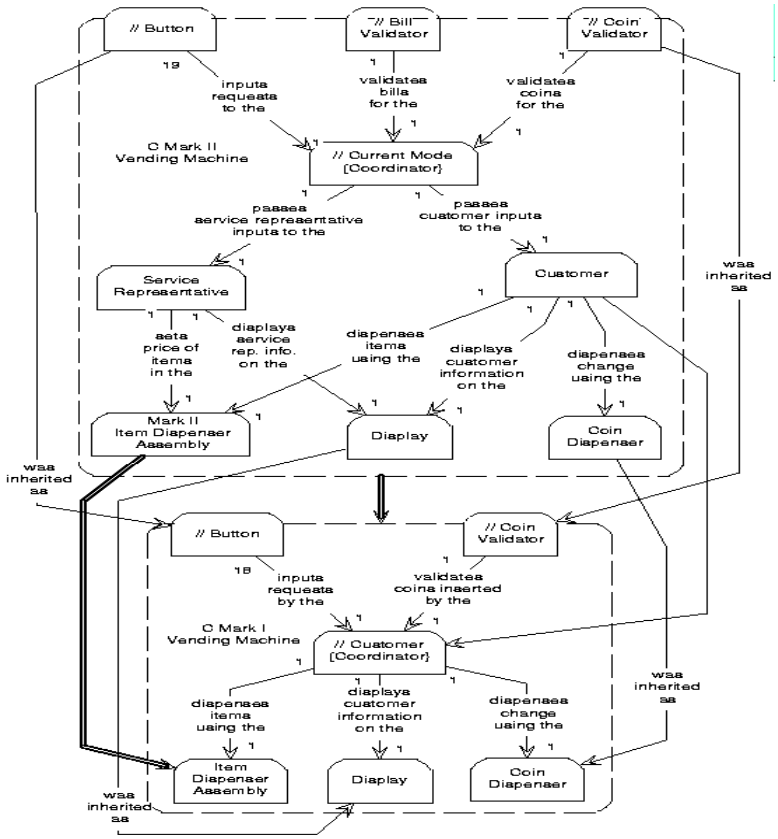

Abb. 10: Cluster Vererbung ([FiHe96], S. 47)

Auch UML benutzt Generalisierungsbeziehungen nicht nur für Klassen. So sind die Beziehungen zwischen *Use Cases* Generalisierungsbeziehungen. Die Erweiterungsbeziehung ist definiert als: "An extends relationship between use cases is shown by a generalization arrow from the use case providing the extension to the base use case. An extends relationship from use case A to use case B indicates that an instance of use case B may include (subject to specific conditions specified in the extension) the behavior specified by A" und die Benutzungsbeziehung als: "A uses relationship between use cases is shown by a generalization arrow from the use case doing the use to the use case being used. The arrow is labeled with the stereotype «uses». A uses relationship from use case A to use case B indicates that an instance of the use case A will also include the behavior as specified by B" ([Rat97i], S. 64). Auch

in diesem Fall ist die Bedeutung der Beziehungen nicht exakt definiert. Ein weiteres Problem ist, daß der Stereotyp <<uses>> eigentlich eine Benutzung und nicht eine Ableitung impliziert.

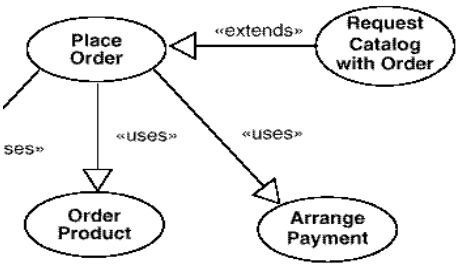

Abb. 11: Uses Beziehung für Use Cases in UML ([Rat97i], S. 65)

OML und UML unterstützen verschiedene Arten von *Inheritance* (*public, private, protected; interface, implementation; whitebox, blackbox*). Sie unternehmen nicht den Versuch, alle Varianten mit einer Version abzudecken. Diese Unterscheidung der einzelnen Varianten ist wichtig. Leider bieten sie keine griffigen Definitionen für die festgelegte Semantik dieser Beziehungen und keine Überprüfung, ob diese Spezialisierungen auch die Anforderungen an Generalisierungsbeziehungen erfüllen.

6.1.1.2.2 Aggregation

Für Aggregation und verwandte Beziehungen gibt es in UML und OML keine einheitliche Definition. OML unterscheidet zwei Formen, die eigentliche Aggregation und *Containment* (Enthaltensein). Aggregation entspricht dabei der Teil/Ganzes-Beziehung ([FiHe96], S. 106 ff.). Im Gegensatz zu UML (*Composition*) werden an diese Beziehung keine weiteren Forderungen gestellt. So kann ein Teil auch unabhängig von seinem Ganzen weiterbenutzt werden. OML verwendet auch den Begriff bidirektionale Aggregation. In dieser Beziehung existiert zusätzlich eine Referenzbeziehung vom Teil zum Ganzen, wie es z. B. bei doppelt verketteten Bäumen der Fall ist. Allerdings ist es fraglich, ob man diese Beziehung als Aggregation bezeichnen kann. Auch OML stellt fest: "Note that bi-directional aggregation does not mean that the part contains the aggregate" ([FiHe96], S. 106). Als Beispiel für bidirektionale Aggregation wird das Vermittlermuster (*mediator pattern*[1]) angegeben, wo Vermittler und Teilnehmer bidirektional verbunden sind. Allerdings kann diese Verbindung auch als einfache Referenz interpretiert werden. Der Begriff Aggregation ist in OML sehr schwach definiert. Er gibt nur eine Erklärung, wann eine Referenz als Aggregation betrachtet werden kann, stellt aber keine zusätzlichen überprüfbaren Forderungen. Zusätzlich zur Aggregation gibt es in OML noch eine Enthaltenseinbeziehung. Dabei handelt es sich um eine Ist-Element-Beziehung. Ein Element ist in einer Menge (*Container*) enthalten. Als Beispiel können die Bücher einer Bibliothek dienen. Zwischen diesen und der Bibliothek besteht eher eine Element- bzw. Enthaltensein-Beziehung als eine Teil/Ganzes-Beziehung. Auch die Enthaltenseinbeziehung ist im wesentlichen nur eine Referenzbeziehung, da keine verbindlichen Forderungen vereinbart sind. Die Unterscheidung in Aggregation und Enthaltensein ist in OML hauptsächlich als Vereinfachung für den Benutzer gedacht. Da Referenzbeziehung, Enthaltensein und Aggregation auf Metaebene keine wesentlichen Unterschiede aufweisen, könnte der Verwendungszweck einer Referenzbeziehung als Aggregation oder Enthaltensein auch durch <<enthält>> und <<ist Teil>> dargestellt werden. Aus unserer Sicht ist eine Unterscheidung zwischen diesen beiden Formen jedoch problematisch, da es zum einen keine exakte Unterscheidung zwischen beiden Beziehungsarten gibt und zum anderen beide auf die gleiche Art im Metamodell repräsentiert werden.

UML unterscheidet *Composition, Aggregation* (und einfache Referenzen). Aggregation wird in UML als Erweiterung von Assoziationen zur Spezifikation von Ganzes/Teilbeziehungen verwendet und nicht näher erklärt. Die inhaltliche Bedeutung von Aggregation ist nur ungenau festgelegt. Die Definition von Komposition dagegen ist sehr speziell und einengend. Komposition definiert die stärkste Kopplung zwischen den Objekten. UML definiert Komposition als: "Composition is a form of aggregation with

1. Vgl. etwa [GHJV95].

strong ownership and coincident lifetime of part with the whole. The multiplicity of the aggregate end may not exceed one (it is unshared). The aggregation is unchangeable (once established the links may not be changed). Parts with multiplicity > 1 may be created after the aggregate itself but once created they live and die with it " ([Rat97i], S. 47). Einige Ausdrücke wie "strong ownership" sind in dieser Definition selbst nicht definiert oder mißverständlich. "The multiplicity of the aggregate end may not exceed one (it is unshared)" sichert nur, daß das Teil nicht mit einem weiteren Aggregat bezüglich dieser konkreten Aggregationsbeziehung in Beziehung steht. Es sichert aber auf gar keinen Fall, daß dieses Element nicht auch für andere Elemente verfügbar ist! Dieser Zugriff kann durch andere Beziehungen erfolgen. Das Element ist dann zwischen diesen Elementen geteilt. Auch die Lebensdauer ist ein fragwürdiger Indikator für Komposition: "The life of a part is usually, but not always, confined to that of its aggregate. As a counterexample, a car engine may be reused meaning that it may exist after the first car exists but before the second car exists" ([FiHe96], S. 106). Zum Verhältnis zwischen Attribut und Komposition stellt UML fest: "Note that an attribute is semantically equivalent to a composition association" ([Rat97i], S. 32). Gilt die obige Definition von Komposition, dann ist diese Aussage unzutreffend, weil Attribute sich während der Lebenszeit eines Objektes ändern können. Dies wird aber durch den Passus "The aggregation is unchangeable (once established the links may not be changed)" nicht erlaubt!

Im Zusammenhang mit Aggregation wird mitunter auch von Transitivität gesprochen. Dabei wird jedoch nicht unterschieden, ob sich diese Aussage auf die Beziehungen im Modell oder auf den zu modellierenden Sachverhalt bezieht. Für Beziehungen im Modell ist diese Aussage nicht richtig. Transitivität entspricht in diesem Fall der Transitivität bei Relationen[1]. In Abb. 12 existiert eine Aggregation von *A* nach *B* und von *B* nach *C*. Ist die Beziehung transitiv, dann müßte auch eine Beziehung von *A* nach *C* existieren. Dies ist aber nicht der Fall! Es spielt dabei keine Rolle, ob die Beziehung von *A* nach *C* durch die beiden anderen simuliert werden kann.

Meist bezieht sich Transitivität auf den zu modellierenden Sachverhalt. D. h., *A* enthält *B*, *B* enthält *C* und aus dem betrachteten Sachverhalt folgt auch, daß *A C* enthält[2]. Auf Modellierungsebene kann diese dritte Beziehung aber eventuell vernachlässigt werden. Nicht die Beziehungen erfüllen die Eigenschaft, sondern der zu modellierende Sachverhalt soll diese Eigenschaft erfüllen. Der Sachverhalt wird darauf getestet, ob es richtig ist, diesen Aspekt durch eine Aggregation auszudrücken. Da aber nicht geklärt ist, ob sich Transitivität auf die Beziehungen oder den Sachverhalt bezieht, ist der Begriff Transitivität in diesem Zusammenhang mehrdeutig.

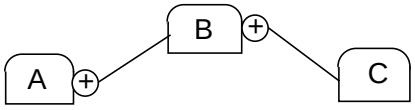

Abb. 12: Transitivität bei Beziehungen

Ein weiterer Kritikpunkt an der Verwendung von Relationseigenschaften ist die Feststellung, daß nicht geklärt ist, ob sich die Aussage auf die Klassen- oder auf die Exemplarebene bezieht. Abb. 13 zeigt auf Klassenebene, daß zwischen *(A,B)* und *(B,A)* jeweils eine Beziehung besteht. Auf Objektebene ist diese Struktur im Beispiel nicht mehr erhalten. Werden Beziehungen auf Klassenebene angegeben, muß zusätzlich beschrieben werden, wie die Beziehungen sich auf der Objektebene widerspiegeln.

1. Eine Relation R ($R \subseteq M \times M$) ist transitiv, wenn gilt: $(x, y) \in R \wedge (y, z) \in R \rightarrow (x, z) \in R$ für alle $x, y, z \in M$.
2. Vgl. hierzu [MoPi93], wo verschiedene Kategorien von Teil/Ganzes-Beziehungen bezüglich ihrer transitiven Verknüpfung untersucht werden.

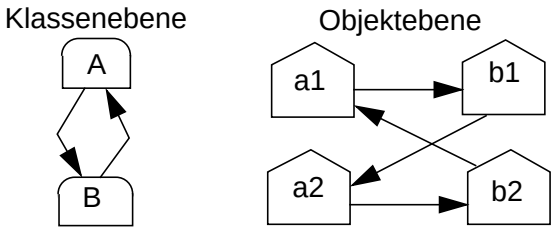

Abb. 13: Beziehungen auf Klassen- und Objektebene

6.1.1.2.3 Beziehungen höherer Ordnung

OML unterstützt keine n-ären Beziehungen. Solche Beziehungen sollen als Klassen modelliert werden. UML unterstützt wie schon OMT n-äre Beziehungen. Für die Analyse können Beziehungen höherer Ordnung sinnvoll sein, da sie wesentliche Zusammenhänge zwischen verschiedenen Objekten realitätsnaher beschreiben können. Im Entwurf und in der Implementierungsphase müssen Beziehungen höherer Ordnung in Klassen und binäre Beziehungen aufgelöst werden. Dies verlangt, daß Analysemodelle verfeinert werden können und geeignete Transformationsregeln durch die Sprache festgelegt sind.

6.1.1.2.4 Or-Assoziationen

UML unterstützt sogenannte Or-Assoziationen. Abb. 14 zeigt ein Beispiel aus dem Notation Guide. Allerdings ist die Bezeichnung *or* irreführend, da es sich um die Auswahl maximal eines Elementes aus einer Anzahl vorgegebener Elemente handelt ([Rat97c], S. 41). "1 aus n" drückt die logische Semantik der Bedingung besser aus! Offen ist auch, ob bei 1:N Beziehungen nur jeweils Objekte einer Klasse erlaubt sind. Ein weiteres Problem tritt bei typisierten Sprachen auf. Or-Assoziationen können in diesen nicht direkt in Programmcode transformiert werden.

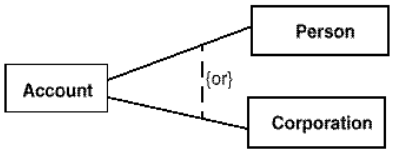

Abb. 14: Or-Assoziationen in UML ([Rat97i], S. 37)

Aus diesem Grund unterstützt OML keine Or-Assoziationen. Statt dessen wird als Alternative Vererbung vorgeschlagen (Abb. 15). Bei Einfachvererbung ist dies jedoch nicht immer möglich, da die Vererbungsbeziehung für eine andere Klassifikation reserviert sein kann. Or-Assoziationen können jedoch auch durch Rollen oder Interfaces (Java) aufgelöst werden. In untypisierten Sprachen wie Smalltalk können Or-Associations direkt implementiert werden.

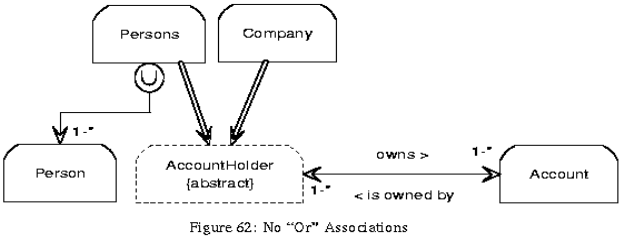

Abb. 15: Auflösen von Or-Assoziationen in OML ([FiHe96], S. 103)

Ähnlich wie Beziehungen höherer Ordnung stellen Or-Assoziationen ein hilfreiches Konzept während der Analyse dar. Für die Implementierung müssen sie jedoch umgebaut werden. Die Modellierungs-

sprache muß daher eine Verfeinerung von Analysemodellen zu Implementierungsmodellen unterstützen.

6.1.2 Konkrete Syntax

Die konkrete Syntax beschreibt die Darstellung der abstrakten Syntax und Semantik und bildet gewissermaßen die Schnittstelle zum Betrachter. Bei ihrer Untersuchung stehen daher die anwenderbezogenen Kriterien im Vordergrund. Ein weiteres wichtiges Untersuchungskriterium ist das Verhältnis zwischen konkreter und abstrakter Syntax. Die Vollständigkeit bezüglich der konkreten Beschreibung einer Modellierungssprache ist ein Maß für die Fähigkeit der konkreten Syntax, die abstrakte Syntax und Semantik darzustellen. Die Modellierungssprache muß die vollständige Beschreibung eines mit der Sprache modellierten Modells gewährleisten.

6.1.2.1 Annotationen

UML und OML enthalten keine anwenderorientierten Annotationen[1] für ihre Sprachmittel. Die UML legt die Bereitstellung textueller Beschreibungen in den Aufgabenbereich der Werkzeughersteller: "The UML does not assume that all of the information in a model will be expressed as diagrams; some of it may only be available as tables. This document does not attempt to prescribe the format of such tables or of the forms that are used to access them, because the underlying information is adequately described in the UML metamodel and the responsibility for presenting tabular information is a tool responsibility" ([Rat97i], S. 5). UML legt den Sprachumfang für die Modellbeschreibung fest. Wie UML feststellt, reichen Diagramme für eine vollständige Modellbeschreibung nicht aus. Die graphische Notation ist nicht ausdrucksmächtig genug, die abstrakte Syntax und Semantik zu beschreiben. Als Modellierungssprache muß UML aber Sprachmittel für eine vollständige Modellbeschreibung bereitstellen. Auch wenn die fehlenden Informationen im Metamodell verankert sind, so ist UML wegen dieser fehlenden Beschreibungsformen nicht in der Lage, Anwendermodelle vollständig zu beschreiben. Erst die Kombination von UML und werkzeugherstellerspezifischen Erweiterungen erlauben eventuell eine vollständige Darstellung. Hier ist dringend eine Erweiterung notwendig.

Wie wichtig textuelle Zusammenfassungen für die einzelnen Sprachmittel sind, zeigt auch folgende Passage: "In general, each appearance of a class in a class diagram presents only those attributes and operations that are relevant to that diagram; the complete interface of a class must be constructed from each appearance of that class across all diagrams" ([Rat97c], S. 2 f.). Um die Bedeutung einer Klasse zu verstehen, müssen alle Diagramme betrachtet werden, die die Klasse darstellen. Textuelle Repräsentationen für die Sprachmittel vermeiden dieses umständliche Verfahren durch eine vollständige Spezifikation. Weiterhin erlauben sie eine textuelle Darstellung ohne Diagramme und vermeiden Redundanz. Sie wären eine verbindliche Basis für die einzelnen Konzepte und könnten direkt aus der Sprachbeschreibung abgeleitet werden.

Auch OML unterstützt keine umfangreichen textuellen Annotationen. Zwar gibt es verschiedene CRC-Cards, aber vollständige Spezifikationen fehlen. In beiden Sprachen fehlen damit wichtige Beschreibungsmittel für die angestrebte Modelldarstellung.

Der Verzicht auf textuelle Spezifikationen bedeutet eine Einschränkung der Vereinheitlichung, weil es in der Verantwortung der Werkzeughersteller liegt, welche textuellen Annotationen bereitgestellt werden. Außerdem ist aus dem Metamodell nicht immer ersichtlich, welche Informationen der einzelnen Modellierungselemente für den Benutzer wichtig sind und welche nicht. Gleichzeitig könnten diese Annotationen die Rolle eines *Data Dictionary* übernehmen.

1. Vgl. hierzu [Boo94], wo Annotationen (*Templates*) für die einzelnen Sprachmittel festgelegt werden.

6.1.2.2 Graphische Anmerkungen

Kommentare und Ergänzungen zu Diagrammen oder einzelnen Diagrammelementen sind eine wichtige Möglichkeit, Diagramme mit zusätzlichen Informationen anzureichern. Sie können reinen Informationscharakter haben. Sie können aber auch wichtige Modellinformationen textuell in die Diagramme integrieren.

6.1.2.2.1 Notizen

UML definiert eine Notiz als: "a comment placed on the diagram. It is attached to the diagram rather than to a model element, unless it is stereotyped to be a constraint" ([Rat97i], S. 5). Es gibt somit keine Notizen für einzelne Diagrammelemente. Warum erfolgt diese Einschränkung? Sie ist eigentlich überflüssig, da Notizen im allgemeinen nur textuelle nicht auswertbare Hinweise enthalten. Eine Zuordnung zwischen Notiz und Diagrammelement wäre also kein Problem.

Darüberhinaus können Notizen zur Darstellung von Bedingungen[1] verwendet werden, wie in Abb. 16 gezeigt ist. Dies ist ein Mißbrauch des Notizsymbols. Ein Notizsymbol assoziiert eine Zusatzinformation, die gegebenenfalls ignoriert werden kann. Eine Bedingung ist ein wichtiges Modellierungselement und keine Notiz! Bei Werkzeugen ergibt sich beim Ausblenden von Notizen zusätzlich das Problem, daß das Werkzeug prüfen muß, ob eine Notiz eine Bedingung ist. Mit der Verwendung des Notizsymbols für Bedingungen verstößt UML gegen die Regel, Notationselemente nicht zu überladen oder mehrfach zu verwenden (vgl. S. 29).

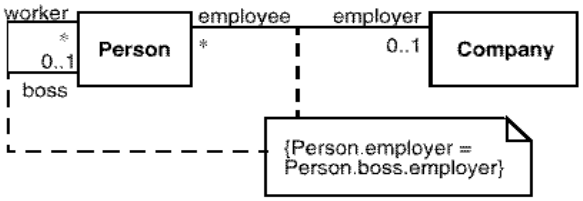

Abb. 16: Notizen als Bedingungen ([Rat97i], S. 7)

6.1.2.2.2 *Drop-Down*-Boxen

Beide Notationen verfügen über die Möglichkeit, Zusatzinformationen für die Symbole darzustellen. Während die UML dies nur für bestimmte Notationselemente innerhalb des Symbols erlaubt, kann in der OML jedes Notationselement durch sogenannte Drop-Down-Boxen erweitert werden. Die Informationen zu einem Symbol variieren während des Entwicklungszyklus und Einsatzzweckes. Jedes Symbol sollte daher annotierbar sein**.** Abb. 17 zeigt die textuellen Symbolannotationen von UML und OML. OML benutzt angehängte Textboxen, während UML die Annotationen in die Symbole integriert.

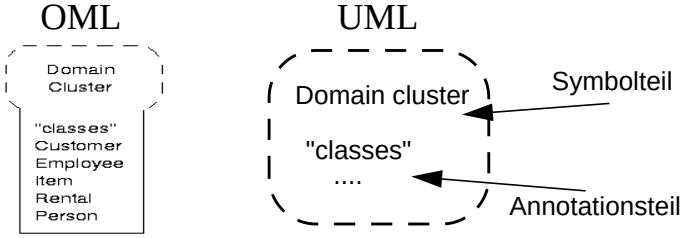

Abb. 17: Textuelle Symbolannotationen in UML und OML

1. Abb. 29, S. 74 zeigt ein Diagramm für ein Entwurfsmuster, in dem das Notizsymbol für Bedingungen verwendet wird. In diesem Diagramm fehlen die geschweiften Klammern der Bedingung!

Die OML-Entwickler favorisieren ihre Lösung durch folgendes Argument: "Dividing icons into multiple compartments (OMT, UML) was rejected as nonworkable because it does not scale well" ([FiHe96], S. 57). Diese Argumentation überzeugt nicht. Ein Symbol setzt sich aus zwei Teilen zusammen, einem Symbol- und einem Annotationsteil. Die feste Einteilung, wie sie die UML vorsieht, hat dagegen keinen Einfluß auf die Integration von Symbol- und Annotationsteil. Auch die Drop-Down-Boxen haben Nachteile. Sie verändern den Symbolumriß.

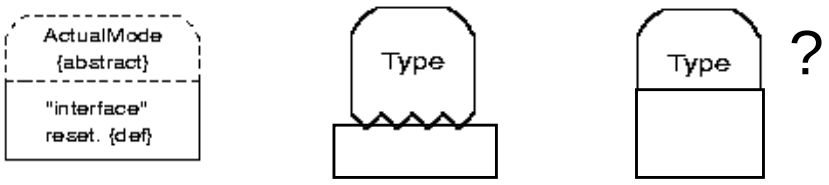

Abb. 18: Ungünstige Benutzung von Drop-Down-Boxen

Abb. 18 zeigt einige problematische Darstellungen, die den Symbolumriß oder die verwendete Linienart beeinträchtigen. Die bevorzugte Variante ist daher, die Annotationen in die Symbole zu integrieren, um den Symbolumriß zu erhalten und keine feste Einteilung zu fordern.

6.1.2.3 Ähnliche Diagrammelemente

Für verwandte Sprachmittel werden meist auch ähnliche Repräsentationen verwendet. Dies erleichtert die Anwendung durch Vereinfachung und Vereinheitlichung der Notation. Gleichzeitig ist damit aber auch die Gefahr verbunden, daß einzelne Notationselemente nicht mehr ausreichend zu unterscheiden sind (vgl. S. 29).

6.1.2.3.1 Typ/Exemplar Dichotomie

UML versucht für Typen und Exemplare gleiche Symbole zu benutzen: "Although diagrams for type-like elements and instance-like elements are not exactly the same, they share many similarities. Therefore it is convenient to choose notation for each type-instance pair of elements such that the correspondence is immediately visually apparent. There are a limited number of ways to do this, each with advantages and disadvantages. In UML the type-instance distinction is shown by employing the same geometrical symbol for each pair of elements and by underlining the name string (including type name, if present) of an instance element. This visual distinction is generally easily apparent without being overpowering even when an entire diagram contains instance elements" ([Rat97i], S. 10). Gegen diese Idee ist nichts einzuwenden. Die Verwendung von Unterstrichen scheint jedoch nicht so günstig. In Abb. 19 sind entweder die Objektunterstriche für die Teilobjekte vergessen worden, oder die Verwendung der Unterstreichungen ist inkonsistent.

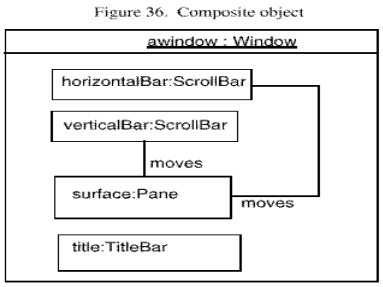

Abb. 19: Inkonsistente Objektunterstreichungen in UML ([Rat97i], S. 82)

OML benutzt für die Unterscheidung von Typ und Exemplar ein "C" vor dem Namen. Abb. 20 zeigt diese Konvention am Beispiel von *Clustern*.

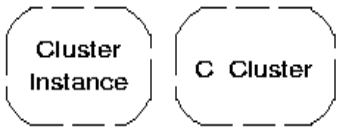

Abb. 20: Darstellung der Typ/Exemplar Dichotomie in OML ([FiHe96], S. 41)

Die Unterscheidung von *Cluster* und *Cluster instance* erfolgt nur durch dieses "C". Abb. 21 zeigt ein Beispiel aus [FiHe96] und veranschaulicht, wie leicht das "C" vergessen werden kann (*Customer "is an instance of " Customer*).

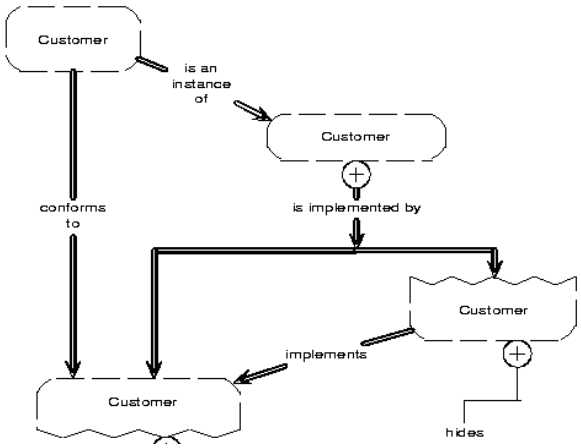

Abb. 21: Customer "is an instance of" Customer? ([FiHe96], S. 78)

Ein analoges Problem entsteht, wenn Bezeichner weggelassen werden. Diesen Fall zeigt Abb. 22. Wegen der fehlenden Bezeichner ist nicht offensichtlich, welches *Cluster*-Symbol eine *Cluster*-Klasse und welches ein Exemplar darstellt. Als Anhaltspunkt können hier nur die Notizen und die enthaltenen Symbole dienen.

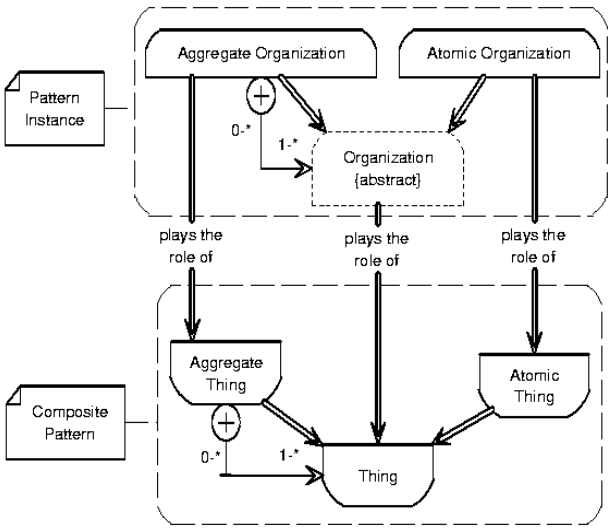

Abb. 22: Fehlende Clusternamen ([FiHe96], S. 111)

UML und OML versuchen, die Typ/Exemplar-Dichotomie durch eine geeignete Notationswahl zu unterstützen. Die Verwendung ähnlicher Symbole für Typ und Exemplar ist zu begrüßen, da sie die Anzahl verwendeter Symbole reduziert. Die Umsetzung ist in beiden Notationen jedoch nicht vollständig gelungen.

6.1.2.3.2 State und Action State Symbol

Die UML benutzt für die Darstellung von *State* und *Action State* zwei verschiedene, aber sehr ähnliche Symbole (Abb. 23). Bei Handzeichnungen sind diese Symbole fast nicht zu unterscheiden. Günstiger wäre hier vielleicht die Verwendung eines Stereotyps <<action>> gewesen. UML verletzt hier die Anforderung nach gut unterscheidbaren Notationselementen (vgl. S. 29).

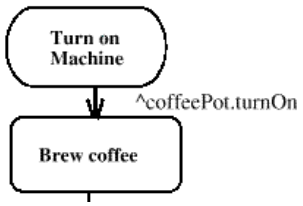

Abb. 23: Ähnliche Zustandssymbole ([Rat97i], S. 107)

6.1.2.4 Notation für Symbole

Aufgrund des großen Sprachumfangs benötigen OML und UML viele verschiedene Symbole zur graphischen Darstellung der Sprachelemente. Um diese Vielzahl einzuschränken, verwenden beide Notationen abgeleitete Sprachelemente mit textuellen Erweiterungen (Stereotypen) oder kleinen Icons zur Unterscheidung. Diese Vorgehensweise ist zu begrüßen, da sie die Vielzahl von Symbolen reduziert.

6.1.2.4.1 Klassen- und Objektsymbole

Die Symbolauswahl für Klasse und Objekt in OML ist Geschmacksache. Die generelle Ächtung des Rechtecks in OML ist übertrieben. Die Symbole für Type und Implementierung mit den Sägezähnen sind schwer von Hand zu zeichnen (vgl. S. 29).

6.1.2.4.2 Multiobjekte

Multiobjekte sind eine gute Möglichkeit, eine Vielzahl gleichartiger Objekte darzustellen. Sie verstekken die Implementierung durch geeignete Container und bieten damit stellenweise eine günstigere Darstellung. Abb. 24 zeigt die Notation für Multiobjekte in OML. Prinzipiell kann man das mehrfache Auftreten ähnlicher Komponenten in einem Diagramm durch versetztes Hintereinanderzeichnen darstellen. Auch die Booch-Notation unterstützte schon Multiobjekte. In UML Version 1.0 existieren keine Hinweise, ob Multiobjekte möglich sind. Allerdings unterstützen verschiedene Werkzeuge für UML diese Darstellung und UML Version 1.1 enthält die Multiobjekt-Darstellung auch ([Rat97p], S. 99).

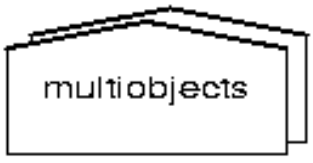

Abb. 24: Multiobjekte in OML ([FiHe96], S. 22)

6.1.2.4.3 Abstrakte Klasse

Die Darstellung abstrakter Elemente ist in beiden Notationen nicht optimal gelöst. OML kennzeichnet abstrakte Elemente durch das Hinzufügen des Stereotypes <<abstract>> oder durch gestrichelte Linien

(Abb. 25). Der gestrichelte Umriß ermöglicht z. B. keine Darstellung abstrakter *Cluster*-Klassen, weil deren Umriß per Default gestrichelt ist. Eine Lösung, die für alle abstrakten Elementtypen anwendbar ist, scheint angemessener und konsistenter.

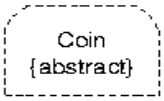

Abb. 25: Abstrakte Klasse in OML ([FiHe96], S. 34)

UML benutzt kursive Schrift zur Hervorhebung abstrakter Klassen (Abb. 26). Diese Festlegung ist aus OMT übernommen. Gerade bei handgezeichneten Diagrammen mit Schreibschrift ist es unmöglich, normale von kursiver Schrift zu unterscheiden.

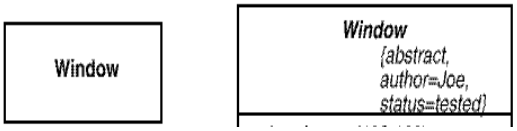

Abb. 26: Abstrakte Klasse in UML ([Rat97i], S. 21)

Am deutlichsten ist die Veranschaulichung abstrakter Sprachelmente durch Stereotypen oder durch kleine Sinnbilder wie in der Booch-Notation.

6.1.2.5 Notation für Beziehungen

Beziehungen können in Diagrammen durch Linien, Nachbarschaftsbeziehungen oder Schachtelungen ausgedrückt werden. Zusätzlich sind Informationen über Art und Richtung der Beziehungen notwendig.

6.1.2.5.1 Gerichtete Beziehungen

In OML sind alle Beziehungen gerichtet, und die Richtung wird auch in der graphischen Repräsentation dargestellt. "A bidirectional linkage between peers is drawn as a single solid line with an arrowhead on both ends as a shorthand for two unidirectional linkages" ([FiHe96], S. 98). Diese Festlegung vermeidet das unübersichtliche Simulieren von bidirektionalen Beziehungen durch unidirektionale Beziehungen. Gleichzeitig kann so zwischen ungerichteten[1], unidirektionalen und bidirektionalen Beziehungen unterschieden werden. Die Problemantik aus Abb. 13, S. 66 wird durch diese Festlegung nicht gelöst. Eine bidirektionale Beziehung auf Klassenebene impliziert meist auch eine bidirektionale Beziehungen zwischen zwei Objekten. Beim Darstellen durch zwei unidirektionale Beziehungen geht diese Information verloren. Unter diesem Gesichtspunkt ist die obige Aussage der OML unexakt, da eine bidirektionale Beziehung mehr Informationen als ihre zwei unidirektionalen Teilbeziehungen umfaßt. Aus diesem Grund sollten für referenzielle Beziehungen explizit bidirektionale Beziehungen unterstützt werden.

UML benutzt im wesentlichen unidirektionale Beziehungen. Es können aber auch bidirektionale Beziehungen verwendet werden. Allerdings verfügt UML über keine Notation, bidirektionale oder nicht gerichtete Beziehungen zu unterscheiden, da keine doppelten Pfeile an den Enden der Beziehungskanten verwendet werden. Hier ist die Notation von OML besser.

6.1.2.5.2 Definierende und referenzielle Beziehungen

OML teilt die Beziehungen im wesentlichen in zwei große Gruppen ein. Es gibt definierende und referenzielle Beziehungen. Alle definierenden Beziehungen werden durch Doppelstriche dargestellt und alle referenziellen durch einfache Striche[2]. Diese Festlegung ist eine gute Konvention. Problematischer

1. Eine ungerichtete Beziehung ist eine Beziehung, deren Richtung noch nicht festgelegt wurde.
2. Uses Relationship in Szenarios werden durch gestrichelte Linien dargestellt. Dies ist inkonsistent zu anderen Beziehungen ([FiHe96], S. 154).

ist dagegen die Festlegung "There is no need to use a different notation for the three different kinds of definititional relationships because each connects a different kind of modeling element" ([FiHe96], S. 73). Dies führt zu einer vermeidbaren Überladung von Notationselementen. Die gleiche Notation für mehrere artgleiche Beziehungen zu verwenden, vereinfacht die Notation. Allerdings sollten diese Notationselmente durch textuelle oder graphische Marker ergänzt werden, die eine schnelle und sichere Unterscheidung der verschiedenen Varianten erlauben. Ansonsten ist es schwierig, die Bedeutung der Beziehungen zu erkennen und zu erinnern. UML führt die Unterscheidung in definierende und referenzielle Beziehungen nicht durch. Es werden auch keine Doppellinien benutzt.

6.1.2.5.3 Bedeutung von Pfeilen an Notationskanten

OML und UML benutzen Pfeile an den Beziehungskanten zur Veranschaulichung der Richtung. UML benutzt auch für die Vererbung einen Pfeil, der nicht ausgemalt ist. In umfangreichen Klassendiagrammen erschwert diese Darstellung die Lesbarkeit, da sich die Pfeile nicht genügend von ihrer Umgebung abheben. Die Notation von OML mit den definierenden Beziehungen ist aus diesem Blickwinkel günstiger.

6.1.2.5.4 Botschaften

OML und UML definieren verschiedene Symbole für den sequentiellen, synchronen und asynchronen Botschaftenaustausch. Beide unterstützen auch das Weiterleiten von Botschaften. Dafür verwendet UML den Stereotyp <<broadcast>> und OML eine an die *Aggregation*- und *Containment*-Beziehung angelehnte Notation. Die vielen dabei verwendeten Spezialpfeile sind jedoch nur schwer zu merken.

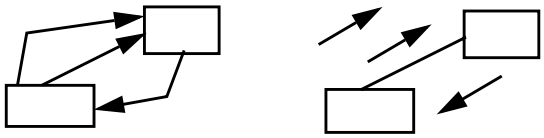

Abb. 27: Verschiedene Darstellungen des Botschaftenflusses

UML und OML verbinden Botschaftentyp und -pfad zu einem Notationselement. Abb. 27 zeigt zu diesem Vorgehen eine Alternative, wie sie in der Booch-Notation verwendet wurde und auch in UML stellenweise verwendet wird. Hierbei sind der Kommunikationspfad und die Botschaft getrennt. Dies erlaubt die mehrfache Benutzung eines Kommunikationspfads für verschiedene Botschaften mit unterschiedlicher Synchronisationsart. Dadurch kann die Übersichtlichkeit gegenüber der ersten Alternative erhöht werden. Der Nachteil dieser Darstellung ist jedoch eine Reduzierung der Zusammengehörigkeit.

6.1.2.5.5 Aggregation

UML verwendet für Aggregation die Raute. OML benutzt einen Kreis mit eingezeichnetem Plus. Obwohl die Symbolik Ansichtsache ist, veranschaulicht das Plus bei OML unseres Erachtens die Aggregation besser.

6.1.2.6 Muster

OML und UML unterstützen Entwurfsmuster. Beide Modellierungssprachen definieren aber nicht exakt, was sie unter einem Muster verstehen. OML benutzt für Muster *Cluster* und veranschaulicht Muster in *Cluster*-Diagrammen. Das allgemeine Muster wird dabei durch einen *Cluster* beschrieben und konkrete Realisierungen durch *Cluster Instances*. Die einzelnen Komponenten im Entwurfsmuster werden dabei als Rollen modelliert (siehe Abb. 22, S. 70 und Abb. 28).

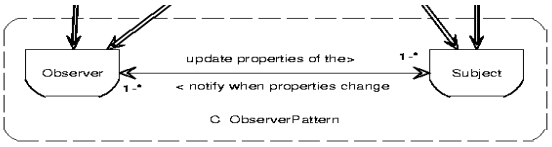

Abb. 28: Entwurfsmuster in OML ([FiHe96], S. 48)

UML benutzt das Sprachkonstrukt *Collaboration* für Muster. *Collaboration*-Diagramme können die Existenz (Abb. 29) und die Implementierung von Entwurfsmustern zeigen.

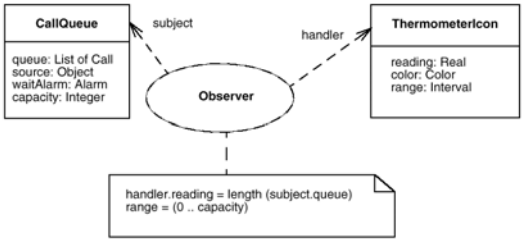

Abb. 29: Entwurfsmuster in UML ([Rat97i], S. 75)

Die Veranschaulichung von Mustern kann auch mit einfachen Klassen- und Objektdiagrammen erfolgen. Diese sind weniger abstrakt und zeigen meist zusätzlich eine vorgeschlagene Beispiellösung (Implementierung). Die Unterstützung von Mustern ist positiv zu bewerten. Allerdings benutzen UML und OML nur ihre Gruppierungsmechanismen für Muster, so daß keine weiteren semantischen Festlegungen erfolgen können. Auch ist die Kombination von textuellen und graphischen Beschreibungen, wie sie [GHJV95] oder [BMRS96] verwenden, wesentlich aussagekräftiger.

6.1.2.7 Sonderzeichen bei Stereotypen

Stereotypes werden durch Sonderzeichen hervorgehoben: "matched guillemets, which are the quotation mark symbols used in French and certain other languages, as for example: «foo». (Note that a guillemet looks like a double angle-bracket but it is a single character in most extended fonts. Double angle-brackets may be used as a substitute by the typographically challenged.)" ([Rat97i], S. 16). Für eine internationale Sprache ist es unverständlich, wie Zeichen verwendet werden können, die nicht in allen Sprachen gebräuchlich sind und insbesondere nicht primär auf Computertastaturen zu finden sind. Klammern oder Anführungszeichen wären die bessere Wahl gewesen.

6.1.2.8 Attribute und Operationen

UML verwendet zur Unterscheidung der Sichtbarkeit von Attributen und Operationen vorangestellte Marken ([Rat97i], S. 32 ff.). Dies sind:

+	Sichtbarkeit public
#	Sichtbarkeit protected
-	Sichtbarkeit private
/	abstract

Klasseneigenschaften werden unterstrichen. Die einzelnen Markierungen sind schwer zu merken. Günstiger ist hier die Übernahme der Technik aus den Programmiersprachen, die die einzelnen Sichtbarkeitsbereiche durch vorangestellte Schlüsselwörter definieren. Das Unterstreichen von Klasseneigenschaften konkurriert mit dem Unterstreichen von Objekten!

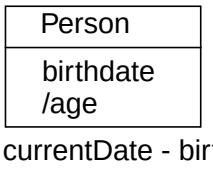

Abb. 30: Abstrakte Attribute in UML ([Rat97i], S. 60)

UML unterstützt auch die Angabe sogenannter abstrakter Attribute (Abb. 30). In diesem Fall ist die Kritik von OML berechtigt, daß UML daten- und attributorientiert ist, da "age" einfach als Abfrageoperation ausgedrückt werden kann.

OML erlaubt die Darstellung geerbter Eigenschaften ([FiHe96], S. 149). Es gibt jedoch keine Möglichkeiten, in einer Klasse festzustellen, welche Eigenschaft geerbt ist und welche nicht. OML kennzeichnet zwar selbstdefinierte Methoden mit *{def}*, diese Kennzeichnung wird jedoch nur optional angegeben. Nach dieser Regelung müßten alle Eigenschaften ohne *{def}* einer Klasse als geerbt angesehen werden. Günstiger wäre hier der entgegengesetzte Ansatz gewesen, indem geerbte Methoden explizit gekennzeichnet werden. Ein zusätzliches Problem sind überschriebene Operationen. Auch die Frage, was beim Löschen oder bei Verlust von Eigenschaften durch Änderung der Hierarchie passiert, bleibt unbeantwortet. Prinzipiell gibt es ein Zuordnungsproblem der Eigenschaften, da nicht feststeht, ob die Klasse oder die Superklasse die Eigenschaft definiert. In einer Klasse sollten aus diesem Grund möglichst nur in ihr definierte Eigenschaften veranschaulicht werden.

6.1.3 Diagramme

UML und OML stellen für die derzeit akzeptierten Sichtweisen objektorientierter Entwurfsmethoden eine Vielzahl von Diagrammen zur Verfügung ((Tabelle 4, S. 47 und Tabelle 5, S. 52). Bei den Diagrammen gibt es konzeptuelle Unterschiede. Jede hat aber im wesentlichen ein entsprechendes Äquivalent. Einige dieser Diagramme wie Zustandsdiagramme, Use-Case-Diagramme oder Systemstrukturdiagramme sind nicht objektorientiert. Sie können losgekoppelt vom objektorientierten Programmierparadigma verwendet werden. Dies zeigt, daß die Diagramme von UML und OML verschiedene Paradigmen vereinen. Meist handelt es sich um Diagramme, die aus bekannten älteren Notationen abgewandelt und übernommen wurden.

In beiden Notationen fehlen jedoch umfassende Richtlinien, wann welches Diagramm eingesetzt werden sollte. Auch die Syntax der einzelnen Diagramme ist nicht exakt definiert. Die Vielzahl der verschiedenen Diagramme erschwert deren Anwendung zusätzlich.

Kritisch anzumerken ist die recht starke Überschneidung und hohe Komplexität einzelner Diagramme. So überschneidet sich der Verwendungszweck einiger Diagramme (*Cluster*-Diagramm, Vererbungsdiagramm in OML; Objektdiagramm, Klassendiagramm, *Collaboration*-Diagramm in UML) recht stark, so daß die Wahl der definierten Diagrammarten nicht optimal erscheint. Gleichzeitig sind einige Diagramme sehr umfangreich, was sich in einer großen Anzahl von Diagrammelementen niederschlägt. Die Diagramme von UML und OML weisen daher in ihrer Gesamtheit einen gewissen Grad an Redundanz auf, und ihre Anwendung ist nicht immer einfach und intuitiv.

6.1.3.1 Systemdiagramme

UML und OML enthalten beide Diagrammformen zur Modellierung der Systemarchitektur. So gibt es in OML *Context*-Diagramme, *Layer*-Diagramme, *Configuration*-Diagramme und *Cluster*-Diagramme. Die Abgrenzung zwischen diesen Diagrammen ist nicht einfach. So können *Cluster*-Diagramme auch die Aufgaben für *Layer*-Diagramme und *Configuration*-Diagramme übernehmen, da sie alle deren Diagrammelemente auch enthalten ([FiHe96], S. 192 ff.). Insbesondere *Layer*- und *Configuration*-Diagramme scheinen sehr ähnliche Aufgaben zu haben. Beide beschreiben die externe Sicht auf *Cluster*.

Configuration-Diagramme unterstützen zusätzlich noch *Cluster instances*. Es wäre einfacher, beide Diagrammtypen zu vereinen. Systemdiagramme wie *Context*-Diagramme werden häufig in der Anforderungsanalyse und beim Systementwurf verwendet. Ihre Integration in eine objektorientierte Modellierungssprache birgt jedoch die Gefahr, die objektorientierte Modellierungssprache mit weiteren Konzepten zu überladen. Die *Layer*-Diagramme der OML sollten besser Systemarchitekturdiagramme heißen, da sie nicht nur die Schichtenstruktur, sondern die gesamte Teilsystemstruktur veranschaulichen. *Cluster*-Diagramme erlauben die Beschreibung der statischen Struktur eines *Cluster*. Da *Cluster* exemplarische Objekte und *Subcluster* enthalten können, ist mit ihnen immer ein dynamischer Aspekt verbunden. Die Verknüpfung von Klassen- und Objektebene ist in diesem Zusammenhang als kritisch zu betrachten. *Deployment*-Diagramme haben in OML und UML dieselben Aufgaben. UML definiert im wesentlichen keine neuen Diagramme für den Systementwurf. Use Cases können zur Darstellung der Systemfunktionalität und Systemabgrenzung verwendet werden. Klassendiagramme können in einer erweiterten Form Pakete darstellen und erlauben so die Darstellung der logischen Struktur. Zusammenfassend läßt sich sagen, daß OML umfangreichere Diagramme für die frühen Phasen der Software-Entwicklung bereitstellt als UML.

6.1.3.2 Klassen und Objektdiagramme

OML und UML unterstützen Diagramme zur Veranschaulichung von Klassen und Objekten. UML, aber auch OML vermischen die Klassen- und Objektdimension. Nach Booch läßt sich ein objektorientiertes System in seiner kanonischen Form durch eine Klassenhierarchie und eine Menge interagierender Objekte beschreiben ([Boo94], S. 14). Die Trennung zwischen Objekt- und Klassenebene erleichtert die Festlegung statischer Eigenschaften der Objekte und ihr dynamisches Zusammenspiel innerhalb des Systems. Die Dualität zwischen beiden ist gesichert, da die Objekte Instanzen der Klassen aus der Hierarchie sein müssen. Die einzigen Beziehungen zwischen Objekten und Klassen sind Instanz- und Zugriffsbeziehungen auf Klasseneigenschaften. Instanzbeziehungen brauchen nicht veranschaulicht zu werden, da die Klasse eines Objektes direkt angegeben werden kann. Beim Zugriff auf Klasseneigenschaften können Klassen als Objekte behandelt werden. Eine Trennung zwischen Klassen- und Objektebene ist daher empfehlenswert. Bei der Vermischung von Objekt- und Klassenebene ergibt sich ein zusätzliches Problem durch die ähnlichen Diagrammelemente für Exemplare und Typen, so daß eine schnelle verwechslungsfreie Unterscheidung nicht immer leicht fällt.

Durch die Typ/Exemplar-Dichotomie ist es notwendig, weitere Diagramme für die anderen Typarten bereitzustellen. So sollte es auch separate Diagramme für die Darstellung von Systemtypen und Systemen geben. Die Typ/Exemplar-Dichotomie sollte sich somit in den Diagrammen widerspiegeln, indem Typdiagramme und Exemplardiagramme für die jeweiligen Konzepte unterstützt werden. Eine Vermischung von Typ- und Exemplarebene sollte vermieden werden.

6.1.3.2.1 Vererbungsdiagramme

OML unterstützt Vererbungsdiagramme ([FiHe96], S. 145 ff.). Diese Diagramme sind hauptsächlich zur Darstellung von Vererbungsbeziehungen vorgesehen: "Whereas documenting delegation (via association/linkage, aggregation, and containment) and specialization/inheritance relationships on the same diagram can often be useful, such diagrams are often cluttered and confusing because they mix different abstractions with different scopes (e.g., cluster vs. inheritance branch). Use cluster diagrams to document delegation and inheritance diagrams to document inheritance" ([FiHe96], S. 146). Diese Argumentation rechtfertigt nicht die Einführung eines zusätzlichen Diagrammtyps! Ähnlich wie schon bei *Layer*- und *Configuration*-Diagrammen ist die Frage berechtigt, ob nicht eine allgemeinere Diagrammart günstiger wäre. Spezielle Verwendungen könnten durch zusätzliche Hinweise und Empfehlungen erklärt werden. Häufig reicht die Betrachtung der Vererbungshierarchie nicht aus, da auch die anderen Beziehungen wichtig sein können. Dies ist zum Beispiel der Fall, wenn Vererbung durch Delegation oder Komposition ersetzt wird. Auch zur Darstellung der Klassenstruktur von Mustern werden zusätzliche Beziehungen benötigt. Entwurfsmuster kombinieren im allgemeinen Vererbung und Komposition, um eine flexible Gesamtstruktur zu bilden. Außerdem bieten Werkzeuge Filtermöglichkeiten

zum Ausblenden von Beziehungen. Im wesentlichen können alle Informationen, die in Vererbungsdiagrammen dargestellt werden, auch mit *Cluster*-Diagrammen veranschaulicht werden. Zusätzliche Vererbungsdiagramme sind damit überflüssig.

6.1.3.2.2 *Collaboration*-Diagramme

Abb. 31 zeigt ein *Collaboration Diagram* aus OML. Die Darstellung ist recht unübersichtlich und schwer verständlich. Weiterhin ist fraglich, welchen Vorteil man durch diese Darstellung gewinnt. Es ist ausreichend, entweder die Struktur der Klasse oder exemplarisch die Struktur eines Objekts zu veranschaulichen. Die Darstellung in Abb. 31 ist daher durch ein großes Maß an Redundanz gekennzeichnet.

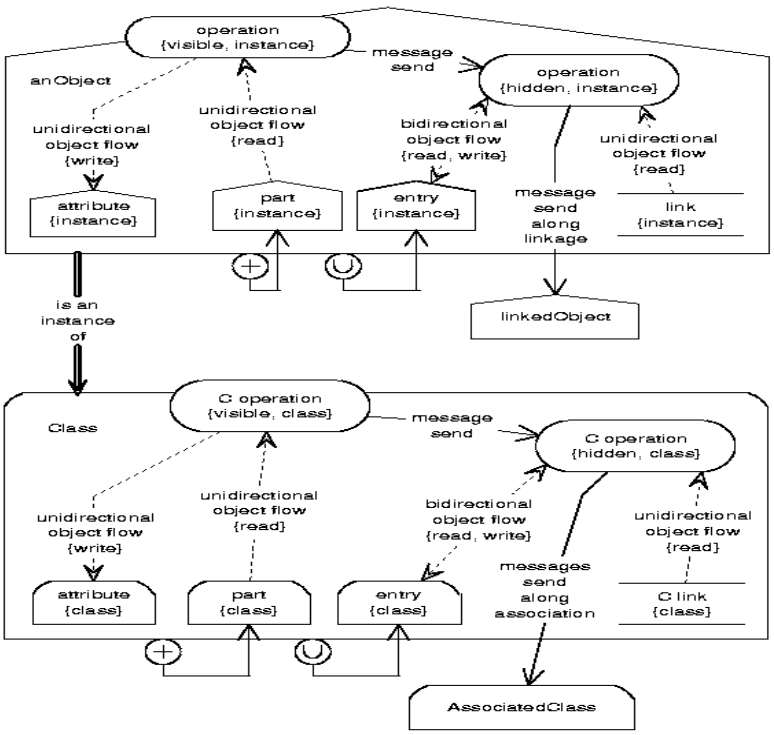

Figure 96: Icons for Characteristics on Internal Collaboration Diagrams

Abb. 31: Typ/Exemplar Dichotomie ([FiHe96], S. 167)

UML definiert das Sprachkonstrukt *Collaboration* (vgl. Abb. 29, S. 74), mit dem Interaktionen, Mechanismen und Muster beschrieben werden können. Es wird jedoch nicht erklärt, in welchen Diagrammen *Collaborations* veranschaulicht werden. *Collaboration*-Diagramme entsprechen im wesentlichen Objektdiagrammen und beschreiben die Objektebene.

6.1.3.2.3 *Static-Structure*-Diagramme

UML definiert Klassendiagramme durch: "A class diagram is a graph of modeling elements shown on a two-dimensional surface. (Note that a ’class’ diagram may also contain types, packages, relationships, and even instances, such as objects and links. Perhaps a better name would be ’static structural diagram’ but ’class’ diagram sounds better)" ([Rat97i], S. 18). Der Begriff *Class*-Diagramm assoziiert eine ganz andere Bedeutung als *Static-Structure*-Diagramm! Einen Diagrammnamen zu wählen, nur weil er besser klingt, ist keine akzeptable Begründung! Klassendiagramme veranschaulichen die Struktur der Klassenhierarchie. Strukturdiagramme dienen zur Beschreibung der Systemstruktur. Die Vermischung beider Sichtweisen ist problematisch, da die Klassen statische Informationen, Teilsysteme aber dynamische Informationen ausdrücken.

Im Notation Guide gibt es eine weitere Erklärung von Klassendiagramm: "A class diagram is a collection of (static) declarative model elements, such as classes, types, and their relationships, connected as a graph to each other and to their contents. Class diagrams may be organized into packages either with their underlying models or as separate packages that build upon the underlying model packages" ([Rat97i], S. 18). Beide Aussagen stimmen nicht überein! Die untere assoziiert die Darstellung von Klassen und die obere die von beliebigen strukturellen Aspekten. Eine Trennung in Diagramme für die Systemstruktur und die Klassenstruktur ist empfehlenswert. Klassendiagramme können dabei Gruppierungsinformationen für Klassen enthalten.

Widersprüchliche Aussagen findet man in UML auch zu Objektdiagrammen: "An object diagram is a graph of instances. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. A dynamic object diagram shows the detailed state of a system over some period of time, including the changes that occur over time; dynamic object diagrams are manifested as collaboration diagrams" und weiter: "There is no need for tools to support a separate format for object diagrams. Class diagrams can contain objects, so a class diagram with objects and no classes is an ’object diagram.’ Collaboration diagrams contain objects, so a collaboration diagram with no messages is an ’object diagram’" ([Rat97i], S. 18). Wenn Objektdiagramme wirklich überflüssig sind, dann sollte vollständig auf sie verzichtet werden!

Für Klassen- und Objektdiagramme ist eine Präzisierung in UML notwendig. Diese umfaßt die Aufgaben und die erlaubten Darstellungen dieser Diagramme.

6.1.3.3 Interaktionsdiagramme

OML und UML verfügen beide über Sequenzdiagramme zur zeitlichen Darstellung des Objektflusses. In OML existieren keine Hinweise, ob die aktive Zeit eines Objektes[1] dargestellt werden kann. Schleifen, Bedingungen und parallele Abläufe können durch *Guard*s dargestellt werden (Abb. 32). Diese Darstellung führt jedoch zu einem Verlust der absoluten Zeitinformation.

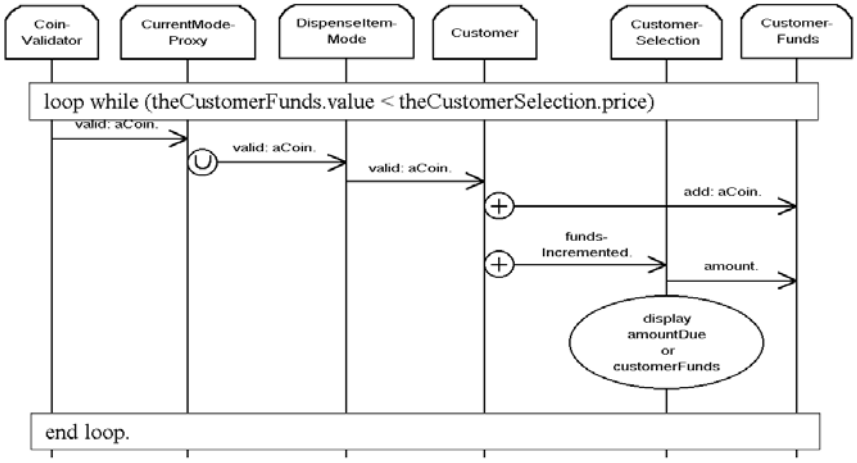

Abb. 32: Interaktionsdiagramm mit Guards ([FiHe96], S. 175)

Die UML definiert Sequenzdiagramme mit vielen Erweiterungen. So gibt es Kontrollstrukturen, das Aufspalten der Objektzeitachse und die Veranschaulichung der aktiven Zeit eines Objektes. Abb. 33 zeigt ein Interaktionsdiagramm in UML-Notation. Aus diesem Diagramm ist nicht eindeutig ersichtlich, welche Bedeutung das Aufspalten der Zeitachse eines Objekts hat. Dient es zum Darstellen von Alternativen oder Parallelität? Aber auch aus zeichentechnischer Sicht ist dieses Aufspalten problema-

1. UML benutzt dafür einen verstärkten Balken auf der Zeitachse des Objekts.

tisch. Das Aufspalten entspricht einer zusätzlichen Z-Dimension in einer zweidimensionalen Darstellung! Auch für Werkzeuge bildet diese Darstellung Probleme.

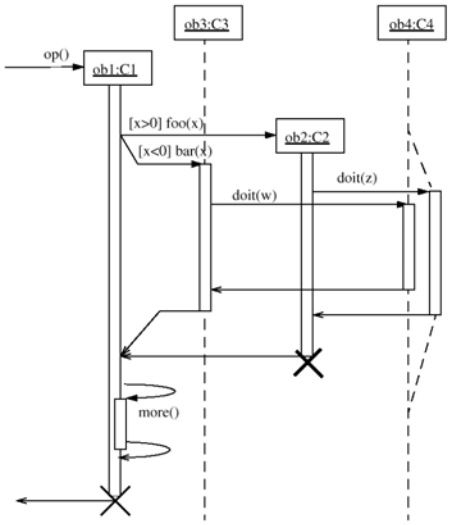

Abb. 33: Interaktionsdiagramm in UML ([Rat97i], S. 68)

Die Sequenzdiagramme in UML sind ein Beispiel, wie aus recht einfachen und überschaubaren Diagrammen durch Hinzunahme zusätzlicher Kontrollstrukturen mächtige, aber auch unübersichtliche Diagramme entstanden sind. Es muß die Frage gestellt werden, ob Sequenzdiagramme zum Veranschaulichen eines komplexen zeitlichen Objektflusses mit Rekursion, Kontrollstrukturen, Parallelität und Synchronisation überhaupt geeignet sind. Die Interaktionsdiagramme von OML sind weniger kompliziert als die in UML. Sie scheinen aus diesem Grund intuitiver zu sein. Die Sequenzdiagramme der UML dagegen sind ein krasses Beispiel für zu hohe Komplexität.

6.1.3.4 Szenariodiagramme

UML und OML definieren *Use-Case*-Diagramme auf ähnliche Weise. Szenarien und *Use Cases* gehen auf Jacobson [Jac92] zurück und sind in beiden Modellierungssprachen enthalten. Die Begriffe Szenario, Mechanismus und *Use Case* können im wesentlichen synonym verwendet werden. *Use Cases* haben primär funktionalen Charakter und kapseln eine Menge von Interaktionen. Abb. 34 zeigt zwei Szenariodiagramme der OML, die sehr stark an prozeßorientierte Darstellungen erinnern. Erklärungsbedürftig ist die Verwendung eines Szenarios oder Mechanismus als Schnittstelle eines *Cluster*. Wie soll diese Tatsache in der späteren Implementierung ausgedrückt werden? Sinnvoller ist es, die zugrundeliegenden Dienste des *Cluster* als Schnittstelle zu verwenden und das Szenario oder den Mechanismus als Wechselwirkung mit dem *Cluster* zu interpretieren. Der stark funktionale und prozeßorientierte Charakter von Use Cases erfordert zusätzlichen Aufwand bei der Integration in die objektorientierte Modellierungssprache[1]. UML und OML beantworten zum Beispiel keine Fragen, wie *Use Cases* auf Klassen und Objekte abzubilden sind.

1. Vgl. dazu [Ber97].

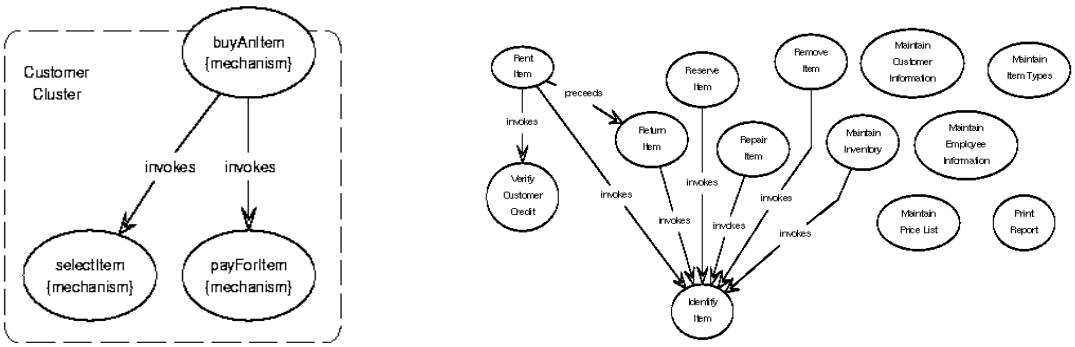

Abb. 34: Prozeßorientierte Darstellungen in OML ([FiHe96], S. 54 und 55)

6.1.3.5 Zustandsdiagramme

OML und UML verwenden Zustandsdiagramme zur Beschreibung des Zustandsverhaltens von Klassen. Beide Notationen geben für die Zustandsdiagramme keine formale Syntax und Semantik an. Die Zustandsdiagramme in OML sind nicht so komplex wie die in UML. Eine interessante Erweiterung in OML ist die Interpretation von Unterzuständen als Teil der Schnittstelle eines Oberzustands: "Attaching the icon of a substate to the boundary of the icon of the superstate is an intuitive way of showing that the substate is part of the interface of the superstate and therefore the state machine can transition from a state external to the superstate to the exported substate" ([FiHe96], S. 184). Diese Festlegung erlaubt eine Informationskapselung. Sie stimmt jedoch nicht mit der üblichen Definition geschachtelter Zustände (Harel) überein. Geschachtelte Zustände werden hauptsächlich verwendet, um die Anzahl von Zuständen und Übergängen einzuschränken. In OML können nur nach außen sichtbare Unterzustände mit externen Zuständen direkt verbunden werden. Es ist jedoch zu berücksichtigen, daß Unterzustände im allgemeinen implizite Übergänge mit externen Zuständen haben (Abb. 35). Bei einer Projektion auf die flache Darstellung verliert man daher die Kapselungsinformationen.

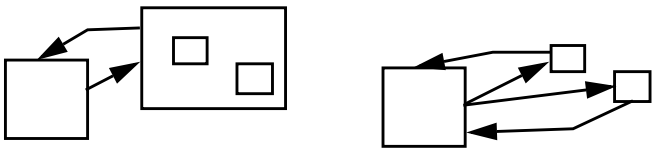

Abb. 35: Geschachtelte Zustände mit Zustandsübergängen

Die Zustandsdiagramme in UML basieren auf den Diagrammen von Harel und OMT. Sie sind recht ausführlich beschrieben ([Rat97i], S. 89 ff.). Aber auch UML gibt keine formalen Definitionen an. Die Zustandsdiagramme in UML besitzen komplizierte Regeln für die Schachtelung von Symbolen, die Parallelausführung und die Verknüpfung von Zuständen mit Operationen und Sonderregeln, wann diese Operationen nicht ausgeführt werden dürfen. Wie schon bei Interaktionsdiagrammen ist auch hier die Frage, ob es nicht adäquatere Beschreibungsmöglichkeiten gibt. Auf alle Fälle wird eine formale Definition der Zustandsautomaten benötigt.

Ein Schwachpunkt in beiden Darstellungen ist die fehlende Berücksichtigung von Vererbung. Allerdings enthält UML Version 1.1 nun ergänzende Aussagen für die Verfeinerung von Zuständen. Ist eine Klasse *B* eine Unterklasse einer anderen Klasse *A*, dann müssen in den Zustandsdiagrammen von *A* auch die Zustände von *B* berücksichtigt werden. Außerdem muß es möglich sein, das Zustandsverhalten von *B* auf der Basis von *A* durch Erweiterung zu spezifizieren. Es ist daher eine wesentlich engere Integration von Zustandsautomat und Objektorientierung notwendig. Diese Integration benötigt eine klare semantische Definition von Vererbung für Klassen, die auch die operationale Semantik der

Klasse umfaßt. UML und OML unterstützen dies nicht oder nur unzureichend. Die Integration der Zustandsdiagramme mit dem Klassenmodell ist nicht vollständig elaboriert.

6.1.3.6 Diagrammelemente

OML enthält einen guten Anhang, in dem zu jedem Diagramm die erlaubten Diagrammelemente aufgelistet sind ([FiHe96], S. 192 ff.). Diese Zuordnung ist übersichtlich und gibt wichtige Informationen über häufig und weniger häufig genutzte Elemente in den Diagrammen. Gleichzeitig zeigt diese Übersicht, daß stellenweise sehr viele Diagrammelemente für eine Diagrammart erlaubt sind (*Cluster*-Diagramm: 20 Symbole, 12 Beziehungsarten). UML stellt im Notation Guide die Diagrammelemente für die Sprachmittel vor. Allerdings fehlt eine tabellarische Referenz wie bei OML.

6.1.3.6.1 Geschachtelte Diagrammelemente

OML benutzt intensiv geschachtelte Diagrammelemente (Abb. 10, S. 63; Abb. 23, S. 71 und Abb. 31, S. 77). Die Verwendung geschachtelter Diagrammelemente beruht auf den älteren Notationen von Odell, Firesmith und Henderson-Sellers. Allerdings leidet durch diese Darstellung häufig die Übersichtlichkeit und Verständlichkeit, da die korrekte Verwendung der geschachtelten Sprachmittel nicht geklärt ist. Problematisch ist zudem die Festlegung der einzelnen Syntaxregeln zur Verknüpfung der Elemente. Geschachtelte Diagrammelemente offenbaren häufig Implementierungsdetails. Auch UML benutzt geschachtelte Diagrammelemente. Allerdings wird diese Darstellungsform sparsamer eingesetzt.

6.1.3.6.2 Duplikate von Diagrammelementen

In den Diagrammen zur Veranschaulichung des Metamodells der UML treten Symbole mit dem gleichen Bezeichner mehrmals auf. Abb. 36 zeigt das mehrmalige Auftreten des Klassensymbols Element. Veranschaulichen diese beiden Symbole eine Klasse oder zwei verschiedene Klassen? Für die einzelnen Diagrammarten muß die Bedeutung des mehrfachen Auftretens von einem Diagrammelement in einem Diagramm geklärt sein. Nur dies sichert die korrekte Zuordnung von Modellierungselement und graphischer Repräsentation. Auch OML enthält keine Angaben zu doppelten Symbolen in Diagrammen.

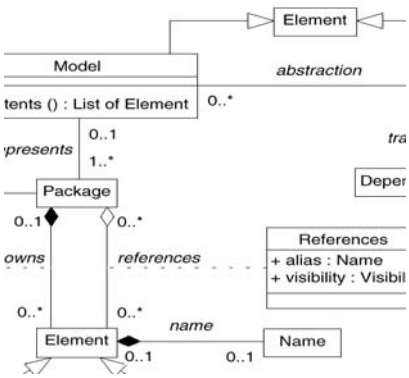

Abb. 36: Mehrfaches Auftreten von Symbolen ([Rat97c], S. 4)

6.1.4 Integration

Die Integration der einzelnen Sprachmittel ist in UML und OML nicht vollständig erklärt. Eines der Hindernisse für eine umfangreiche Integration der einzelnen Teilsichten und Modelle ist die fehlende exakte Formulierung der abstrakten Semantik. In vielen Fällen kann eine Integration der Teilsichten nur über die Bedeutung der einzelnen Sprachelemente erfolgen. Wenn die Semantik dieser Sprachmittel nicht präzise beschrieben ist, dann ist es unmöglich, diese Sprachmittel als Verbindungsglieder für die Integration zu benutzen. Zum einen könnten diese Sprachmittel in der jeweiligen Teilsicht differen-

zierte Bedeutungen haben. Zum anderen könnte eines dieser Sprachmittel aber auch Grundlage für die Definition eines Sprachmittels in einer anderen Teilsicht sein.

So offenbart gerade die Anbindung von Konzepten wie Zustandsautomaten, Subsysteme oder Szenarien an das Objektmodell Lücken. In Zustandsautomaten können bei Zustandsübergängen Botschaftenbezeichner, externe Signale oder einfache Zeichenketten verwendet werden. Für die Integration und Überprüfung mit Klassen ist eine eindeutige Festlegung notwendig, die auch Vererbung berücksichtigt. Da UML und OML Vererbung bezüglich ihrer dynamischen Semantik im Objektmodell nicht präzise formulieren, ist es geradezu unmöglich, auf dieser Basis Vererbung bei Zustandsautomaten einzuführen. Auch die Integration zwischen der Klassen- und Objektebene ist nicht ausformuliert. Auf Objektebene dürften eigentlich nur Dinge veranschaulicht werden, die durch die Klassenebene abgedeckt werden. Die Sprache muß für diesen Fall Bedingungen enthalten, die die Beziehungen zwischen Klassen- und Objektebene vereinbaren.

Zusammenfassend läßt sich sagen, daß die Integration der verschiedenen Teilsichten nicht zufriedenstellend ist. Beide Modellierungssprachen vermitteln eher den Eindruck, es handele sich um eine Menge separater Teilsprachen als um eine wohlintegrierte Gesamtsprache. Die wesentlichen Ursachen für diese Situation sind in der unpräzisen Beschreibung der dynamischen Semantik der Sprachmittel zu suchen, die eine hohe Integration der einzelnen Teilsichten ausschließt.

6.1.5 Dokumentation

Die Dokumentation objektorientierter Modellierungssprachen unterscheidet sich nicht wesentlich von der anderer Sprachen. Sie sollte die Sprachdefinition enthalten, die die Syntax und Semantik der Sprache festlegt. Zusätzlich kann sie noch eine Anwenderdokumentation enthalten.

6.1.5.1 Gesamteindruck

Beide Modellierungssprachen werden durch umfangreiche Dokumentationen beschrieben. Sie benutzen in der Dokumentation die eigene Notation als Metanotation. UML verwendet dabei nur die UML-Notation. OML benutzt einen nicht so rigorosen Ansatz, indem eine Behelfsnotation für noch unbekannte Notationselemente benutzt wird. Jeder dieser Ansätze hat Vor- und Nachteile. Zum Verständnis von UML sind eigentlich schon Kenntnisse in UML erforderlich. Durch den Ausschluß zusätzlicher Behelfsnotationen ist jedoch die Einheitlichkeit der Darstellung gewahrt. OML versucht die schrittweise Einführung und Benutzung der Diagrammelemente. Dabei werden jedoch ungeklärte Symbole und Beziehungen verwendet. So verwendet OML in den Diagrammen für das Metamodell einen Mix aus COMN und weiteren Symbolen (Rechtecken), deren Semantik nicht erklärt ist. In den einzelnen Diagrammen werden bekannte Konzepte mit OML-Notation und unbekannte Konzepte mit Behelfssymbolen dargestellt. Jedoch existiert dadurch keine einheitliche Notation des Metamodells. Dies ist beim späteren Nachschlagen nachteilig. Auffällig sind auch die zwei verschiedenen Dokumentationsvorgehensweisen, die UML und OML verwenden. UML stellt Notation und Metamodell in zwei separaten Dokumenten vor. Allerdings enthalten diese beiden Dokumente redundante Informationen. OML benutzt nur ein Dokument, in dem die Notationsabschnitte und das Metamodell vermischt dargestellt sind. Dies ist durch die unterschiedlichen Ansätze bei der Dokumentation bedingt. Zusammenfassend erscheint die Strukturierung der Dokumentation von UML günstiger als die von OML, da UML abstrakte Syntax und Semantik von der konkreten Syntax (Notation) trennt.

In beiden Dokumentationen fehlen umfangreiche Querverweise und Indizes, um die zu einem Sprachkonzept relevanten Textstellen zu finden. Bei UML sind zusätzliche dokumentübergreifende Querverweise nötig, da die Informationen zu einem Sprachelement über die verschiedenen Dokumente verteilt sind. Querverweise sind eine Maßnahme, die die Les- und Anwendbarkeit erhöhen können.

Beide Dokumentationen benutzen einen informalen Ansatz für die Sprachbeschreibung[1]. Dies führt dazu, daß die Bedeutung der einzelnen Sprachmittel nicht exakt beschrieben ist. Zusätzlich wird die Verständlichkeit durch offene Fragen und widersprüchliche Aussagen erschwert. Die Präzisierung der Sprachbeschreibung ist für beide Sprachen dringend erforderlich. Die Verwendung von OCL in UML Version 1.1 und die starke Überarbeitung des *Sematics Guide* haben zu einer wesentlichen Verbesserung der Verständlichkeit und Exaktheit geführt. Allerdings ist die dynamische Semantik immer noch unzureichend beschrieben.

6.1.5.2 Sprachspezifikation

Kritisch ist das Fehlen einer separaten Sprachbeschreibung zu sehen, die die Semantik und Syntax der Sprache an einer Stelle beschreibt. UML enthält semantische Informationen zum Beispiel im *Notation Guide*, im *Semantics Guide* und im Glossar. Dabei kommt es zu widersprüchlichen Festlegungen in den einzelnen Dokumenten. So heißt es im *Semantics Guide*: "Activity diagram is a subtype of diagram" ([Rat97c], S. 92). Nach den Festlegungen im Glossar gilt aber: "Activity diagram is a special case of a state diagram" [Rat97d], S. 2). Welche dieser Aussagen ist nun verbindlich? Die Definitionen für die einzelnen Sprachmittel hätten an nur einer Stelle vereinbart werden dürfen. In den anderen Dokumenten müßte dann nur noch auf die entsprechende Stelle verwiesen werden. Auch die Metamodelle beider Sprachen sind kein Ersatz für die fehlenden Sprachspezifikationen, da sie Sprachcharakteristika und konkrete Metamodellrealisierung nicht trennen.

6.1.5.3 Metamodell

Sowohl die OML als auch die UML werden durch einen Metamodell-Ansatz unter Anwendung objektorientierter Technologien beschrieben. Metamodelle sind jedoch keine Beschreibung der Konzepte einer Sprache. Sie enthalten zu viele technische Hilfskonstrukte. Sie sind eher mit Grammatiken vergleichbar, die eine Definition für die Sprache bilden. Das Metamodell schlägt im wesentlichen einen Entwurf und eine Implementierung für die Sprache vor. Jedes Metamodell muß überprüft werden, ob es auch die gewünschte Sprache beschreibt. Durch das Fehlen einer Sprachspezifikation fehlen die Voraussetzungen für eine Überprüfung der Metamodelle bezüglich den Anforderungen und Konzepten der Modellierungssprache. Gerade beim Übergang der Metamodelle von UML Version 1.0 zur UML Version 1.1 war es aufgrund der fehlenden Sprachspezifikation nicht möglich zu unterscheiden, ob Änderungen am Metamodell nur interne Auswirkung haben oder ob dadurch auch die betrachtete Sprache verändert wird. Ein objektorientiertes Metamodell erlaubt die Generalisierung über Sprachelementgemeinsamkeiten. Es beeinträchtigt aber den Überblick über ein Sprachelement, da alle Superklassen dieses Sprachelements mit untersucht werden müssen. Noch problematischer ist jedoch das Hinzufügen von Metaelementen, die in der Sprache keine Entsprechung haben. Allgemein muß die Frage gestellt werden, ob es ausreicht, eine Sprache durch ein Metamodell zu beschreiben oder ob es Aspekte gibt, die durch Metamodelle nicht adäquat beschrieben werden können.

Ein generelles Problem ist auch die Repräsentation beider Metamodelle. Die textuelle Beschreibung mit einigen Diagrammen reicht nicht aus, die Metamodelle zu beschreiben. Nur die vollständigen Metamodelle können hier Abhilfe schaffen. Nur sie erlauben es zu überprüfen, wie im Metamodell Dinge festgelegt werden oder ob Dinge fehlen. Das eigentliche Metamodell ist daher der in OML und UML verwendeten Variante vorzuziehen. Für die UML Version 1.1 ist das Metamodell nun im Dateiformat für *Rational Rose* verfügbar. Dies ermöglicht die ganzheitliche Untersuchung und Überprüfung des Modells. Negativ ist jedoch anzumerken, daß im Modell keine Entscheidungen dokumentiert sind.

Das Metamodell der OML ist nicht so ausgeprägt wie das der UML. Dies liegt an der anwendernäheren Beschreibung der OML, die nicht explizit ihr Metamodell festlegt. Das Metamodell von OML enthält Entscheidungen, denen nicht ohne weiteres zugestimmt werden kann. So definiert OML *Cluster Characteristic* und leitet anschließend Klassen, Typen und Rollen von dieser ab. Diese Entscheidung

1. UML Version 1.1. geht mit der Übernahme von OCL in Richtung einer formalen Spezifikation der Sprachmittel.

erfolgt nur, damit *Cluster* diese Komponenten umfassen können. Es ist eine reine Metamodellentscheidung. Weitere solche Metamodellentscheidungen sind das Definieren von *Cluster* auf der Basis von *Cluster Charakteristic*, um auch *Cluster* in *Cluster* schachteln zu können oder die Definition von *ObjectClass* auf der Basis von *Object*, damit Klassen spezielle Objekte sind.

Das Metamodell von UML ist sehr detailliert beschrieben. An vielen Stellen erinnert es jedoch an das Modell eines Werkzeugherstellers, das eine Realisierung der Sprache enthält. So spezifiziert UML die Diagramme im Metamodell[1]. Dies ist eine technische Entscheidung, um die Eigenschaften der einzelnen Diagramme festlegen zu können. Ein weiteres Problem ist die Verflechtung zwischen Meta- und Sprachebene bei UML durch die Beschreibung der UML mit Hilfe von UML. Dies führt zu einer Reihe von Problemen bei der Verwendung von Begriffen und Formulierungen wie, *eine Klasse ist eine Klasse* oder ein *Typ ist eine Klasse*, da keine abgegrenzte Metasprache existiert. Dieser Begriffsverwirrung hätte durch eine stärkere Abgrenzung zwischen Sprachkonzepten und Metamodell, in dem viele Sprachkonzepte als Klassen modelliert werden, vorgebeugt werden können. Auch die im Metamodell getroffenen Entscheidungen sind stellenweise nicht nachvollziehbar. So wird das Sprachkonstrukt *StateMachine* von *Behavior* abgeleitet [Rat97c], S. 67). Für die Sprache UML ist diese Entscheidung jedoch nicht von Bedeutung. Es handelt sich um eine metamodellinterne Entscheidung. Eine Erklärung, warum diese Modellierung erfolgt, findet sich leider nicht.

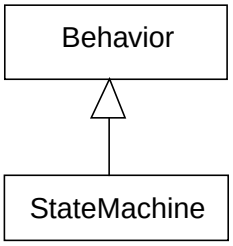

Abb. 37: StateMachine und Behavior ([Rat97c], S. 67)

Das Beispiel in Abb. 38 zeigt wiederum, daß Entscheidungen im Metamodell und objektorientierte Prinzipien nicht übereinstimmen müssen. UML definiert eine Klasse als Implementation eines Typs. In diesem Fall muß eine Klasse kein Typ sein. Im Diagramm ist Klasse von Typ abgeleitet. Eine Klasse ist dann aber auch ein Typ. Welche Klasse ist die Implementierung dieses Typs? Diese Frage bleibt unbeantwortet. Ein anderer Konflikt ergibt sich durch die Ableitung von *PrimitiveType* und *UseCase* von *Type*. *Type* ist ein Derivat von *GeneralizableElement*. Damit können auch primitive Typen in Generalisierungsbeziehungen verwendet werden. Dies ist ein Widerspruch zur Annahme, das primitive Typen nicht abgeleitet werden können. Ein weiteres Problem ist die Vermischung verschiedener Unterklassen von *Type* in Generalisierungsbeziehungen, falls keine Typisierungsregeln festgelegt sind. Ein *UseCase* könnte dann von einem *PrimitiveType* abgeleitet werden. Das Metamodell erfordert in diesem Punkt eine Präzisierung und Überarbeitung.

1. In Version 1.1 sind die Diagramme nicht mehr im Metamodell enthalten.

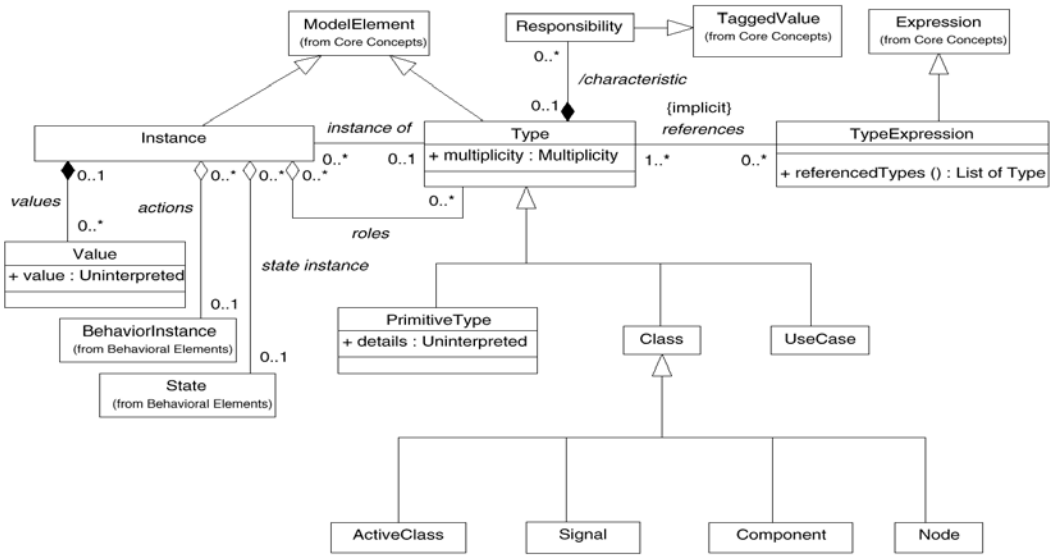

Abb. 38: Ausschnitt aus dem Metamodell der UML ([Rat97c], S. 24)

In Version 1.1 ist das Modell für Typen und Klassen überarbeitet worden. Zusätzlich enthält es die geforderten Regeln für die Typisierungsregeln bei der Generalisierung. Im UML Metamodell sind sprachliche Definitionen enthalten, die allenfalls im Kontext des Metamodells gültig, aber keinenfalls allgemeingültig sind: "An expression is a string. ... A string is a stream of text. ... A time is a string representing an absolute or relative moment in time and space" ([Rat97c], S. 21). Statt dessen hätte man sagen sollen, daß diese Konstrukte durch Zeichenketten ausgedrückt werden. Im UML Metamodell existiert kein Hinweis, ob Mehrfachvererbung im Metamodell angewendet wird. Es gibt aber die Festlegungen: "Type is a subtype of ModelElement" ([Rat97c], S. 26) und "Type is a subtype of GeneralizableElement" ([Rat97c], S. 36). Diese Modellierung setzt jedoch Mehrfachvererbung voraus.

6.1.5.4 Anwenderdokumentation

OML und UML enthalten keine umfangreiche Dokumentation für den Anwender. Jedoch sind bei beiden kleine Beispiele zur Erklärung der Notation und Sprachkonzepte in die Dokumentation integriert. Einige dieser Beispiele sind aus software-technischer Sicht fragwürdig. Abb. 39 zeigt ein Beispiel zur Verwendung von Aggregation in OML. Diese Implementierung von Polygon mit Liniensegmenten ist jedoch ungünstig und nicht optimal. Das Verwenden von Punkten dagegen sichert die Zusammengehörigkeit!

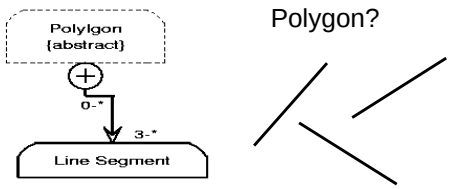

Abb. 39: Definition der Klasse Polygon ([FiHe96], S. 110)

Für UML und OML gilt daher, daß die anwenderorientierte Beschreibung der Sprache verbessert werden muß. Dies betrifft insbesondere die Bedeutung und Anwendung der Sprachkonstrukte aus Benutzersicht. Längerfristig werden sicher zusätzliche Lehrbücher und Tutorials für UML bzw. OML durch externe Autoren angeboten werden. Ein Beispiel hierfür ist [Bur97].

6.2 Anwendung des Beschreibungsrahmens auf UML und OML

Im folgenden soll der Bewertungsrahmen, der im Abschnitt 4.5, S. 41 vorgestellt wurde, auf UML und OML angewendet werden. Zur Verbesserung der Übersicht und Vergleichbarkeit beider Sprachen wird er in die Abschnitte Beschreibung, abstrakte Sprachkonzepte, konkrete Sprachmittel, Anwendung und Kriterien unterteilt.

6.2.1 Beschreibung

Die Beschreibung der Sprache ist eine der Dimensionen, die bei der Untersuchung berücksichtigt werden (vgl. auch Abschnitt 6.1.5, S. 82).

Tabelle 6: Bewertungsrahmen Beschreibung für UML

		Merkmal	Ausprägung	Bewertung
Beschreibung	Definition	Grammatik	für OCL (UML V. 1.1)	OCL ist eine formale Sprache zum Darstellen von Bedingungen, Invarianten sowie Vor- und Nachbedingungen. Sie ist seiteneffektfrei. Sie ist speziell an objektorientierte Belange angepaßt (self). Sie stellt aber keine vollständige Spezifikationssprache dar. Die Integration formaler Ansätze in UML ist positiv zu bewerten.
		Metamodell	objektorientiertes Metamodell in UML	Das Metamodell der UML beschreibt die UML auf einem recht hohen formalen Niveau (UML V. 1.1). Durch die Verwendung objektorientierter Konzepte sind im Metamodell jedoch eine Reihe von künstlichen Konzepten enthalten, die keine entsprechenden Sprachkonzepte der UML darstellen. Einige Umsetzungen der Sprachkonzepte im Metamodell zeigen die gedachte Implementierung und nicht die logische Bedeutung der Sprachelemente. (vgl. Abschnitt 6.1.5.3, S. 83)
	Dokumentation	Sprach-spezifikation	Die abstrakte Syntax und Semantik ist im Semantics Guide durch das Metamodell beschrieben. Die konkrete Syntax ist im Notation Guide beschrieben.	Die Trennung in Semantik und Notation Guide ist gut. Es fehlt jedoch eine Übersichtsdarstellung der abstrakten Sprachkonzepte der UML, da im Metamodell zusätzliche künstliche Konzepte enthalten sind und die Aussagen im Notation Guide zu oberflächlich sind. Bei der Darstellung der konkreten Syntax sind zusätzliche semantische Informationen enthalten, die jedoch nicht immer mit den Aussagen im Semantics Guide übereinstimmen. (vgl. Abschnitt 6.1.1, S. 55 und Abschnitt 6.2.3, S. 91)
		Anwender-dokumentation	keine	Die fehlende Anwenderdokumentation stellt kein großes Problem dar, da sie in absehbarer Zeit durch Tutorials und Lehrbücher ersetzt wird.

Tabelle 7: Bewertungsrahmen Beschreibung für OML

		Merkmal	Ausprägung	Bewertung
Beschreibung	Definition	Grammatik	keine	
		Metamodell	objektorientiertes Metamodell	Das Metamodell von OML ist nur in groben Ansätzen vorgestellt. Formale Darstellungen fehlen fast vollständig. Die Verwendung einer Behelfsnotation für unbekannte Sprachkonstrukte erschwert das gesamtheitliche Verständnis des Metamodells. (vgl. Abschnitt 6.1.5.3, S. 83)
	Dokumentation	Sprach-spezifikation	Referenzmanual	OML stellt abstrakte Syntax und Semantik und die konkrete Syntax in einem Dokument dar. Eine Trennung zwischen abstrakten und konkreten Teilen erfolgt nicht. Die Darstellung im Referenzmodell ist informal. Die Auflistung der Diagrammelemente für die einzelnen Diagramme im Anhang ist gut. (vgl. Abschnitt 6.1.1, S. 55 und Abschnitt 6.2.3, S. 91)
		Anwender-dokumentation	keine	Die fehlende Anwenderdokumentation stellt kein großes Problem dar, da sie in absehbarer Zeit durch Tutorials und Lehrbücher ersetzt wird.

6.2.2 Abstrakte Syntax und Semantik

Aus Abb. 1, S. 20 ist ersichtlich, daß die abstrakte Syntax und Semantik den eigentlichen Sprachkern darstellt. Ihrer Untersuchung kommt daher eine besondere Bedeutung zu. Da UML und OML als objektorientierte Modellierungssprachen sich im wesentlichen auf die gleiche abstrakte Syntax und Semantik beziehen, sollten die Gemeinsamkeiten überwiegen. Unterschiede werden vor allem bei speziellen Konstrukten auftreten.

Tabelle 8: Bewertungsrahmen abstrakte Syntax und Semantik für UML

		Merkmal	Ausprägung	Bewertung
	Aufbau	Grundkon-zeption	Teilsprachen, aber Versuch eines gemeinsamen Metamodells	Die Unterteilung ist historisch begründet. UML integriert verschiedene Teilsichten oder -sprachen, die aus älteren Modellierungssprachen übernommen worden sind. Mit dem gemeinsamen Metamodell soll die Integration zwischen den einzelnen Teilsprachen verbessert werden.

Tabelle 8: Bewertungsrahmen abstrakte Syntax und Semantik für UML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Sprache	Konzepte (abstrakte Syntax und Semantik)	Objektorientierte Konzepte	Klasse, Objekt, Vererbung, Polymorphismus	UML unterstützt die wichtigsten objektorientierten Konzepte. Im wesentlichen sind dies Konzepte, die schon die Notationen von OMT oder Booch enthalten. Diese Konzepte werden jedoch nicht vollständig definiert, sondern der Benutzer hat Interpretationsfreiräume. Die fehlende exakte Definition der Konzepte ist die Ursache für eine Reihe von Widersprüchen. Negativ ist zu bewerten, daß Standardkonzepte wie Klasse oder Vererbung beim Übergang von UML Version 1.0 zu Version 1.1 plötzlich eine völlig neue Interpretation bekommen, die nicht mit der bisherigen Praxis übereinstimmt. Dies betrifft auch das Verhältnis zwischen Objekt und Klasse. Spezielle Konzepte, wie vererbte Beziehungen oder Verfeinern von Kardinalitäten sind nicht berücksichtigt. (vgl. dazu Abschnitt 6.1.1.1.2, S. 56)
		Modularisierungskonzepte	*Package*, Teilsystem, Klasse, *Collaboration*, *Use Case*	Der grundlegende Modularisierungsmechanismus ist das *Package.* Es gibt eine Reihe weiterer Sprachkonzepte, die neben ihrer eigentlichen Intension zusätzlich Gruppierungsfunktionen wahrnehmen. Problematisch ist die unvollständige und informale Definition von *Package.* (vgl. dazu Abschnitt 6.1.1.1.6, S. 62)
		Dynamische Konzepte	Zustandsautomat, *Use Case*, *Activity Diagram* (Abläufe), *Collaboration*, *Sequence Diagram*	UML unterstützt eine Vielzahl von Konzepten und Beschreibungsmitteln zur Darstellung des Verhaltens. Diese Beschreibungsmittel sind verschiedenen Paradigmen (zustandsorientiert, prozedural und objektorientiert) zugeordnet. Eine gute Integration der nicht objektorientiertenKonzepte mit dem objektorientierten Paradigma ist daher notwendig. Zusätzlich ist auch eine Integration mit der statischen Strukturbeschreibung erforderlich. Beide Integrationen sind jedoch nur ansatzweise erkennbar. Auch die dynamischen Konzepte sind nicht exakt beschrieben.
		Nebenläufige Konzepte	aktive Objekte, asynchrone Botschaften, Knoten (Prozessor), Parallelausführung in Zuständen	Nebenläufigkeit ist in eine Reihe verschiedener Konzepte integriert. Allerdings sind diese noch nicht voll elaboriert und meist nur informativ. Dies liegt an dem eher sequentiellen Charakter vieler heutiger objektorientierter Programmiersprachen. Zur Unterstützung stark paralleler Anwendungen wird daher in UML eine Weiterentwicklung und Integration weiterer paralleler Beschreibungsmittel notwendig sein.
		Prozeßorientierte Konzepte	*Use Case, Activity Diagram*	Nur *Use Cases* und vielleicht *Activity Diagrams* können als prozeßorientiert bezeichnet werden. Allerdings benötigen diese beiden Konzepte auch eine sorgfältige Integration in die Objektorientierung. Eine Verknüpfung von Prozessen und Objekten (Objekt-Prozeß-Modelle, objektorientierte Petrinetze und Prozeßkalküle) kann auch zu einer Verbesserung der Unterstützung für Nebenläufigkeit führen, da Prozesse stark nebenläufig sein können.
		Weitere Konzepte		

Tabelle 8: Bewertungsrahmen abstrakte Syntax und Semantik für UML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
		Integration	ansatzweise, informal	Die Integration der einzelnen Konzepte ist als unbefriedigend zu bezeichnen. So ist es ohne weiteres möglich, in den verschiedenen Teilsichten widersprüchliche Aussagen zu modellieren. Eine Ursache dafür ist die unexakte Beschreibung der Semantik einzelner Sprachmittel. Dies führt zu Mehrdeutigkeiten und Interpretationsfreiräumen bei der Integration. Aber auch viele Integrationsregeln sind nur informal textuell beschrieben und erlauben insbesondere keine automatische Überprüfung. UML Version 1.1. ermöglicht mit der OCL eine bessere Beschreibung der Integrationsbedingungen. Die Verbesserung der Integration ist eine der wichtigsten Forderungen an UML.

Tabelle 9: Bewertungsrahmen abstrakte Syntax und Semantik für OML

		Merkmal	Ausprägung	Bewertung
	Aufbau	Grundkonzeption	Teilsprachen	Die Unterteilung in Teilsprachen ist historisch begründet. OML integriert verschiedene Teilsichten oder -sprachen, die aus älteren Modellierungssprachen übernommen worden sind.
		Objektorientierte Konzepte	Klasse, Objekt, Vererbung, Polymorphismus, Rollen	OML unterstützt die wichtigsten objektorientierten Konzepte. Dies sind im wesentlichen alle Konzepte, die schon aus den älteren Modellierungssprachen bekannt sind. Diese Konzepte werden jedoch nicht vollständig definiert, sondern dem Benutzer Interpretationsfreiräume gelassen. Die fehlende exakte Definition der Konzepte ist die Ursache für eine Reihe von Widersprüchen. Rollen kommen in OML eine große Bedeutung zu. Allerdings wird nicht gezeigt, wie dieses Konzept für die Implementierung nahtlos auf Klassen und Objekte abgebildet werden kann. Spezielle Konzepte, wie vererbte Beziehungen oder Verfeinern von Kardinalitäten sind nicht berücksichtigt.
		Modularisierungskonzepte	*Cluster*, Klasse	Der grundlegende Modularisierungsmechanismus ist der *Cluster.* Es gibt eine Reihe weiterer Sprachkonzepte, die neben ihrer eigentlichen Intension zusätzlich Gruppierungsfunktionen wahrnehmen. Problematisch ist die unvollständige und informale Definition von *Cluster.* OML unterstützt sogenannte *Cluster classes*. Problematisch ist jedoch, daß keine weitere Verfeinerung von *Cluster* vorgenommen wurde. So können *Cluster* für Teilsysteme, Entwurfsmuster oder einfache Gruppierungen verwendet werden. Unterschiede in der Bedeutung der einzelnen Ausprägungen können dadurch nicht berücksichtigt werden. Spezielle Sichtbarkeitsregeln fehlen. Nach außen sichtbare Komponenten werden auf den Umriß des *Cluster* gezeichnet. (vgl. dazu Abschnitt 6.1.1.1.6, S. 62)

Tabelle 9: Bewertungsrahmen abstrakte Syntax und Semantik für OML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Sprache	Konzepte (abstrakte Syntax und Semantik)	Dynamische Konzepte	Zustandsautomat, *Use Case, Task Script*, *Sequence Diagram*, *Collaboration Diagram*	OML unterstützt eine Vielzahl von Konzepten und Beschreibungsmitteln zur Darstellung des Verhaltens. Diese Beschreibungsmittel sind verschiedenen Paradigmen (zustandsorientiert, prozedural und objektorientiert) zugeordnet. Eine gute Integration der nicht objektorientierten Konzepte mit dem objektorientierten Paradigma ist daher notwendig. Zusätzlich ist auch eine Integration mit der statischen Strukturbeschreibung erforderlich. Beide Integrationen sind jedoch nur ansatzweise erkennbar. Auch die dynamischen Konzepte sind nicht exakt beschrieben.
		Nebenläufige Konzepte	*concurrent Object*, asynchrone Botschaften, Knoten (Prozessor), keine Unterstützung von Parallelität in Zuständen	Nebenläufigkeit ist in eine Reihe verschiedener Konzepte integriert. Allerdings sind diese noch nicht voll elaboriert und meist nur informativ. Dies liegt an dem eher sequentiellen Charakter vieler heutiger objektorientierter Programmiersprachen. Zur Unterstützung stark paralleler Anwendungen wird daher eine Weiterentwicklung und Integration weiterer paralleler Beschreibungsmittel notwendig sein.
		Prozeßorientierte Konzepte	*Use Case*	Nur *Use Cases* können als prozeßorientiert bezeichnet werden. Eine sorgfältige Integration in die Objektorientierung ist notwendig, aber nicht vollständig umgesetzt. Eine Verknüpfung von Prozessen und Objekten (Objekt-Prozeß-Modelle, objektorientierte Petrinetze und Prozeßkalküle) kann auch zu einer Verbesserung der Unterstützung für Nebenläufigkeit führen, da Prozesse stark nebenläufig sein können.
		Weitere Konzepte	nicht objektorientierte Konzepte	OML unterstützt einige Beschreibungsformen für Daten und Funktionen. Diese sind im wesentlichen informativer Art. Die Semantik wird nicht angegeben. Vorteilhaft können diese Beschreibungsmittel bei der Modellierung hybrider Systeme sein. Allerdings könnten auch stereotypisierte *Cluster* verwendet werden.
		Integration	ansatzweise. informal	Die Integration der einzelnen Konzepte ist als unbefriedigend zu bezeichnen. So ist es ohne weiteres möglich, in den verschiedenen Teilsichten widersprüchliche Aussagen zu modellieren. Eine Ursache dafür ist die unexakte Beschreibung der Semantik einzelner Sprachmittel. Dies führt zu Mehrdeutigkeiten und Interpretationsfreiräumen bei der Integration. Aber auch viele Integrationsregeln sind nur informal textuell beschrieben und erlauben insbesondere keine automatische Überprüfung. Die Verbesserung der Integration ist eine der wichtigsten Forderungen an OML. (vgl. dazu Abschnitt 6.1.4, S. 81)

6.2.3 Konkrete Syntax

Bezüglich der konkreten Syntax unterscheiden sich beide Modellierungssprachen stärker als in der abstrakten Syntax und Semantik. UML und OML basieren auf prinzipiell denselben Konzepten, die die abstrakte Syntax und Semantik festlegen. Bei der konkreten Syntax existiert so eine Festlegung nicht.

Tabelle 10: Bewertungsrahmen konkrete Syntax für UML

		Merkmal	Ausprägung	Bewertung
Sprache	Notation (konkrete Syntax)	Diagrammarten	vgl. Tabelle 4, S. 47	UML unterstützt eine Vielzahl von Diagrammen. Die Auswahl ist vor allem durch pragmatische Gründe geprägt. Viele Diagramme wurden schon in anderen Modellierungssprachen benutzt und haben sich zu einem gewissen Standard entwickelt. Kritisch ist jedoch anzumerken, daß keine Übersichtsdarstellung über die Diagramme und deren Diagrammelemente existiert. Auch ist der Verwendungszweck einiger Diagramme nicht nachvollziehbar. Überschneidungen in den Aufgabenbereichen erlauben die Frage, ob alle Diagramme notwendig sind oder nicht einige weggelassen bzw. ersetzt werden könnten. Die Zustands- und Sequenzdiagramme wirken überladen. Auch syntaktische Regeln wie etwa für Duplikate sind nicht festgelegt. (vgl. dazu Abschnitt 6.1.3, S. 75)
		Diagrammelemente	Für alle wichtigen Sprachkonzepte existieren korrespondierende graphische Notationen. (vgl. UML *Notation Guide* ([Rat97c], S. 21))	Die Auswahl der Notationselemente ist von OMT, Booch und OOSE geprägt. Durch die große Anzahl von Diagrammen gibt es sehr viele Notationselemente. Gut ist die Verwendung von Stereotypen zur Reduzierung von Symbolen und die Anwendung genereller Prinzipien. Die Umsetzung ist aber noch nicht immer zur Zufriedenheit gelöst. Einige Notationsfestlegungen sind aus ergonomischer Sicht unbefriedigend. Eine andere wichtige Frage, die unbeantwortet bleibt, ist die Abbildung der Notationselemente auf die Sprachkonzepte. (vgl. dazu Abschnitt 6.1.2, S. 67)
		Annotationen	keine	UML unterstützt keine textuellen Notationen. Einige Aspekte ließen sich textuell besser darstellen als durch Diagramme. Als Alternative und Ergänzung für Diagramme wären textuelle Annotationen hilfreich. Auf keinen Fall darf die Bereitstellung textueller Annotationen Werkzeugherstellern überlassen werden, da dies zu einer Reduzierung der Einheitlichkeit führen könnte. (vgl. dazu Abschnitt 6.1.2.1, S. 67)

Tabelle 10: Bewertungsrahmen konkrete Syntax für UML (Fortsetzung)

	Merkmal	Ausprägung	Bewertung
	Integration	unvollständig	Die Integration der einzelnen Teilsichten erfolgt implizit über das zugrundeliegende Modell. Die Integration bezieht sich im wesentlichen auf das Abbilden der Notationselemente auf die Sprachkonzepte. Diese Abbildungen müssen exakt und eindeutig sein, da nur so aus den einzelnen Sichten das Gesamtmodell erzeugt werden kann. Die Abbildungen sind jedoch nur unvollständig beschrieben und enthalten zum Beispiel keine Hinweise über Duplikate oder mehrfaches Auftreten von Notationselementen in verschiedenen Diagrammen. Die Abbildung zwischen Notationselement und Sprachkonzept muß noch verbessert werden. UML Version 1.1 enthält für die Abbildung des Notationselements auf das Sprachkonzept erweiterte Angaben. Allerdings sind diese Ausführungen nicht vollständig.

Tabelle 11: Bewertungsrahmen konkrete Syntax für OML

	Merkmal	Ausprägung	Bewertung
	Diagrammarten	vgl. Tabelle 4, S. 47	OML unterstützt eine Vielzahl von Diagrammen. Die Auswahl ist vor allem durch pragmatische Gründe geprägt. Diagramme wurden teilweise aus den Notationen von MOSES[a] und Firesmith übernommen. Positiv zu werten ist die Auflistung aller Diagramme und ihrer Diagrammelemente im Anhang von OML ([FiHe96], S. 192 ff.). Die Untergliederung der Diagramme in OML ist zu fein. So gibt es verschiedene Diagrammarten, die auch durch ein allgemeineres Diagramm ausdrückbar sind. Auch ist der Verwendungszweck einiger Diagramme nicht nachvollziehbar. Durch diese beiden Punkte wird die Anwendbarkeit der Diagramme erschwert. Überschneidungen in den Aufgabenbereichen erlauben die Frage, ob alle Diagramme notwendig sind oder nicht einige weggelassen bzw. ersetzt werden könnten. Auch syntaktische Regeln wie etwa für Duplikate sind nicht festgelegt. Eine Reduzierung und Verallgemeinerung der Diagramme ist notwendig. (vgl. dazu Abschnitt 6.1.3, S. 75)

Tabelle 11: Bewertungsrahmen konkrete Syntax für OML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Sprache	Notation (konkrete Syntax)	Diagrammelemente	Für alle wichtigen Sprachkonzepte existieren korrespondierende graphische Notationen.	Die Auswahl der Notationselemente ist von MOSES und Firesmith geprägt. OML enthält sehr viele Notationselemente. Dies erlaubt zwar die Unterscheidung einer Vielzahl von Konzepten, bedeutet aber auch eine große Anforderung an den Benutzer, der alle diese Symbole kennen muß. Gut ist die Verwendung von Stereotypen zur Reduzierung von Symbolen und die Anwendung genereller Prinzipien. Einige Notationsfestlegungen sind aus ergonomischer Sicht unbefriedigend. So wirft die starke Schachtelung von Notationselementen eine Vielzahl von Fragen auf. Andere Entscheidungen, wie die Auswahl spezieller Umrisse, sind subjektive Entscheidungen. Eine wichtige Frage, die unbeantwortet bleibt, ist die Abbildung der Notationselemente auf die Sprachkonzepte. (vgl. dazu Abschnitt 6.1.2, S. 67)
		Annotationen	CRC Cards für Rollen, Klassen, Objekte und *Cluster*	OML unterstützt CRC Cards. Detaillierte Annotationen fehlen jedoch. Die Unterstützung detaillierter textueller Annotationen wäre günstig. (vgl. dazu Abschnitt 6.1.2.1, S. 67)
		Integration	unvollständig	Die Integration der einzelnen Teilsichten erfolgt implizit über das zugrundeliegende Modell. Die Integration bezieht sich so im wesentlichen auf das Abbilden der Notationselemente auf die Sprachkonzepte. Diese Abbildungen müssen exakt und eindeutig sein, da nur so aus den einzelnen Sichten das Gesamtmodell erzeugt werden kann. Die Abbildungen sind jedoch nur unvollständig beschrieben und enthalten zum Beispiel keine Hinweise über Duplikate oder mehrfaches Auftreten von Notationselementen in verschiedenen Diagrammen.

a. MOSES (Methodology for Object-oriented Software Engineering of Systems) [HeEd94]

6.2.4 Anwendungsperspektive

Der geplante Anwendungsbereich einer Modellierungssprache bestimmt die Auswahl der Sprachmittel und erlaubt gleichzeitig eine Bewertung der Sprachmittel bezüglich ihrer Ausdrucksmächtigkeit und Angemessenheit relativ zum Modellierungszweck.

Tabelle 12: Bewertungsrahmen Anwendung für UML

		Merkmal	Ausprägung	Bewertung
	Benutzersichten	Anwendersicht	dominierend zwischenmenschliche Kommunikation Codegenerierung	UML ist eine Sprache für die Entwickler. Die Ausrichtung auf menschliche Benutzer und die Bereitstellung anschaulicher graphischer Beschreibungsformen ist daher positiv zu begrüßen. UML enthält auch Möglichkeiten zur Festlegung von Implementierungsdetails. Diese ermöglichen eine effektivere Codegenerierung.
		Metasicht	Erweiterungsmechanismen (Stereotypen, Name-Wert-Paare und Bedingungen) zum Anpassen der Sprache	Die Bereitstellung von Erweiterungsmöglichkeiten ist positiv zu bewerten. Die Erweiterungsmechanismen können zum einen im UML Metamodell angewendet werden. Dies erlaubt die Erweiterung des Metamodells ohne eine Strukturänderung. Zum anderen erlauben sie die Anpassung der Anwendersprache. Mit Stereotypen, Name-Wert-Paaren und Bedingungen stehen mächtige Werkzeuge für die Metasicht zur Verfügung. Das Fehlen formaler Grundlagen für die Erweiterungsmechanismen ist negativ zu bewerten und birgt ein großes Gefahrenpotential für die Wohldefiniertheit der Sprache. (vgl. Abschnitt 6.1.1.1.4, S. 60)
	Zweck	Anwendungszweck	Beschreibung und Modellierung von Systemen mit objektorientierten Modellen (und anschließender Software-Realisierung)	UML unterstützt die Modellierung und ist als Modellierungssprache prinzipiell geeignet. Dies ist dadurch begründet, daß UML viele Beschreibungsmittel für die objektorientierte Modellierung enthält, die aus älteren Ansätzen übernommen wurden und sich dort bewährt haben. Diese Aussage bedeutet jedoch nicht, daß alle Konzepte in UML adäquat beschrieben sind. Vielmehr muß für viele Konzepte die Semantik und Syntax genauer festgelegt werden. Auch das Grundkonzept von UML weist noch Mängel auf.

Tabelle 12: Bewertungsrahmen Anwendung für UML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Anwendung	Entwicklungstätigkeiten	Analysetätigkeiten	Objektorientierte Konzepte, *Use Cases*	Ein Vorteil der Objektorientierung ist ihre phasenübergreifende Anwendung in der Software-Entwicklung. Als objektorientierte Modellierungssprache unterstützt UML daher Analysetätigkeiten. Eine Differenzierung und Anpassung der objektorientierten Konzepte bezüglich der Analyse erfolgt nicht. Die zusätzliche Bereitstellung von *Use Cases* zur Beschreibung typischer Anwendungsfälle ist positiv zu bewerten. Negativ einzuschätzen, ist dagegen, daß UML keine Beschreibungsformen für (formale) Anforderungen enthält, die als Abstraktion aus den Anwenderfällen gewonnen werden können. Eine Möglichkeit wären z.B. an die Objektorientierung angepaßte Pflichtenhefte.
		Entwurfstätigkeiten	Objektorientierte Konzepte, Zustandsautomaten, *Activity Diagram*	Auch UML unterstützt wie die meisten objektorientierten Modellierungssprachen den Entwurf. Der Entwurf ist eine der Kernaufgaben der Modellierung. Die bereitgestellten Konzepte sind ausreichend.
		Implementierungstätigkeiten	Objektorientierte Konzepte Sichtbarkeitsregeln, Implementierungsdetails, textuelle (informale) Methodenrümpfe, *Activity Diagram* (Abläufe)	Die Implementierung wird von UML nur schwach unterstützt. Dies ist auch nicht verwunderlich, da UML eine Modellierungs- und keine Programmiersprache ist. Allerdings enthält UML Sprachmittel, die die Angabe von Implementierungsdetails erlaubt und Entwurfsmodelle zu Implementierungsmodellen verfeinern kann. Die eigentliche Implementierung liegt jedoch außerhalb der UML.
		Verification and Validation (V&V)	keine Unterstützung	V&V Maßnahmen sind eigentlich keine Modellierungsaufgaben. Es sollte jedoch eine Möglichkeit geben, V&V Maßnahmen mit UML zu verknüpfen, da sie als integrierter Bestandteil in allen Phasen der Software-Entwicklung angewendet werden sollten. Dies könnten z.B. Testfälle für spezielle Szenarien sein. UML unterstützt diese Form der Integration derzeit nicht. Zusatzinformationen für V&V können nur als Notizen an Notationselemente geheftet werden.
		Integration	durch die gleichen Sprachmittel während der einzelnen Tätigkeiten keine explizite Berücksichtigung der differenzierten Modellierungsziele in den einzelnen Phasen. formal unvollständig	Die Integration der einzelnen Phasen erfolgt durch die Objektorientierung. Dies ist einer der großen Vorteile der Objektorientierung. Allerdings werden die differenzierten Ziele in den einzelnen Phasen nicht berücksichtigt. Dies ist negativ, da manche der objektorientierten Konzepte in den einzelnen Phasen verschiedene Bedeutungen haben. Die Integration der einzelnen Phasen läßt sich weiterhin mit der Integration der Beschreibungsmittel, die in den einzelnen Phasen verwendet werden, bewerkstelligen.

Tabelle 13: Bewertungsrahmen Anwendung für OML

		Merkmal	Ausprägung	Bewertung
	Benutzersichten	Anwendersicht	zwischenmenschliche Kommunikation	OML ist eine Sprache für die Entwickler. Die Ausrichtung auf menschliche Benutzer und die Bereitstellung anschaulicher graphischer Beschreibungsformen ist daher positiv zu begrüßen.
		Metasicht	keine Unterstützung	OML enthält keine Konzepte, um die Sprache anzupassen und zu erweitern. Dies ist jedoch nicht als nachteilig zu bewerten, da für den Modellierer die Anwendersicht bedeutsamer ist. Für die Weiterentwicklung und Anpassung der Sprache bietet OML damit aber keine integrierten Mechanismen.
	Zweck	Anwendungszweck	Beschreibung von Systemen mit objektorientierten Modellen (und anschließender Software-Realisierung)	OML unterstützt die Modellierung und ist als Modellierungssprache prinzipiell geeignet. Dies ist dadurch begründet, daß OML viele Beschreibungsmittel für die objektorientierte Modellierung enthält, die aus älteren Ansätzen übernommen wurden und sich dort bewährt haben. Diese Aussage bedeutet jedoch nicht, daß alle Konzepte adäquat beschrieben sind. Vielmehr muß für viele Konzepte die Semantik und Syntax genauer festgelegt werden.
		Analysetätigkeiten	Objektorientierte Konzepte, *Use Cases, Task Scripts, Context Diagrams, Configuration Diagrams*	Ein Vorteil der Objektorientierung ist ihre phasenübergreifende Anwendung in der Software-Entwicklung. Als objektorientierte Modellierungssprache unterstützt OML daher Analysetätigkeiten. OML stellt eine Vielzahl von Beschreibungsmitteln für Analyse und Entwurf bereit. Dies begründet sich aus der Tatsache, daß OML auf älteren Modellierungssprachen basiert, die diese Betonung der frühen Entwicklungsphase ebenfalls enthielten. Eine Differenzierung und Anpassung der objektorientierten Konzepte bezüglich der Analyse erfolgt nicht. Die zusätzliche Bereitstellung von *Use Cases* oder *Task Scripts* zur Beschreibung typischer Anwendungsfälle ist positiv zu bewerten. Negativ einzuschätzen, ist dagegen, daß OML keine Beschreibungsformen für (formale) Anforderungen enthält, die als Abstraktion aus den Anwenderfällen gewonnen werden können. Eine Möglichkeit wären z.B. an die Objektorientierung angepaßte Pflichtenhefte.
		Entwurfstätigkeiten	Objektorientierte Konzepte, Zustandsautomaten	Auch OML unterstützt wie die meisten objektorientierten Modellierungssprachen den Entwurf. Der Entwurf ist eine der Kernaufgaben der Modellierung. Die bereitgestellten Konzepte sind ausreichend.

Tabelle 13: Bewertungsrahmen Anwendung für OML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Anwendung	Entwicklungstätigkeiten	Implementierungstätigkeiten	Objektorientierte Konzepte textuelle (informale) Methodenrümpfe	Die Implementierung wird von OML schwach unterstützt. Dies ist auch nicht verwunderlich, da OML eine Modellierungs- und keine Programmiersprache ist. Allerdings enthält OML Sprachmittel, die die Angabe von Implementierungsdetails erlaubt und Entwurfsmodelle zu Implementierungsmodellen verfeinern kann. Die eigentliche Implementierung liegt jedoch außerhalb der OML.
		Verification and Validation (V&V)	keine Unterstützung	V&V Maßnahmen sind eigentlich keine Modellierungsaufgaben. Es sollte jedoch eine Möglichkeit geben, V&V Maßnahmen mit OML zu verknüpfen, da sie als integrierter Bestandteil in allen Phasen der Software-Entwicklung angewendet werden sollten. Dies könnten z.B. Testfälle für spezielle Szenarien sein. OML unterstützt diese Form der Integration derzeit nicht. Zusatzinformationen für V&V können nur als Notizen an Notationselemente geheftet werden.
		Integration	durch die gleichen Sprachmittel während der einzelnen Tätigkeiten keine explizite Berücksichtigung der differenzierten Modellierungsziele in den einzelnen Phasen. formal unvollständig	Die Integration der einzelnen Phasen erfolgt durch die Objektorientierung. Dies ist einer der großen Vorteile der Objektorientierung. Allerdings werden die differenzierten Ziele in den einzelnen Phasen nicht berücksichtigt. Dies ist negativ, da manche der objektorientierten Konzepte in den einzelnen Phasen verschiedene Bedeutungen haben. Die Integration der einzelnen Phasen läßt sich weiterhin mit der Integration der Beschreibungsmittel, die in den einzelnen Phasen verwendet werden, bewerkstelligen.

6.2.5 Kriterien

Die Kriterien beleuchten die Ausprägung ausgewählter Eigenschaften einer Modellierungssprache und vermitteln gleichzeitig einen Eindruck über die Qualität der Sprache.

Tabelle 14: Bewertungsrahmen Kriterien für UML

		Merkmal	Ausprägung	Bewertung
	anwenderbezogen	Anwendbarkeit	eingeschränkt durch: widersprüchliche Aussagen hohe Sprachkomplexität gewährleistet durch: Praxiserfahrung mit objektorientierten Konzepten Übernahme "guter Praxis" aus älteren Notationen	Die Einschätzung der Anwendbarkeit ist unterschiedlich und vom Betrachter abhängig. So sind gerade Zustands- und Sequenzdiagramme wegen ihrer hohen Komplexität nur eingeschränkt anwendbar. Bei anderen Konzepten fehlt ein Bezug zum empfohlenen Verwendungszweck. Der sehr große Umfang von UML erschwert die Anwendbarkeit der Sprache, da keine Empfehlung gegeben wird, welche Konzepte in welcher Situation benutzt werden sollen. Auch die fehlende exakte Semantik der Konzepte und widersprüchliche Aussagen wirken sich negativ aus. Prinzipiell ist jedoch die praktische Anwendung gewährleistet.
		Anschaulichkeit Verständlichkeit	relativ gut positiv: Vertrautheit mit objektorientierten Konzepten Generelle Strategien bei der Symbolauswahl Graphische Notation negativ: komplizierte Konzepte komplizierte, unübersichtliche Darstellungen widersprüchliche Definitionen in der Beschreibung	UML ist in seiner Gesamtheit als recht anschaulich und verständlich einzuschätzen. Allerdings gibt es auch eine Vielzahl von Konzepten, die nicht anschaulich sind. Insbesondere komplizierte Konzepte, wie Stereotypen, Name-Wert-Paare oder der Zusammenhang Use *Case* / *Szenario,* können nicht als anschaulich empfunden werden. Auch komplexe und unübersichtliche Diagramme, wie Zustands- oder Sequenzdiagramme, sind hier zu nennen. Die Verständlichkeit leidet auch an der unvollständigen Sprachbeschreibung und der unexakten Festlegung der Semantik.
		Angemessenheit	UML umfaßt die wichtigsten Konzepte für die objektorientierte Modellierung	Die Einschätzung der Angemessenheit ist schwierig und stark subjektiv. Im wesentlichen umfaßt UML die Sprachkonzepte, die als angemessen für die objektorientierte Modellierung angesehen werden.
		Überprüfbarkeit	ansatzweise (Bedingungen, OCL)	Wegen der unvollständigen und nicht eindeutigen Darstellung der Konzepte, fehlt die Grundlage für eine Überprüfbarkeit der Modelle.

Tabelle 14: Bewertungsrahmen Kriterien für UML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Kriterien (Gesamtsicht)	modellbezogen	Mächtigkeit	ausreichend große Mächtigkeit erfüllt ihren Zweck als Sprache für die objektorientierte Modellierung nicht unterstützte Modellierungsideen	UML ist eine sehr mächtige Sprache mit vielen Konzepten. Auf Grund ihrer Komplexität unterstützt sie die objektorientierte Modellierung gut, jedoch nur ansatzweise Verteilung und Nebenläufigkeit. Es gibt auch eine Reihe spezieller Konstrukte, die nicht durch die UML abgedeckt werden (vererbte Beziehungen und Kardinalitäten). Prinzipiell ist hier immer ein Kompromiß zwischen Mächtigkeit und Einfachheit zu suchen.
		Eindeutigkeit	keine exakten Definitionen der Sprachkonzepte	Die fehlenden eindeutigen Definitionen der Sprachmittel beeinträchtigen die Qualität der Sprache negativ und wirken sich direkt und indirekt auf eine Reihe weiterer Eigenschaften aus. So ist mit der unvollständigen Spezifikation der Semantik einzelner Sprachmittel auch eine Einschränkung der Integration, Überprüfbarkeit und Konsistenz verbunden.
		Konsistenz	nur ansatzweise mit Bedingungen	Die Konsistenzprüfung zwischen Modell und modellierter Realität kann UML nicht leisten. UML kann durch Bereitstellung angemessener Beschreibungsmittel diese Überprüfung aber unterstützen. Die fehlende eindeutige Festlegung der abstrakten Syntax und Semantik wirkt sich auch negativ auf die Konsistenz aus.
		Formalisierungsgrad	willkürliche Festlegungen keine Einordnung in allgemeine Theorien oder axiomatische Definitionen ansatzweise OCL Bedingungen	UML enthält sowohl formale wie auch informale Teile. Der Formalisierungsgrad von UML ist unzureichend. So ist die Semantik vieler Konzepte unzureichend beschrieben. Gerade für Konzepte, die verschieden interpretiert werden, ist eine Festlegung notwendig. Auch wurde kein Versuch unternommen, bekannte formale Spezifikationssprachen als Bestandteil in UML zu integrieren. Allerdings enthält UML Version 1.1 nun OCL. Dies ist ein wichtiger und richtiger Schritt zur Verbesserung der Eindeutigkeit und Formalisierung.
		Integrationsgrad	gering gemeinsames Sprachmodell ist richtiger Ansatz	Die fehlende Formalisierung wirkt sich negativ auf die Integration aus. Die Integration muß entscheidend verbessert werden. Der Ansatz eines gemeinsamen Sprachmodells für die einzelnen Teilsichten ist positiv zu bewerten.

Tabelle 14: Bewertungsrahmen Kriterien für UML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
	ökonomisch	Wiederverwendbarkeit	gute Möglichkeiten auf Beschreibungs- und Modellebene	Die Verwendung genereller Prinzipien und die Unterstützung von Komposition und Ableitung ist positiv zu bewerten. Alle diese drei Möglichkeiten stehen sowohl dem Sprachentwickler als auch dem Modellierer zur Verfügung.
		Erweiterbarkeit	gute Erweiterungsmöglichkeiten, aber keine eindeutige und vollständige Definition dieser Mechanismen	Die Bereitstellung von Erweiterungsfähigkeiten ist als durchweg positiv zu beurteilen.

Tabelle 15: Bewertungsrahmen Kriterien für OML

		Merkmal	Ausprägung	Bewertung
		Anwendbarkeit	eingeschränkt durch: widersprüchliche Aussagen hohe Sprachkomplexität viele Diagramme mit überschneidenen Aufgaben viele Symbole und Beziehungen gewährleistet durch: Praxiserfahrung mit objektorientierten Konzepten Übernahme "guter Praxis" aus älteren Notationen OML Light	Die Einschätzung der Anwendbarkeit ist unterschiedlich und vom Betrachter abhängig. Der sehr große Umfang von OML erschwert die Anwendbarkeit der Sprache, da keine Empfehlung gegeben wird, welche Konzepte in welcher Situation benutzt werden sollen. So erschweren gerade die vielen Diagramme mit ähnlichen Aufgaben die richtige Auswahl. Bei anderen Konzepten fehlt ein Bezug zum empfohlenen Verwendungszweck. Auch die fehlende exakte Semantik der Konzepte und widersprüchliche Aussagen wirken sich negativ aus. Prinzipiell ist jedoch die praktische Anwendung gewährleistet. Insbesondere die Bereitstellung einer OML Light ist ein richtiger Ansatz und positiv einzuschätzen, da sie durch eine Reduzierung der Sprachkomplexität den Einstieg und die Anwendbarkeit der OML verbessert.

Tabelle 15: Bewertungsrahmen Kriterien für OML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
	anwenderbezogen	Anschaulichkeit Verständlichkeit	relativ gut positiv: Vertrautheit mit objektorientierten Konzepte Generelle Strategien bei der Symbolauswahl Graphische Notation negativ: komplizierte Konzepte komplizierte unübersichtliche Darstellungen widersprüchliche Definitionen in der Beschreibung	OML ist in seiner Gesamtheit als recht anschaulich und verständlich einzuschätzen. Allerdings gibt es auch eine Vielzahl von Konzepten, die nicht anschaulich sind. Insbesondere die starke Schachtelung und die Vielzahl verschiedener Symbole können nicht als anschaulich empfunden werden. Auch komplexe und unübersichtliche Diagramme sind hier zu nennen. Das Rollenkonzept ist ein Beispiel für ein recht anschauliches und mächtiges Sprachkonzept. Es benötigt aber eine exakte Definition und Integration mit den objektorientierten Konzepten. Die Verständlichkeit leidet an der unvollständigen Sprachbeschreibung und der unexakten Festlegung der Semantik.
		Angemessenheit	OML umfaßt die wichtigsten Konzepte für die objektorientierte Modellierung	Die Einschätzung der Angemessenheit ist schwierig und stark subjektiv. Im wesentlichen umfaßt OML die Sprachkonzepte, die als angemessen für die objektorientierte Modellierung angesehen werden.
		Überprüfbarkeit	ansatzweise	Wegen der unvollständigen und nicht eindeutigen Darstellung der Konzepte, fehlt die Grundlage für eine Überprüfbarkeit der Modelle.
		Mächtigkeit	ausreichend große Mächtigkeit erfüllt ihren Zweck als Sprache für die objektorientierte Modellierung nicht unterstützte Modellierungsideen	OML ist eine sehr umfangreiche Sprache mit vielen Konzepten. Auf Grund ihrer Komplexität unterstützt sie die objektorientierte Modellierung gut. Allerdings unterstützt sie Verteilung und Nebenläufigkeit nur ansatzweise. Es gibt auch eine Reihe spezieller Konstrukte, die nicht durch die OML abgedeckt werden (vererbte Beziehungen und Kardinalitäten). Prinzipiell ist hier immer ein Kompromiß zwischen Mächtigkeit und Einfachheit zu suchen.

Tabelle 15: Bewertungsrahmen Kriterien für OML (Fortsetzung)

		Merkmal	Ausprägung	Bewertung
Kriterien (Gesamtsicht)	modellbezogen	Eindeutigkeit	keine exakten Definitionen der Sprachkonzepte	Die fehlenden eindeutigen Definitionen der Sprachmittel beeinträchtigen die Qualität der Sprache negativ und wirken sich direkt und indirekt auf eine Reihe weiterer Eigenschaften aus. So ist mit der unvollständigen Spezifikation der Semantik einzelner Sprachmittel auch eine Einschränkung der Integration, Überprüfbarkeit und Konsistenz verbunden.
		Konsistenz	nur ansatzweise mit Bedingungen	Die Konsistenzprüfung zwischen Modell und modellierter Realität kann OML nicht leisten. OML kann durch Bereitstellung angemessener Beschreibungsmittel diese Überprüfung aber unterstützen. Die fehlende eindeutige Festlegung der abstrakten Syntax und Semantik wirkt sich negativ auf die Konsistenz aus.
		Formalisierungsgrad	willkürliche Festlegungen keine Einordnung in allgemeine Theorien oder axiomatische Definitionen ansatzweise	OML enthält vorwiegend informale Teile. Der Formalisierungsgrad von OML ist unzureichend. So ist die Semantik vieler Konzepte unzureichend beschrieben. Gerade für Konzepte, die verschieden interpretiert werden, ist eine Festlegung notwendig. Auch wurde kein Versuch unternommen, bekannte formale Spezifikationssprachen als Bestandteil in OML zu integrieren.
		Integrationsgrad	gering gemeinsames Sprachmodell ist richtiger Ansatz	Die fehlende Formalisierung wirkt sich negativ auf die Integration aus. Die Integration muß entscheidend verbessert werden. Der Ansatz eines gemeinsamen Sprachmodells für die einzelnen Teilsichten ist positiv zu bewerten. Allerdings ist das OML Metamodell nur sehr oberflächlich beschrieben.
	ökonomisch	Wiederverwendbarkeit	gute Möglichkeiten auf Beschreibungs- und Modellebene	Die Verwendung genereller Prinzipien und die Unterstützung von Komposition und Ableitung ist positiv zu bewerten. Alle diese drei Möglichkeiten stehen dem Modellierer zur Verfügung. Die Nutzung von Widerverwendungsmöglichkeiten auf Beschreibungsebene dient hauptsächlich der Verbesserung des Sprachverständnisses und weniger der Erweiterung der Sprache.
		Erweiterbarkeit	keine Unterstützung	Die Integration von Erweiterungsmechanismen in die Sprache ist vorteilhaft, aber nicht notwendig. Aus diesem Grund erfolgt für OML keine Bewertung dieses Aspekts.

6.3 Andere Vergleiche von UML und OML

Aus der Gruppe der Verfasser der OML gibt es eine Reihe von Vergleichen zwischen UML und OML (([FiHe96], S.200 ff.), [FiHe97], [Hen97]). Abgesehen davon, daß hier Zweifel an der Unparteilichkeit

der Autoren aufkommen (s. Zitat in Abschnitt 7, S. 103), fällt im Unterschied zum oben ausgeführten Vergleich auf, daß es sich dabei um quantitativ ausgerichtete Vergleiche der zur Verfügung stehenden Sprachmittel handelt - mitunter ergänzt um eine Diskussion über die Wahl verschiedener graphischer Repräsentationen für einzelne Modellelemente. Eine kritische qualitative Betrachtung der Definitionen der einzelnen Sprachmittel fehlt dagegen.

7 Abschließende Bemerkungen

Die Auswahl von Modellierungssprachen empfiehlt eine sorgfältige, vergleichende Beurteilung wesentlicher Eigenschaften. Der vorliegende Arbeitsbericht hat gezeigt, daß eine solche Beurteilung in Teilen ausgesprochen problematisch ist. So sind einige Eigenschaften, wie etwa Angemessenheit, nicht unabhängig vom jeweiligen Verwendungskontext zu beurteilen. Angesichts der breiten Streuung möglicher Einsatzbereiche ist es deshalb kaum möglich, zu generellen Aussagen über die diesbezügliche Qualität zu gelangen. Daneben gibt es Eigenschaften, wie etwa Verständlichkeit von Konzepten oder Anschaulichkeit der Notation, deren Beurteilung empirisch ausgerichtete Arbeiten im Bereich der Kognitionspsychologie empfiehlt. Solange es keine gehaltvollen Untersuchungsergebnisse aus diesem Bereich gibt, muß sich eine Beurteilung solcher Eigenschaften mit wenigen Plausibilitätsannahmen begnügen. Vor diesem Hintergrund ist es wichtig, noch einmal zu betonen, daß der vorgestellte Bezugsrahmen kein Bewertungsverfahren darstellt, das mit leicht nachvollziehbaren Meßvorschriften die Evaluation objektorientierter Modellierungssprachen ermöglicht. Stattdessen handelt es sich um einen gedanklichen Bezugsrahmen, der wesentliche Beurteilungskriterien, ggfs. zusammen mit denen ihnen innewohnenden Problemen, aufzeigt. Die sinnvolle Anwendung des Bezugsrahmens setzt deshalb eine qualifizierte und kritische Interpretation der Beurteilungskriterien voraus.

Die Betrachtung von UML und OML hat eine Reihe von Ergebnissen gezeitigt. Positiv ist zu vermerken, daß gegenüber früheren objektorientierten Modellierungsansätzen (wie etwa [Boo94], [Jac92], [Rum91]) deutliche Fortschritte zu verzeichnen sind. Sie betreffen die präzisere Beschreibung der Sprachkonzepte, das in Teilen erfolgreiche Bemühen um Erweiterbarkeit sowie die Korrektur zahlreicher Ungereimtheiten, die in einschlägigen Ansätzen in der Vergangenheit zu finden waren. Dessen ungeachtet weisen beide Vorschläge deutliche Unzulänglichkeiten auf. So sind immer noch unvollständige oder mehrdeutige Beschreibungen einzelner Sprachmittel (Diagramme und Elemente) zu beklagen. Daneben erschwert die erhebliche Vielfalt von zum Teil redundanten Diagrammarten das Erlernen und Anwenden beider Modellierungssprachen. Für die Verwendung in der Lehre sind vor allem die UML-Dokumente kaum geeignet. Wir selbst - obschon wir uns mit der objektorientierten Modellierung vertraut fühlen - empfanden die Lektüre der UML-Dokumentation mitunter reichlich verwirrend, um nicht zu sagen: ziemlich lästig. Das folgende Zitat mag einen Eindruck davon vermitteln, warum wir zu diesem Urteil gelangen:

> "Instance of is an association between an Instance instance and its Type instance. The responsibility of instance of is to specify that the Instance instance is a concrete manifestation of the Type instance. Every Type instance may have zero or more Instance instances, and every Instance instance is the instance of not more than one Type instance." ([Rat97c], S. 32)

Aus wissenschaftlicher Sicht ist zudem zu beklagen, daß die Verfasser beider Vorschläge es mitunter im - wie auch immer entstandenen - Eifer des Gefechts versäumen, die nötige Sachlichkeit walten zu lassen. Dieser Umstand ist umso bedauerlicher, als die wissenschaftliche Ausseinandersetzung mit dem Thema ohnehin erheblich unter den wirtschaftlichen Einflüssen, die mit der Standardisierung einer Modellierungssprache einhergehen, leidet. So verkündet die Firma Rational in ebenso gönnerhafter wie selbstgefälliger Überheblichkeit, ihr Anliegen sei es "(to) bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features" ([Rat97n], S. 12). Auch die Verfasser der OML - obgleich weniger durch wirtschaftliche Interessen getrieben - lassen sich zu unsachlichen Aussagen

hinreißen: "It is our professional opinion that OML COMN is currently the best available notation for object-oriented modeling." ([FiHe96], S. 200)

Angesichts des Umstand, daß viele Fragen, die mit dem Entwurf und der Verwendung von Modellierungssprachen zusammenhängen, noch gar nicht oder nur unzureichend wissenschaftlich durchdrungen sind (vgl. dazu [Fra97], S. 33 f.), wirken derartige Äußerungen ziemlich deplaziert. Auch wenn es aus ökonomischen Gründen sinnvoll sein mag, möglichst bald zu einer Standardisierung zu gelangen, gibt es aus wissenschaftlicher Sicht noch einen erheblichen Bedarf an einschlägigen Untersuchungen. Das gilt für die Überarbeitung bzw. den Neuentwurf objektorientierter Modellierungssprachen und für korrespondierende empirische Begleitforschung wie für die damit in Wechselwirkung stehende Verfeinerung einschlägiger Anforderungs- bzw. Beurteilungskriterien. Der dargestellte Bezugsrahmen ist in diesem Sinn zu verstehen.

Literatur

[AhUl96] Aho, A.V.; Ullman, J.D.: Informatik. Datenstrukturen und Konzepte der Abstraktion. Bonn, Albany et al.: Thomson 1996

[ArBo91] Arnold, P.; Bodoff, S.; Coleman, D.; Gilchrist, H.; Hayes, F.: An Evaluation of Five Object-Oriented Development Methods. Report No. HPL-91-52, June 1991 Bristol 1991

[BeLu80] Berger, P.L.; Luckmann, T. : Die gesellschaftliche Konstruktion der Wirklichkeit. Eine Theorie der Wissenssoziologie. Frankfurt/M.: Fischer 1980

[Ber97] Berard, E.V.: Be Careful With "Use Cases". 1997. Obtained via http://www.toa.com/pub/html/use_case.html

[Big97] Biggs, P.: A Survey of Object-Oriented Methods. Obtained via http://www.dur.ac.uk/~dcs3pjb/survey.html

[BMRS96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. New York et al.: Wiley. 1996.

[Boo94] Booch, G.: Object-oriented Analysis and Design with applications. 2nd ed., Redwood/CA.: Benjamin Cummings 1994

[Bur94] Burkhardt, R.: Modellierung dynamischer Aspekte mit dem Objekt-Prozess-Modell. Dissertation, Ilmenau. 1994

[Bur97] Burkhardt, R.: UML - Unified Modeling Language: Objektorientierte Modellierung fuer die Praxis. Reading/Mass. et al.: Addison-Wesley. 1997

[BrDr93] Bröhl, A.-P.; Dröschel, W. (Eds.): Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden. München, Wien: Oldenbourg 1993

[Car66] Carnap, R.: Der logische Aufbau der Welt. 3. Auflage. Hamburg:Meiner 1966

[Che81] Checkland, P.: Systems thinking, systems practice. Chichester et al.: Wiley 1981

[Cho77] Chomsky, N.: Reflexionen über die Sprache. Frankfurt/M.: Suhrkamp 1977

[CrRo92] Cribbs, J.; Roe, C.; Moon, S.: An Evaluation of Object-Oriented Analysis and Design Methodologies. New York: SIGS Books 1992

[CuBe89] Cunningham, W.; Beck, K.: A laboratory for teaching object oriented thinking. In: OOPSLA ’89 - conference proceedings. SIGPLAN Notices, 24, 10, S. 1-6. 1989.

[DeFa92] De Champeaux, D.; Faure, P.: A comparative study of object-oriented analysis methods. In: JOOP, No. 1, Vol. 5, 1992, pp. 21-33

[Dod96] Dodani, M.: Object-oriented methodologies in practice: The "Big Picture". In: JOOP. March-April 1996.

[FiHe96] Firesmith, D.; Henderson-Sellers, B.; Graham, I.; Page-Jones, M.: OPEN Modeling Language (OML). Reference Manual. Version 1.0. 8 December 1996. http://www.csse.swin.edu.au/OPEN/comn.html

[FiHe97] Firesmith, D.; Henderson-Sellers, B.: Evaluating Third Generation OO Software Development Approaches. 17.10.1997 (http://www.csse.swin.edu.au/cotar/OPEN/istwww.pdf)

[FiKe91] Fichman, R.G.; Kemerer, C.F.: Object-Oriented and Conventional Analysis and Development Methodologies: Comparison and Critique Boston, Ma. 1991

[Fir92] Firesmith, D.: Object-oriented requirements analysis and logical design. Chichester. 1992. ISBN 0-471-57807-X

[Fir95] Firesmith, D.: Basic Object-Oriented Concepts and Terminology: A Comparison of Methods and Languages. In: OOP 95 Conference Proceedings. S. 67-103. München 1995

[Fra97] Frank, U.: Towards a Standardization of Object-Oriented Modelling Languages? Arbeitsberichte des Instituts fuer Wirtschaftsinformatik, Nr. 3, Koblenz 1997. http://www.uni-koblenz.de/iwi/

[Fra93] Frank, U.: A Comparison of two Outstanding Methodologies for Object-Oriented Design. Arbeitspapiere der GMD, No. 779, Sankt Augustin 1993

[FrHa97] Frank, U.; Halter, S.: Enhancing Object-Oriented Software Development with Delegation. Arbeitsberichte des Instituts für Wirtschaftsinformatk, No. 2, Koblenz 1997

[GHJV95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Reading/Mass. et al.: Addison-Wesley 1995

[GoRu92] Goldberg, A.; Rubin, K.S.: Object Behaviour Analysis. In: Communications of the ACM. Vol. 35, No. 9, 1992, pp. 48-62

[GoSt90] Goldstein, R.C.; Storey, V.C.: Some Findings on the Intuitiveness of Entity Relationship Constructs. In: Lochovsky, F.H. (Ed.): Entity Relationship Approach to Database Design and Query. Amsterdam: Elsevier 1990

[Gra91] Graham, I.: Object-Oriented Methods. Wokingham et al.: Addison-Wesley 1991

[GrHa87] Grewendorf, G.; Hamm, F.; Sternefeld, W.: Sprachliches Wissen. Frankfurt/M.: Suhrkamp 1987

[Hen92] Henderson-Sellers, B.: A Book of Object-Oriented Knowledge: Object-Oriented Analysis, Design and Implementation. A new Approach to Software Engineering. Englewood Cliffs, NJ: Prentice Hall 1992

[Hen97] Henderson-Sellers, B.: Choosing between UML and OPEN. 1997
(http://www.csse.swin.edu.au/cotar/OPEN/CHOOSING/choosing.html)

[HeEd94] Henderson-Sellers, B.; Edwards, J.M.: Book Two of Object-Oriented Knowledge: The Working Object. Object-Oriented SoftwareEngineering: Methods and Management. Sidney et al.: Prentice Hall 1994

[Hen75] Henle, P.: Sprache, Denken, Kultur. Frankfurt/M.: Suhrkamp 1975

[Hew91] Hewlett Packard: An Evaluation of Five Object-Oriented Development Methods. Software Engineering Department, HP Laboritories. Bristol 1991

[Hit95] Hitchman, S.: Practitioner Perceptions on the Use of some Semantic Concepts in the Entity Relationship Model In: European Journal of Information Systems, Vol. 4, 1995, pp. 31-40

[HoGo93] Hong, S.; Goor, G.: A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In: Nunamaker, J.F.; Sprague, R.H. (Ed.): Information Systems: Collaboration Technology, Organizational Systems, and Technolgoy. Proceedings of the 26th International Hawaii International Conferenc on System Sciences. Los Alamitos 1993, pp. 689-698

[Hsi92] Hsieh, D.: Survey of object-oriented analysis/design methodologies and future CASE frameworks. Menlo Park, Ca. 1992

[IBM97] IBM; ObjecTime Limited: OMG OA&D RFP Response Version 1.0. 1997.
http://www.omg.org/library/schedule/AD_RFP1.htm
http://www.omg.org/docs/ad/97-01-18.pdf

[Jac92] Jacobson, I.; Christerson, M; Jonsson, P; Overgaard, G.: Object-Oriented Engineering. A Use Case Driven Approach. Reading, Mass.: Addison-Wesley 1992

[Kow96] Kowalk, W.P.: System - Modell - Programm. Heidelberg, Berlin, Oxford: Spektrum 1996

[Lor96] Lorenz, K: Sprache. In: Enzyklopädie Philosophie und Wissenschaftstheorie. Hg. von J. Mittelstraß. Bd. 4, Stuttgart, Weimar: Metzler 1996, S. 49-53

[MaOd92] Martin, J.; Odell, J. J.: Object-Oriented Analysis and Design. Englewood Cliffs, Prentice Hall 1992. ISBN: 0-13-630245-9

[Mat87] Maturana, H.R.: Kognition. In: Schmidt, S.J. (Hg.): Der Diskurs des Radikalen Konstruktivismus. Frankfurt/M.: Suhrkamp 1987, S. 89-118

[Mey84] Meyer, W.J.: Linguistik. In: Enzyklopädie Philosophie und Wissenschaftstheorie. Hg. von J. Mittelstraß. Bd. 2, Mannheim, Wien, Zürich: B.I.-Wissenschaftsverlag 1984, S. 612-616

[Mey97] Meyer, B.: Object-Oriented Software Construction. 2nd Ed., Englewood Cliffs: Prentice Hall 1997

[MoPi93] Motschnig-Pitrik, R.: The Semantics of Parts Versus Aggregates in Data/Knowledge Modelling. In: Rolland,C.; Bodart, F.; Cauvet, C.: Advanced Information Systems Engineering, CAiSE'93, Paris, France, June 8-11, 1993, Proceedings. Lecture Notes in Computer Science Vol. 685 Springer.1993. ISBN 3-540-56777-1. S.352-373

[MoPu92] Monarchi, D.E.; Puhr, G.: A Research Typology for Object-Oriented Analysis and Design. In: Communications of the ACM, Vol. 35, No. 9, 1992, pp. 35-47

[Obj97] Object Agency: A Comparison of Object-Oriented Development Methodologies. Obtained via http://www.toa.com/pub/html/mcr.html

[OMG96] Object Management Group: Object Analysis & Design RFP-1, ad/96-05-01, 1996. Obtained via http://www.omg.org/library/public-doclist.html

[OtWi90] Ottmann, Thomas ; Widmayer, Peter: Algorithmen und Datenstrukturen. Mannheim (u.a.) : BI-Wissenschaftsverlag. 1990. ISBN 3-411-03161-1.

[Rat97a] Rational: UML: Proposal to the Object Management Group's Analysis and Design Task Force in Response to OADTF-RFP1. 13.01.1997. Nr: 97-01-01.
http://www.omg.org/library/schedule/AD_RFP1.htm
http://www.rational.com

[Rat97b] Rational: UML Summary. Version 1. 0. 13.01.1997. Nr: 97-01-02.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97c] Rational: UML Semantics. Version 1.0. 13.01.1997. Nr:97-01-03.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97d] Rational: UML Semantics. Appendix M1 - UML Glossary. Version 1.0. 13.01.1997. Nr:97-01-04.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97e] Rational: UML Semantics. Appendix M2: UML Meta-Metamodel. Version 1.0. 13.01.1997. Nr:97-01-05.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97f] Rational: Appendix M3: UML Meta-Metamodel Alignment with MOF and CDIF. Version 1.0. 13.01.1997. Nr:97-01-06.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97g] Rational: UML Semantics. Appendix M4: Relationship to OMG Technologies. Version 1.0. 13.01.1997. Nr:97-01-07.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97h] Rational: UML Semantics Appendix M5 -Relationship to Reference Model of Open Distributed Computing. Version 1.0. 13.01.1997. Nr:97-01-08.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97i] Rational: UML Notation Guide. Version 1.0. 13.01.1997. Nr:97-01-09.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97j] Rational: Process-Specific Extensions. Version 1.0. 13.01.1997. Nr:97-01-010.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97k] Rational: UML-Compliant Object Analysis & Design Facility.Version 1.0 beta 1. 13.01.1997. Nr:97-01-11.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97l] Rational: Mapping of UML to CORBA IDL. Version 1.0 beta 1. 13.01.1997. Nr:97-01-12.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97m] Rational: UML-Compliant Interchange Format. Version 1.0. 13.01.1997. Nr:97-01-13.
http://www.rational.com
http://www.omg.org/library/schedule/AD_RFP1.htm

[Rat97n] Rational: UML-Summary. Version 1.1. 01.09.1997.
http://www.rational.com

[Rat97o] Rational: OCL. Version 1.1. 01.09.1997.
http://www.rational.com

[Rat97p] Rational: UML-Notation Guide. Version 1.1. 01.09.1997.
http://www.rational.com

[Rat97q] Rational: UML-Semantics. Version 1.1. 01.09.1997.
http://www.rational.com

[Ree95] Reenskaug, T.: Working with Objects: The OORAM Software Engineering Method. Englewood Cliffs: Prentice Hall 1995

[Rum91] Rumbaugh, J. et al.: Object-oriented Modelling and Design. Englewood Cliffs, N.J. 1991

[Rum96a] Rumbaugh, J.: Notation notes: Principles for choosing notation In: Journal of Object-Oriented Programming (JOOP), Vol. 8, No. 10, Mai 1996, pp. 11-14

[Sch94] Schäfer, S.: Objektorientierte Entwurfsmethoden: Verfahren zum objektorientierten Softwareentwurf im Überblick. Addison-Wesley. 1994. ISBN 3-89319-692-7

[Sch97] Schienmann, B.: Objektorientierter Fachentwurf. Teubner Verlag. Stuttgart, Leipzig. 1997. ISBN 3-8154-2305-8.

[Ste93] Stein, W.: Objektorientierte Analysemethoden - Vergleich, Bewertung, Auswahl. Mannheim: BI-Verlag. 1993

[SüEb97] Süttenbach, R.; Ebert, J.: A Booch Metamodel. Fachberichte Informatik 5/97, Universität Koblenz-Landau 1997

[Tas97] Taskon: The OOram Meta-Model - combining role models, interfaces, and classes to support system centric and program centric modeling. A proposal in response to OMG OA&D RFP-1, 1997. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[Wit71] Wittgenstein, L.: Philosophische Untersuchungen. Frankfurt/M.: Suhrkamp 1971

[WiWi90] Wirfs-Brock, R.J.; Wilkerson, B.; Wiener, L.: Designing Object-Oriented Software. Englewood Cliffs, NJ.: Prentice Hall 1990