Vereinheitlichte Spezifikation von Komponenten

Grundlagen, UNSCOM Spezifikationsrahmen und Anwendung

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
an der Wirtschaftswissenschaftlichen Fakultät
der Universität Augsburg

vorgelegt von
Dipl.-Wirtsch.-Inform.
Sven Overhage
aus
D-61191 Rosbach vor der Höhe

Erstgutachter: Professor Dr. Klaus Turowski
Zweitgutachter: Professor Dr. Robert Klein
Vorsitzender der mündlichen Prüfungskommission: Professor Dr. Axel Tuma
Datum der mündlichen Prüfung: 22. Juni 2006

Für meine Eltern
SO

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADL	Architecture Description Language
BNF	Backus-Naur-Form
CCDL	Component Contract Definition Language
CCM	CORBA Component Model
COM	Component / Common Object Model
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
DCOM	Distributed Component / Common Object Model
EJB	Enterprise Java Beans
FCM	Factor Criteria Metrics
GQM	Goal Question Metric
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JAD	Joint Application Design
NAICS	North American Industry Classification System
NATO	North Atlantic Treaty Organization
OCL	Object Constraint Language
OMG	Object Management Group
PDL	Protocol Definition Language
QDL	Quality of Service Definition Language
QIDL	Quality Interface Definition Language
QML	Quality of Service Modeling Language
QoS	Quality of Service
QuO	Quality of Service for Objects
RDF	Resource Description Framework
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
UNSCOM	Unified Specification of Components
UNSPSC	United Nations Standard Products and Services Code
XML	Extensible Markup Language

Abbildungsverzeichnis

Auch eine Reise von tausend Meilen
beginnt mit einem ersten Schritt.
(Lao-Tse, chinesischer Philosoph, 6.Jh. v. Chr.)

1 Einleitung

Das Software Engineering und die betriebliche Anwendungsentwicklung stehen zu Beginn des 21. Jahrhunderts vor einer Vielzahl scheinbar gegensätzlicher Herausforderungen. So ist einerseits ein verstärkter Druck zur Senkung der Entwicklungs-, Einführungs- und Betriebskosten von Anwendungssystemen sowie zur Straffung der entsprechenden Prozesse zu verspüren. Gleichzeitig wachsen andererseits die an betriebliche Anwendungssysteme und deren Entwicklung gestellten Ansprüche stetig weiter. Dies gilt insbesondere, seit Unternehmen die Bedeutung ihrer Anwendungssysteme als strategischen Wettbewerbsfaktor zu erkennen und in ihrer Geschäfts- bzw. E-Business-Strategie entsprechend zu berücksichtigen beginnen [BROWN 2000:5]. Heute zählen vor allem die Beherrschung der hohen Anwendungssystem- und Entwicklungskomplexität, die Möglichkeit zur flexiblen Anpassung von Anwendungssystemen an Änderungen im Geschäftsumfeld sowie die Reduzierung der Entwicklungszeit [MEYER 1997:17-19; BROWN 2000:6-12; CHEESMAN und DANIELS 2001:2] zu den zentralen Anforderungen, die an das Software Engineering und die betriebliche Anwendungsentwicklung gestellt werden.

So sind in den letzten Jahrzehnten sowohl die funktionalen als auch die nicht-funktionalen (also die Qualitäts-) Anforderungen, die an Anwendungssysteme gestellt werden, kontinuierlich gestiegen und verursachen bei der Entwicklung einen immer höheren Komplexitätsgrad. Hiervon ist insbesondere die Entwicklung von betrieblichen Anwendungssystemen betroffen, die heutzutage zahlreiche Dienste zur Unterstützung verschiedener Funktionsbereiche in integrierter und effizienter Form anbieten müssen [KALAKOTA und ROBINSON 2001:54]. Darüber hinaus verändern Unternehmen ihre Geschäftspraktiken sowie die damit verbundenen Geschäftsprozesse mit stetig zunehmender Häufigkeit. Dies geschieht im Zuge der aktuellen wirtschaftlichen Entwicklung, mit der unter anderem die Reduzierung der Fertigungstiefe, die Reorganisation von Geschäftsprozessen und die Schaffung flexibler (virtueller) Kooperationen einhergehen [BROWN 2000:208f.]. Betriebliche Anwendungssysteme sind daher so zu entwickeln, dass sie eine hohe Flexibilität besitzen und sich rasch an eine Vielzahl von Veränderungen anpassen lassen [ALLEN und FROST 1998:3f.; CHEESMAN und DANIELS 2001:2]. Schließlich wird die Anwendungsentwicklung dadurch erschwert, dass Entwicklungsprojekte trotz der hohen Komplexität und der zu realisierenden Flexibilität der Anwendungssysteme immer zügiger durchzuführen und fle-

xibler zu steuern sein müssen. Denn gerade hiervon wird die erfolgreiche Partizipation eines Unternehmens an den rasch entstehenden und sich wandelnden globalen (elektronischen) Märkten der neuen Ökonomie maßgeblich beeinflusst [BROWN 2000:209].

Mit dem schnellen Anwachsen der Ansprüche vermochten das Software Engineering und die betriebliche Anwendungsentwicklung trotz vieler Fortschritte nicht Schritt zu halten. Daher stellt die Entwicklung von Anwendungssystemen, die den genannten zentralen Anforderungen genügen, eine derzeit kaum zu bewältigende Aufgabe dar [BROWN 2000:4f.; HERZUM und SIMS 2000:27]. Zur Beendigung der durch das Auseinanderlaufen von Anspruch und Wirklichkeit verursachten „Software-Krise" wurde bereits in den 1960er Jahren die Einführung der *komponentenorientierten Anwendungsentwicklung* als neue Entwicklungstechnik vorgeschlagen [MCILROY 1968]. Das Konzept der Komponentenorientierung wurde aus den anderen Ingenieursdisziplinen (dem Maschinenbau, der Elektrotechnik etc.) übernommen, in denen es jeweils im Rahmen eines historischen Reifungsprozesses eingeführt wurde [SPEED, et al. 2001] und wesentlich zur Entstehung der modernen industriellen Massenfertigung bzw. kundenindividuellen Massenfertigung (Mass Customization [PINE II 1993]) aus standardisierten, vorgefertigten Teilen beigetragen hat [CZARNECKI und EISENECKER 2000:3-5]. Der Einführung der Komponentenorientierung im Software Engineering und der betrieblichen Anwendungsentwicklung wird ein vergleichbares Potenzial zugeschrieben, den gegenwärtigen Missstand zu revolutionieren. Insbesondere verspricht ihr modulares Entwicklungsparadigma einen Beitrag zur Lösung *jeder* der eingangs genannten zentralen Anforderungen zu leisten [LIM 1994; ORFALI, et al. 1996:29-32; SAMETINGER 1997:11-15; BROWN 2000:74f.; CHEESMAN und DANIELS 2001:2].

Gemäß diesem neuen Paradigma sind Anwendungssysteme durch die Zerlegung der Entwicklungsaufgabe in leichter handhabbare Teile und die daran anschließende Komposition von ausgewählten bzw. neu geschaffenen Komponenten, die diese Teilaufgaben lösen, zu entwickeln [BROWN 2000:72]. Dabei wird mit der Zerlegung von Entwicklungsaufgaben in überschaubare Komponenten (dem sog. „Divide Et Impera") zunächst eine anerkannte, bewährte Strategie zur *Komplexitätsreduktion* und damit zur Beherrschung der Komplexität in das Software Engineering bzw. die betriebliche Anwendungsentwicklung eingeführt [SPEED, et al. 2001:675]. Gleichzeitig wird die *Anpassungsfähigkeit* von Anwendungssystemen verbessert, da sich aus eigenständigen Komponenten aufgebaute Anwendungssysteme durch Überarbeiten und Austauschen einzelner Komponenten leichter an Veränderungen anpassen lassen als monolithische Anwendungssysteme. Bei letzteren ist hierzu normalerweise eine umfassende Überholung in Form eines Re-Engineering notwendig [CHEESMAN und DANIELS 2001:2]. Mit dem modularen Entwicklungsparadigma lassen sich schließlich auch *flexibler steuerbare* und *zügiger durchzuführende Entwicklungspro-*

jekte realisieren, da die Entwicklung der einzelnen Komponenten eines Anwendungssystems sowohl inkrementell, also in verschiedenen Ausbaustufen, als auch parallelisiert durchgeführt werden kann, nachdem die Anwendungssystemarchitektur festgelegt wurde [BROWN 2000:74f.]. Wegen der Möglichkeit, dabei bereits zuvor entwickelte Komponenten wieder zu verwenden oder – die Entstehung entsprechender Komponentenmärkte vorausgesetzt – von spezialisierten Drittherstellern am Markt zu erwerben, bietet die Komponentenorientierung außerdem ein erhebliches Potenzial zur *Senkung der Entwicklungskosten* sowie zur *Straffung des Entwicklungsprozesses*, etwa durch eine Reduzierung der Fertigungstiefe in der Anwendungsentwicklung [LIM 1994]. Prinzipiell ermöglicht es die Einführung der Komponentenorientierung sogar, die jeweiligen Vorteile bei der Verwendung von Standard- bzw. Individual-Software im Rahmen eines sog. *Make and Buy* zu verbinden [KURBEL, et al. 1994].

Ungeachtet der vielfältigen Vorteile hat sich die Einführung der Komponentenorientierung in der (betrieblichen) Praxis der Anwendungsentwicklung jedoch als schwierig erwiesen. Zwar wird die komponentenorientierte Entwicklung heute durch ausgereifte Implementierungsplattformen (wie *OMG CORBA 3.0*, *Microsoft COM/.NET*, *Sun EJB* oder die *XML Web-Services Technologie*), eine akzeptierte einheitliche Terminologie [SZYPERSKI 1998; SZYPERSKI, et al. 2002], spezialisierte Entwurfstechniken [ALLEN und FROST 1998; D'SOUZA und WILLS 1999; CHEESMAN und DANIELS 2001] sowie eine angepasste Version der *Unified Modeling Language* (UML 2.0 [OMG 2003b]) unterstützt. Dennoch beschränken sich Erfolgsmeldungen aus der Praxis meist auf einzelne, anwendungsunabhängig einsetzbare (generische) Komponenten, wie bspw. Datenbank- und Workflow-Management-Systeme oder Elemente zur Gestaltung graphischer Benutzungsschnittstellen [GRIFFEL 1998:22; MAURER 2000]. Eine der ursprünglichen Vision entsprechende, durchgängig komponentenorientierte (betriebliche) Anwendungsentwicklung, die ausschließlich auf Komponenten als Entwicklungsobjekten aufsetzt und somit auch anwendungsspezifische Komponenten – sog. *Fachkomponenten* [TUROWSKI 2003] – verwendet, hat sich in der Praxis bislang noch nicht durchsetzen können.

1.1 Motivation und Problemstellung

Eine wesentliche Ursache für den ausbleibenden praktischen Erfolg der Komponentenorientierung besteht darin, dass der mit dem neuen Paradigma eingeführte Entwicklungsprozess noch nicht ausreichend methodisch unterstützt wird [GRIFFEL 1998:46]. Dieser Entwicklungsprozess unterscheidet grundsätzlich zwischen der *Anwendungsentwicklung*, bei der die Zerlegung einer Aufgabenstellung in Komponenten und deren anschließende Kom-

position zu einem Anwendungssystem im Mittelpunkt steht, sowie der *Komponentenentwicklung*, die die Bereitstellung der einzelnen (wieder verwendbaren) Komponenten zum Ziel hat. Dabei bestehen zahlreiche unbeantwortete Entwicklungsfragen, die einem Einsatz der Komponentenorientierung in der (betrieblichen) Praxis entgegenstehen [GARLAN, et al. 1995; MILI, et al. 1995; SAMETINGER 1997:15-18; BACHMANN, et al. 2000:7f., 43; WEYUKER 2001; CRNKOVIC 2002:132f.]: wie erfolgt die *Dekomposition einer Aufgabenstellung*; wie lassen sich die *Eigenschaften der einzelnen Komponenten* aus den Eigenschaften der zu entwickelnden Anwendung bestimmen; wie ist die *Eignung bzw. Kompatibilität* einzelner Komponenten zu ermitteln; welche *Verschiedenheiten (Heterogenitäten)* können zwischen Komponenten auftreten und wie ist mit diesen umzugehen; wie können die *Eigenschaften eines Anwendungssystems* auf Basis der Eigenschaften der einzelnen Komponenten vorhergesagt bzw. verifiziert werden; wie soll die *Komposition von Komponenten* erfolgen etc.? Da eine methodische Unterstützung, mit der diese Fragen beantwortet werden könnten, derzeit nicht vorhanden ist, droht bei der Einführung der Komponentenorientierung ein erneutes Auseinanderlaufen von Anspruch und Wirklichkeit – und damit die Entstehung einer neuen Krise, einer sog. „Modularitätskrise“ [HERRMANN, et al. 2001].

Um die Entstehung einer solchen Krise zu vermeiden und eine erfolgreiche Einführung der Komponentenorientierung in die (betriebliche) Praxis der Anwendungsentwicklung zu ermöglichen, ist es notwendig, eine speziell auf die Komponentenorientierung angepasste *Entwicklungsmethodik* zu schaffen [SPEED, et al. 2001; SZYPERSKI, et al. 2002:457]. Diese operationalisiert den komponentenorientierten Entwicklungsprozess durch die Vorgabe konkreter Regeln und Vorgehensweisen, die idealerweise als System *integrierter* (d.h. ineinander greifender und miteinander verträglicher) Methoden und Werkzeuge bereitgestellt werden [MEYER 1997:664; SZYPERSKI, et al. 2002:457; PAHL und BEITZ 2003:10f.]. Die zu schaffende Entwicklungsmethodik trägt wesentlich zur Entstehung einer an die komponentenorientierte Anwendungsentwicklung angepassten *Konstruktionslehre* [PAHL und BEITZ 2003] bei, deren Verfügbarkeit auch in den eingangs als Vorbilder genannten Ingenieursdisziplinen eine entscheidende Voraussetzung für den Erfolg der Komponentenorientierung darstellte [SHAW und GARLAN 1996:9f.; SPEED, et al. 2001].

Zum kritischen Erfolgsfaktor bei der Schaffung einer solchen Entwicklungsmethodik wird dabei vor allem die Einführung eines *Spezifikationsstandards*, mit dem sich diejenigen Eigenschaften von Komponenten, die für den Entwicklungsprozess relevant sind, umfassend und in abgestimmter Form (als Spezifikation) beschreiben lassen [GARLAN, et al. 1995; D'SOUZA und WILLS 1999:233; BROWN 2000:102; HERZUM und SIMS 2000:20; APPERLY, et al. 2001; CHEESMAN und DANIELS 2001:5; SPEED, et al. 2001; WEYUKER 2001; SZYPERSKI, et al. 2002:36, 50, 412f.; WALLNAU 2003:2, 10]. Mit solch einem Standard

lässt sich die *Effizienz des Entwicklungsprozesses* nachhaltig steigern, da Informationen über Komponenten, die im Rahmen verschiedener Arbeitsschritte zur Erfüllung der jeweiligen Aufgaben benötigt werden, prozessübergreifend aus einer einzigen Quelle – der *Komponentenspezifikation* – zur Verfügung gestellt werden können. Dies gilt sowohl für den Prozess der Komponentenentwicklung, bei dem Komponentenspezifikationen zur Anforderungsdefinition genutzt werden, als auch den Prozess der Anwendungsentwicklung, bei dem Komponentenspezifikationen ausgewertet werden, um Informationen über die miteinander zu verbindenden Komponenten zu erhalten.

Durch die Nutzung von Komponentenspezifikationen als Informationsquelle bei der Anwendungsentwicklung kann insbesondere der Einsatz alternativer Techniken zur Beschaffung von Informationen über Komponenten – wie bspw. von Methoden des Testens bzw. des Reverse Engineering [GARLAN, et al. 1995; WEYUKER 2001] – vermieden werden. Gerade der Einsatz solcher Methoden gilt als wesentliche Ursache für den ausbleibenden Erfolg der Komponentenorientierung, da durch sie ein erheblicher zusätzlicher Aufwand verursacht wird, der den mit der Einführung der Komponentenorientierung erzielten Effizienzgewinn rasch übersteigen kann. Außerdem sind die genannten Methoden meist nicht in der Lage, Informationen aus gekapselten Komponenten in der benötigten akkuraten Weise zu ermitteln [GARLAN, et al. 1995; WEYUKER 2001]. Ihr Einsatz setzt indes die Verfügbarkeit, also die Beschaffung bzw. Realisierung von Komponenten voraus. Zur Eignungsprüfung von Komponenten, bei der von einer Verfügbarkeit noch nicht ausgegangen werden kann, können sie deshalb schon aus diesem Grund nicht sinnvoll verwendet werden (zur ausführlichen Diskussion vgl. [OVERHAGE und THOMAS 2004:110f.]). Zu einer grundlegenden Verbesserung der Situation führt somit erst die Bereitstellung und Nutzung von Komponentenspezifikationen als Informationsquelle.

Die Einführung eines Spezifikationsstandards besitzt ferner eine ebenso wichtige Bedeutung für die *Schaffung von Methoden und Werkzeugen*, die einzelne Arbeitsschritte des Entwicklungsprozesses unterstützen sollen und dabei Informationen über die Komponenten verarbeiten. Für die Werkzeugentwicklung bildet der Spezifikationsstandard eine *gemeinsame methodische Grundlage*, die die zur Unterstützung eines Arbeitsschritts jeweils verwendbaren Informationen vorgibt (und dabei ggf. auch einschränkt). Obwohl dies die Flexibilität bei der Entwicklung von Methoden und Werkzeugen im Allgemeinen schmälert, ist die Vorgabe einer solchen gemeinsamen Grundlage von Vorteil, da sie die Entstehung einer *integrierten* Methodik fördert. Die Methoden und Werkzeuge einer solchen Methodik zeichnen sich dadurch aus, dass sie dieselben Spezifikationen verwenden und durch deren Austausch zudem direkt miteinander zusammenarbeiten können [HARMSEN, et al. 1994]. Die Vorgabe eines Spezifikationsstandards als Integrationsinstrument erlangt

insbesondere dann Bedeutung, wenn einzelne Methoden und Werkzeuge weitgehend unabhängig voneinander und von verschiedenen, spezialisierten Beteiligten entwickelt werden. In diesem Fall wird durch einen gemeinsam verwendeten Spezifikationsstandard die Homogenität der einzelnen Methoden und Werkzeuge gefördert und die Entstehung einer integrierten Methodik begünstigt, deren Bestandteile sich mit Bereitstellung *einer* Spezifikation nutzen lassen. Es ist also die Schaffung eines Spezifikationsstandards, die gleichermaßen eine Vorbedingung für eine *effiziente Entwicklung* und die Entstehung einer *integrierten Methodik* darstellt.

1.2 Thema und Beitrag

Das Thema dieser Arbeit ist die Darstellung des UnSCom (Unified Specification of Components) Spezifikationsrahmens, mit dem sich die Außensicht von Komponenten beschreiben lässt. Hierzu werden auf Basis eines allgemeingültigen Komponentenmodells in systematischer Weise die zentralen, zur Durchführung der Entwicklungsarbeit benötigten Eigenschaften von Komponenten bestimmt und Vorgaben für deren Beschreibung geschaffen. Dabei werden wissenschaftliche Erkenntnisse sowie bewährte Techniken aus der Informatik und Wirtschaftsinformatik, die für die Spezifikation von Komponenten bedeutsam sind, kombiniert und zu einem abgestimmten Rahmen integriert. Gleichzeitig baut der darzustellende Spezifikationsrahmen auf den Erfahrungen auf, die im Rahmen eines Projekts zur Vereinheitlichung der Spezifikation von Fachkomponenten [Ackermann, et al. 2002] gesammelt wurden und generalisiert dessen Ergebnisse bzw. ergänzt diese an zahlreichen Stellen um weitere Erkenntnisse. Dadurch werden vor allem die Anforderungen umgesetzt, die während einer Arbeitssitzung zum Thema „Spezifikation von Komponenten“ im Rahmen eines internationalen Workshops [Bosch, et al. 2003:8-10; Overhage 2003] identifiziert wurden.

Mit der Entwicklung des UnSCom Spezifikationsrahmens ist das Ziel verbunden, *Informationen über Komponenten*, die während vieler Arbeitsschritte, also *aufgabenübergreifend* benötigt werden, *umfassend* und *in abgestimmter Form* bereitzustellen. Damit leistet diese Arbeit einen wichtigen Beitrag zur mittelfristig anzustrebenden und im vorangehenden Abschnitt motivierten Schaffung eines *Spezifikationsstandards*. Da sich der UnSCom Spezifikationsrahmen jedoch auf die Beschreibung der Komponenteneigenschaften konzentriert, die sich aus einem allgemeingültigen Komponentenmodell (also weitgehend unabhängig von der Betrachtung konkreter Entwicklungsaufgaben) ableiten lassen, ist er ggf. um Vorgaben zur Beschreibung weiterer Komponenteneigenschaften zu erweitern, die zur Unterstützung spezieller Aufgaben benötigt werden. Deshalb wird mit dem UnSCom Spe-

zifikationsrahmen nur ein Fundament gelegt, das es bei der Schaffung eines Standards zu nutzen und – etwa im Rahmen eines Standardisierungsprojekts – entsprechend weiterzuentwickeln gilt. Darüber hinaus sind die im Rahmen dieser Arbeit dargestellten Vorgaben, die sich auf den *Inhalt* und die *Repräsentation* von Komponentenspezifikationen beziehen, ggf. um solche Vorgaben zu ergänzen, die die *Vorgehensweise,* d.h. die bei der Spezifikation einzusetzende(n) Methode(n) betreffen.

Jedoch sind gerade die durch den UNSCOM Spezifikationsrahmen berücksichtigten, aufgabenübergreifend nutzbaren Informationen über Komponenten in besonderer Weise geeignet, eine Vielzahl von Arbeitsschritten gleichzeitig methodisch zu unterstützen. Deshalb lässt sich durch seine Einführung die Effizienz des Entwicklungsprozesses bereits nachhaltig verbessern. Daneben bildet der UNSCOM Spezifikationsrahmen mit seiner aufgabenunabhängigen Konzeption und seinen primär inhaltlichen Vorgaben auch eine geeignete gemeinsame methodische Basis für die Entwicklung von Methoden und Werkzeugen, mit denen spezielle Aufgaben des Entwicklungsprozesses unterstützt werden sollen. Somit leistet der in dieser Arbeit dargestellte Spezifikationsrahmen den gewünschten Beitrag zur Erreichung der in Abschnitt 1.1 genannten Ziele und stellt einen wichtigen Schritt hin zu einer Methodik dar, die die komponentenorientierte (betriebliche) Anwendungsentwicklung umfassend unterstützt.

Die Darstellung des UNSCOM Spezifikationsrahmens folgt einem grundlegenden methodologischen Prinzip [MEYER 1997:665], nach dem die Formulierung einer Methode bzw. Konstruktion eines Werkzeugs stets auf einer fundierten theoretischen Begründung aufzubauen hat, und geschieht in zwei Schritten. Im Rahmen der *konzeptionellen Grundlegung* wird zunächst bestimmt, welche Eigenschaften von Komponenten (*was*) es mit einer Spezifikation zu beschreiben gilt. Hierzu werden die allgemeinen Funktionen, die Komponenten als Strukturierungseinheiten im Rahmen der Anwendungsentwicklung erfüllen, analysiert und allgemeingültige Definitionen für den Komponentenbegriff sowie seine wesentlichen Elemente entwickelt. Auf diesem Begriffsverständnis aufbauend werden anschließend systematisch die zentralen Eigenschaften von Komponenten, die in einer Spezifikation zu beschreiben sind, bestimmt. Im Wesentlichen entstehen bei der konzeptionellen Grundlegung folgende Beiträge (vgl. auch Abb. 1.1 oben):

- **Komponentenmodell.** Ein allgemeingültiges Komponentenmodell bildet die Basis für die Bestimmung der zu beschreibenden Eigenschaften von Komponenten und die Schaffung des Spezifikationsrahmens. Es konkretisiert das mit der Komponentenorientierung eingeführte neue Entwicklungsparadigma und beinhaltet Definitionen für dessen zentrale Konzepte: *Komponenten, Schnittstellen, Konnektoren und Kompositionen*. Zudem werden die allgemeinen Funktionen, die Komponenten als Strukturierungsein-

heiten erfüllen, durch geeignete Konzepte im Komponentenmodell abgebildet und folglich bei der Entwicklung des Spezifikationsrahmens berücksichtigt: die Funktion von Komponenten als *Abstraktionseinheiten* wird durch die Einführung spezieller Schnittstellenarten – sog. Angebots- und Nachfrageschnittstellen (Provided bzw. Required Interfaces) – als primären Beschreibungsobjekten abgebildet; die Funktion von Komponenten als *Analyseeinheiten* wird durch den Entwurf eines speziellen Komponententypsystems unterstützt, das die Durchführbarkeit statischer Analysen zur Entwicklungszeit gewährleistet; die Funktionen von Komponenten als *Entwurfs- und Kompositionseinheiten* werden schließlich durch eine Erweiterung des Konzepts des vertragsbasierten Entwurfs (Design by Contract [MEYER 1992; MEYER 1997:331-406]) im Komponentenmodell berücksichtigt, mit der sog. Dienst- und Komponentenverträge sowie parametrisierte Komponentenverträge eingeführt werden.

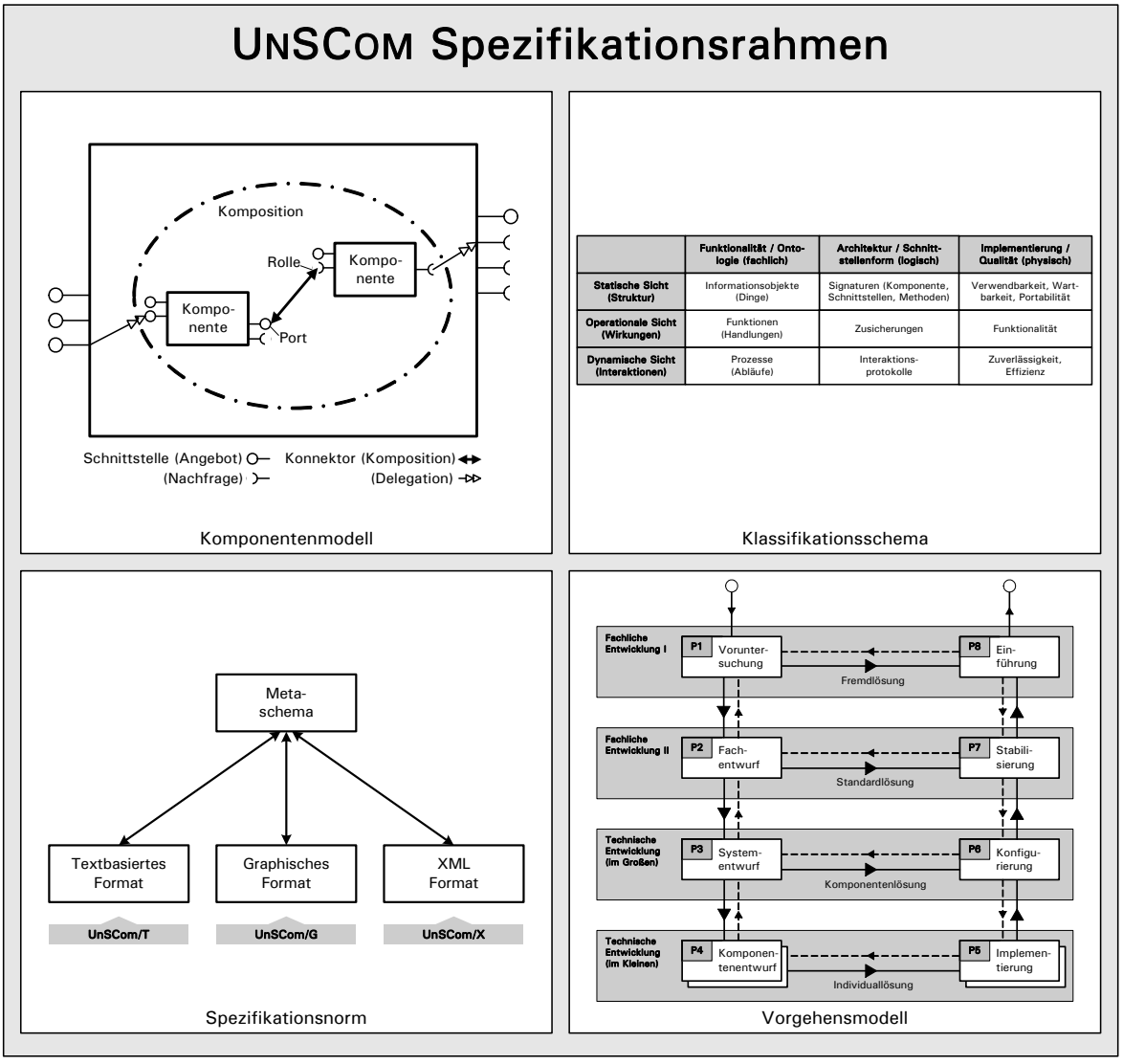


Abb. 1.1: Überblick über den UNSCOM Spezifikationsrahmen und seine vier wesentlichen Beiträge.

- **Klassifikationsschema.** Zur Bestimmung der zu beschreibenden Komponenteneigenschaften wird ein aspektorientiertes Klassifikationsschema eingesetzt. Es ermöglicht die umfassende und abgestimmte Identifikation aller Eigenschaften, die zur Durchführung der Entwicklungsarbeit benötigt werden. Um dieses Ziel zu erreichen, werden im Klassifikationsschema einerseits die drei Abstraktionsebenen des Entwicklungsprozesses und ihre jeweiligen Entwicklungsschwerpunkte berücksichtigt: die *fachliche Abstraktionsebene* fokussiert auf die fachliche Funktionalität, die *logische Abstraktionsebene* auf die Software-Architektur (Form) und die *physische Abstraktionsebene* schließlich auf die nicht-funktionalen (Qualitäts-) Eigenschaften einer Komponente. Orthogonal dazu werden ferner die verschiedenen Sichten (Perspektiven) des Komponentenmodells, die zur vollständigen Beschreibung eines Software-Artefakts einzunehmen sind, unterschieden: die *statische Sicht* betrachtet die Struktur, die *operationale Sicht* die Wirkungen und die *dynamische Sicht* schließlich die Interaktionen einer Komponente. Durch eine Gegenüberstellung der orthogonalen Aspekte „Abstraktionsebene“ und „Beschreibungsperspektive“ werden neun Klassen von Komponenteneigenschaften als zu beschreibende Eigenschaften identifiziert und anschließend weiter differenziert.

Aufbauend auf der konzeptionellen Grundlegung werden in einem zweiten Schritt die methodischen Vorgaben des UNSCOM Spezifikationsrahmens beschrieben. Diese bauen auf den im Rahmen der Grundlegung entwickelten Konzepten auf und legen fest, auf welche Weise (*wie*) die dort identifizierten Komponenteneigenschaften zu beschreiben sind. Hierbei werden Vorgaben dargestellt, mit denen sowohl die inhaltliche Struktur von Komponentenspezifikationen als auch die zur Repräsentation einzusetzenden Notationen normiert werden. Darüber hinaus wird der zeitliche Ablauf einer Komponentenspezifikation (der Spezifikationsprozess) beschrieben und in den allgemeinen komponentenorientierten Entwicklungsprozess eingebettet. Im Einzelnen bestehen die methodischen Vorgaben des UNSCOM Spezifikationsrahmens dabei aus den folgenden Beiträgen (vgl. Abb. 1.1):

- **Spezifikationsnorm.** Eine Reihe verbindlicher Vorgaben regelt die Beschreibung der zuvor identifizierten Komponenteneigenschaften im Rahmen einer Komponentenspezifikation. Als Bestandteil dieser Vorgaben wird zunächst ein Metaschema entwickelt, das die inhaltliche Struktur der zu beschreibenden Komponenteneigenschaften festlegt und diese zu einem Gesamtsystem, der Komponentenspezifikation, integriert. Aufbauend auf dem Metaschema werden anschließend drei Formate zur Repräsentation einer Komponentenspezifikation beschrieben. Diese bestehen vorzugsweise aus etablierten, bewährten Notationen und ermöglichen eine an verschiedene Aufgabenstellungen und Zielgruppen angepasste Darstellung von Komponentenspezifikationen: als Spezifikati-

onssprache für den Programmierer wird ein *textbasiertes Format* (UNSCOM/T) bereitgestellt; als Diagrammsprache für den Analytiker bzw. Architekten wird ein *graphisches Format* basierend auf den Notationen der UML 2.0 (UNSCOM/G) eingeführt; als Format zur internen Repräsentation von Komponentenspezifikationen in Werkzeugen und zum Austausch zwischen Werkzeugen wird zudem ein *XML Format* (UNSCOM/X) entwickelt. Jedes Repräsentationsformat ist dabei durch das zugrunde liegende gemeinsame Metaschema integriert, so dass eine effiziente Übersetzung zwischen verschiedenen Formaten aufgrund ihrer Strukturgleichheit gewährleistet ist.

- **Vorgehensmodell.** Der Spezifikationsprozess wird unter Rückgriff auf ein ergebnisorientiertes Vorgehensmodell beschrieben und in den komponentenorientierten Entwicklungsprozess eingeordnet. Dabei werden die einzelnen *Teile* einer Komponentenspezifikation verschiedenen *Phasen* des Entwicklungsprozesses als Ergebnisse zugeordnet. Mit dem Spezifikationsprozess wird die Anwendung des UNSCOM Spezifikationsrahmens beschrieben und zugleich ein grober *zeitlicher Ablauf* für die Erstellung von Komponentenspezifikationen skizziert. Die Ausführungen zum Spezifikationsprozess dienen dabei auch dem Ziel, die Komplexität der Spezifikationsaufgabe zu reduzieren. Allerdings werden Angaben zur konkreten methodischen Vorgehensweise bei der Spezifikation aufgrund ihrer meist geringeren Allgemeingültigkeit weitgehend ausgeklammert. Stattdessen wird fallweise auf Methoden verwiesen, die von bereits existierenden Methodiken bereitgestellt werden und bei der Spezifikation eingesetzt werden können. Über den Spezifikationsprozess hinaus wird in dieser Arbeit auch auf die verschiedenen Einsatzmöglichkeiten für (Teile der) Komponentenspezifikationen zur Lösung von Aufgaben eingegangen, die sich bei der komponentenorientierten Entwicklung ergeben. Die beschriebene Anwendung des UNSCOM Spezifikationsrahmens wird somit auch auf die Bewältigung konkreter Entwicklungsaufgaben ausgedehnt.

1.3 Aufbau und Einordnung

Der Aufbau dieser Arbeit orientiert sich an der deduktiv-theoriekritischen Vorgehensweise [EBERHARD 1999] zur Ableitung einer Lösung für ein bestehendes Problem (bzw. einen erkannten Mangel) und beinhaltet die folgenden Teile: Gegenstandsbestimmung mit Problemanalyse und Lösungsvorschlag (Kapitel 2), Entwicklung von Lösungsgrundlagen (Kapitel 3) und Ausarbeitung des Lösungsansatzes (Kapitel 4). Die Arbeit endet mit einer Schlussbetrachtung, in deren Rahmen das Erreichte logisch überprüft und ein Ausblick auf noch verbleibende Aufgabenfelder gegeben wird (Kapitel 5). Ausgehend von diesem grundsätzlichen Aufbau gliedert sich die Arbeit wie folgt:

Kapitel 2 ist der Bestimmung des *Gegenstandsbereichs* gewidmet und beschreibt zunächst die Entstehung des *komponentenorientierten Entwicklungsparadigmas* sowie die allgemeinen Funktionen, die Komponenten als Strukturierungseinheiten erfüllen (Abschnitt 2.1). Darüber hinaus werden der *komponentenorientierte Entwicklungsprozess* und die neuartigen Entwicklungsaufgaben, die es im Rahmen einer spezialisierten *Methodik* zu unterstützen gilt, dargestellt (Abschnitt 2.2). Dabei wird vor allem die zentrale Bedeutung eines Spezifikationsrahmens zur Beschreibung von Komponenten hervorgehoben. Ausgehend von den zu unterstützenden Aufgaben wird auf die *grundlegenden Anforderungen*, die von einem Spezifikationsrahmen zu erfüllen sind, näher eingegangen (Abschnitt 2.3). Eine *Untersuchung bestehender Ansätze* und ihrer Defizite liefert schließlich die Begründung für die Konzeption und nachfolgende Darstellung des UNSCOM Spezifikationsrahmens. Das Kapitel endet mit der Beschreibung einer *Fallstudie*, mit der die Anwendung des UNSCOM Spezifikationsrahmens beispielhaft demonstriert wird (Abschnitt 2.4), und einigen notationstechnischen Konventionen (Abschnitt 2.5).

Kapitel 3 beinhaltet die *konzeptionelle Grundlegung*, auf der die später folgende Darstellung des UNSCOM Spezifikationsrahmens aufbaut. Im Rahmen der Grundlegung wird zunächst auf geeignete *Konzepte zur Beschreibung* von Komponenten eingegangen. Im Anschluss daran werden die *Eigenschaften von Komponenten* identifiziert, die für die Durchführung der Entwicklungsarbeit benötigt werden und deshalb zu beschreiben sind. Zu Beginn wird ein *allgemeingültiges Komponentenmodell* erarbeitet, mit dem das komponentenorientierte Entwicklungsparadigma konkretisiert wird (Abschnitt 3.1). Um die allgemeinen Funktionen, die Komponenten als Strukturierungseinheiten erfüllen, vollständig in das Modell abzubilden, werden anschließend Aspekte eines *Komponententypsystems* skizziert (Abschnitt 3.2) und das *Konzept des vertragsbasierten Entwurfs* auf die Beschreibung von Komponenten angepasst (Abschnitt 3.3). Auf der Basis des so erweiterten Komponentenmodells wird schließlich ein *Klassifikationsschema* entwickelt, mit dem die zur Durchführung der Entwicklungsarbeit benötigten Eigenschaften von Komponenten systematisch in Klassen unterteilt und in einen Gesamtzusammenhang eingeordnet werden (Abschnitt 3.4).

Kapitel 4 beschreibt den *UNSCOM Spezifikationsrahmen*, der auf der konzeptionellen Grundlegung aufbaut und sowohl den Inhalt als auch das Format von Komponentenspezifikationen normiert. Dabei wird zunächst der allgemeine *Aufbau* des Spezifikationsrahmens dargestellt, der sich an dem zuvor entwickelten Klassifikationsschema orientiert und jeder Klasse von Komponenteneigenschaften eine Spezifikationsebene zuordnet (Abschnitt 4.1). Gleichzeitig werden verschiedene Arten von Vorgaben unter-

schieden, die sich auf den *Inhalt*, die *Repräsentation* bzw. die *Geltung* von Spezifikationsteilen beziehen. Im Anschluss daran werden dann die Vorgaben für die verschiedenen Spezifikationsteile im Detail betrachtet. Dabei wird zunächst auf die Spezifikation *allgemeiner und kommerzieller Informationen* (Abschnitt 4.2) sowie auf die *Klassierung* von Komponenten (Abschnitt 4.3) eingegangen. Danach wird die Spezifikation der *fachlichen Funktionalität* (Abschnitt 4.4), der *architekturspezifischen Merkmale* (Abschnitt 4.5) sowie der *Qualitätseigenschaften* (Abschnitt 4.6) dargestellt. Am Ende des Kapitels wird auf die *Anwendung* des UNSCOM Spezifikationsrahmens während des Entwicklungsprozesses eingegangen (Abschnitt 4.7). Dabei wird zu Illustrationszwecken auf die in der Gegenstandsbestimmung dargestellte Fallstudie zurückgegriffen.

Kapitel 5 schließt die Arbeit mit einer *Zusammenfassung* der einzelnen Beiträge (Abschnitt 5.1) ab und unterzieht die dargestellten Konzepte einer *kritischen Überprüfung* (Abschnitt 5.2). In einem *Ausblick* werden noch zu lösende Fragestellungen benannt und zukünftig anzustrebende Entwicklungen beschrieben, die auf dem UNSCOM Spezifikationsrahmen aufsetzen (Abschnitt 5.3). Dabei wird auch auf die Schaffung einer umfassenden Methodik zur Unterstützung der komponentenorientierten Anwendungsentwicklung eingegangen, die bereits in der Einleitung angesprochen wurde.

Anhang A beinhaltet beispielhaft die Spezifikation einer Komponente aus der Fallstudie, die im UNSCOM/T Format dargestellt ist.

Anhang B beinhaltet die XML Schemata, die bei der Entwicklung des UNSCOM/X Formats definiert wurden.

Anhang C beinhaltet die Taxonomien, die bei der Entwicklung des UNSCOM Spezifikationsrahmens zur Klassierung definiert wurden.

Aus wissenschaftlicher Sicht wird mit der Entwicklung des UNSCOM Spezifikationsrahmens zum einen ein Beitrag zum informatischen Forschungsgebiet des *Software Engineering* (dt. Software-Technik) geleistet, das sich mit der ingenieurmäßigen Entwicklung – d.h. dem Entwurf und der Implementierung – von Software sowie den dazu notwendigen Methoden und Techniken befasst [NAUR und RANDELL 1969; SOMMERVILLE 1992; BROY und ROMBACH 2002]. Mit der Entwicklung des Spezifikationsrahmens bzw. der methodischen Vorgaben zur Beschreibung von Komponenten werden gemäß einer Klassifikation von Forschungsgebieten der Informatik, die von einer Projektgruppe im Auftrag der ACM entwickelt wurde [DENNING, et al. 1989], im Wesentlichen Beiträge zu den Teilbereichen *Abstraktion* und *Entwurf* geleistet. Dabei tragen das entwickelte Komponentenmodell, das

Klassifikationsschema, die aus ihm abgeleiteten inhaltlichen Vorgaben des Spezifikationsrahmens sowie das zur Beschreibung des Spezifikationsprozesses entwickelte Vorgehensmodell zum Teilgebiet der Abstraktion bei. Die Vorgaben bezüglich des Formats von Komponentenspezifikationen sind dagegen wegen ihrer unmittelbaren praktischen Anwendbarkeit zum Teilgebiet des Entwurfs zu zählen [DENNING, et al. 1989:68f.].

Zum anderen trägt die Arbeit mit der Entwicklung eines einheitlichen Ansatzes zur Spezifikation von Komponenten, der sowohl fachliche als auch technische Erfordernisse berücksichtigt, auch dazu bei, die Komponentenorientierung als einen eigenständigen Methodikansatz in die *Systemplanung und -entwicklung* [HEINRICH 1996:25-28] einzuführen. Die Systemplanung und -entwicklung befasst sich schwerpunktmäßig mit der Planung und Realisierung betrieblicher Anwendungssysteme als soziotechnische [ROPOHL 1999:135-150] bzw. Mensch-Aufgabe-Technik-Systeme [HEINRICH 1993:13f.] und stellt ein zentrales Forschungsgebiet der *Wirtschaftsinformatik* dar [HEINRICH 1994; HEINRICH 1996]. Mit ihrem Thema besitzt diese Arbeit dabei einen engen Bezug zur *Modellierung* bzw. zur *Referenzmodellierung*, die wichtige Teilbereiche des Forschungsgebiets der Systemplanung und -entwicklung darstellen. Zudem wird durch die nähere Betrachtung einer auf die komponentenorientierte Entwicklung angepassten Methodik auch der Änderungsbedarf beschrieben, der sich durch die Einführung der Komponentenorientierung im Vergleich zur traditionellen Systemplanung und -entwicklung ergibt.

Entsprechend der heute in der Wissenschaft vorherrschenden deduktiv-theoriekritischen Vorgehensweise, die auch der Entwicklung des UNSCOM Spezifikationsrahmens zugrunde liegt, ist jede für ein zuvor identifiziertes Problem entwickelte Lösung neben einer *logischen* jedoch auch einer *empirischen* Überprüfung zu unterziehen [EBERHARD 1999]. Aufgrund des speziellen Gegenstands dieser Arbeit ergeben sich dabei vor allem im Hinblick auf eine empirische Überprüfung der Frage, inwieweit die erreichten Ergebnisse tatsächlich zur Erreichung der in Abschnitt 1.2 genannten Ziele beitragen, einige Einschränkungen. So lässt sich anhand von Fallstudien zwar allgemein eine Effizienzsteigerung des Entwicklungsprozesses durch die Einführung von Spezifikationstechniken [SOMMERVILLE 1992:126] und auch ein Fortschritt bei der Entwicklung einzelner Methoden und Werkzeuge belegen [CRNKOVIC 2002:129]. Jedoch können üblicherweise weder die dokumentierten Effizienzsteigerungen noch die bei der Entwicklung von Methoden und Werkzeugen erzielten Erfolge wissenschaftlich eindeutig, d.h. zweifelsfrei und exakt messbar, auf den Einfluss der jeweils entwickelten Spezifikationskonzepte zurückgeführt werden.

Für die mangelnde empirische Überprüfbarkeit gibt es vor allem zwei Gründe. Zum einen beziehen sich die zu überprüfenden Konzepte auf Strategien bzw. Vorgehensweisen und stellen somit Abstraktionen dar, die stets einer praktischen Anwendung während des Ent-

wicklungsprozesses bedürfen. Der Einfluss der entwickelten Konzepte lässt sich also grundsätzlich nicht direkt, sondern nur *mittelbar* bei ihrer Anwendung überprüfen. Zum anderen wird der Einfluss der entwickelten Konzepte gerade bei der praktischen Anwendung von der Individualität der Entwickler und der Kontextabhängigkeit der Entwicklungsprojekte *überlagert*. Somit treten bei einer empirischen Überprüfung zwangsläufig zahlreiche nicht oder nur mit hohem Aufwand kontrollierbare Variablen auf, die die Aussagekraft der Ergebnisse in Zweifel ziehen. An die Stelle empirischer Überprüfungen, in denen Prüfhypothesen zu formulieren und im Rahmen einer Studie zu verifizieren sind, tritt bei der Erforschung neuer Methoden und Prozesse im Software Engineering bzw. der betrieblichen Anwendungsentwicklung aus diesem Grund häufig eine stärker *analytische* Überprüfung. Dabei wird im Rahmen einer Plausibilitätsprüfung üblicherweise gezeigt, dass neu entwickelte Konzepte entweder zur Lösung bislang nicht handhabbarer Problemklassen beitragen (sog. *Proof-of-Existence*) oder zu besser verwendbaren Lösungen für bereits handhabbare Problemklassen führen als bislang verwendete Ansätze (sog. *Proof-of-Concept*).

Im Rahmen dieser Arbeit erfolgt eine analytische Überprüfung des UNSCOM Spezifikationsrahmens im Rahmen eines *Proof-of-Concept*, mit dem einerseits seine *Anwendbarkeit* im Rahmen des Entwicklungsprozesses und andererseits seine *Überlegenheit* im Vergleich mit existierenden Ansätzen belegt wird. Hierzu wird in der Arbeit zum einen die Spezifikation von Komponenten anhand einer komplexen *Fallstudie* im Detail illustriert (Abschnitt 4.7) und zum anderen ein *Vergleich* mit existierenden Methoden durchgeführt (Abschnitt 2.3.2). Darüber hinaus wird explizit auf *Verwendungsmöglichkeiten* der erstellten Spezifikationen zur Lösung von konkreten Aufgaben während des Entwicklungsprozesses eingegangen (Abschnitt 2.2.3).

2 Gegenstandsbestimmung

In diesem Kapitel werden die Voraussetzungen für eine vertiefende Beschäftigung mit dem Gegenstand dieser Arbeit, der Darstellung des UNSCOM Spezifikationsrahmens, geschaffen. Hierzu wird im Rahmen einer historischen Betrachtung in Abschnitt 2.1 zunächst die Entstehung des modernen komponentenorientierten Entwicklungsparadigmas der (Wirtschafts-) Informatik beschrieben. Dabei werden auch das strukturierte und das objektorientierte Entwicklungsparadigma analysiert, deren jeweils zentrale Konzepte die Komponentenorientierung im Rahmen einer kontinuierlichen Fortentwicklung entscheidend mitgeprägt haben [CHEESMAN und DANIELS 2001:2f.]. Die vorgenommene Betrachtung dient vor allem der Abgrenzung des heute in der (Wirtschafts-) Informatik vorherrschenden Komponentenbegriffs, der für die Konzeption des UNSCOM Spezifikationsrahmens von zentraler Bedeutung ist. Zudem werden im Rahmen der Betrachtung die allgemeinen Funktionen von Komponenten als Strukturierungseinheiten, die bei der Konzeption des Spezifikationsrahmens zu berücksichtigen sind, identifiziert und erläutert.

Aufbauend auf einer Analyse des komponentenorientierten Entwicklungsprozesses und der in seinem Rahmen zu bewältigenden neuen Aufgaben wird anschließend in Abschnitt 2.2 näher auf die Bestandteile einer spezialisierten Methodik zur Unterstützung des Entwicklungsprozesses sowie auf die Bedeutung eines Spezifikationsrahmens als ein wesentliches Element einer solchen Methodik eingegangen. Aus den beschriebenen Anwendungen des Spezifikationsrahmens im Rahmen des Entwicklungsprozesses werden in Abschnitt 2.3 die von Komponentenspezifikationen zu erfüllenden grundlegenden Anforderungen abgeleitet und die mit der Konzeption des UNSCOM Spezifikationsrahmens verfolgten Ziele beschrieben. In diesem Zusammenhang wird auch eine kritische Betrachtung bereits existierender Ansätze zur Spezifikation von Komponenten durchgeführt, deren Defizite die Neukonzeption des UNSCOM Spezifikationsrahmens begründen. Mit der Beschreibung von Lösungsansätzen zur Beseitigung der erkannten Defizite wird schließlich zum Schwerpunkt dieser Arbeit, der Darstellung des UNSCOM Spezifikationsrahmens, übergeleitet.

2.1 Entwicklungsparadigmen

Die Herausbildung des heutigen komponentenorientierten Entwicklungsparadigmas in der (betrieblichen) Anwendungsentwicklung [SZYPERSKI 1998; HERZUM und SIMS 2000; SZYPERSKI, et al. 2002; TUROWSKI 2003] ist das Resultat eines Evolutionsprozesses, in dessen

Verlauf die Entwicklungsarbeit hauptsächlich durch unterschiedliche, einander in ihrer Bedeutung für die Strukturierung von Anwendungssystemen jeweils ablösende Paradigmen[1] geprägt wurde. Die fortwährende Herausbildung neuer Entwicklungsparadigmen ist dabei vor allem ein Ausdruck der Suche nach geeigneten Konzepten (Einheiten) für die Strukturierung von Anwendungssystemen in abtrennbare, d.h. unabhängige und weniger komplexe Teile, mit denen die in der Einleitung genannten Vorteile einer modularen Anwendungsentwicklung realisiert werden können [MEYER 1997:39f.].

Die Entstehung der verschiedenen Entwicklungsparadigmen setzte vor allem im Anschluss an die Software-Engineering-Konferenz der NATO ein, die im Jahre 1968 in Garmisch stattfand. Dort wurde mit dem erstmals formulierten Bestreben, die (betriebliche) Anwendungsentwicklung zu einer Ingenieursdisziplin fortzuentwickeln, die bis heute andauernde Schaffung einer umfassenden, theoretisch abgesicherten Methodik für die Entwicklung großer Systeme eingeleitet [NAUR und RANDELL 1969]. Im Zuge dieser Bestrebung haben sich seit Ende der 1960er Jahre im Wesentlichen drei Entwicklungsparadigmen herausgebildet, die sich in Bezug auf die ihnen zugrunde liegenden Konzepte (Einheiten) für die Strukturierung von Anwendungssystemen unterscheiden (vgl. Abb. 2.1): das *strukturierte Entwicklungsparadigma*, das *objektorientierte Entwicklungsparadigma* und schließlich das heutige *komponentenorientierte Entwicklungsparadigma* [HERZUM und SIMS 2000:16f.].

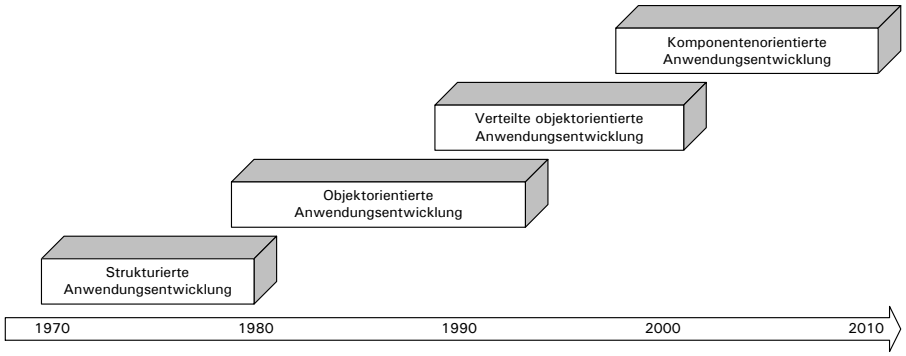


Abb. 2.1: Evolution der Entwicklungsparadigmen (in Anlehnung an [HERZUM und SIMS 2000:17]).

Das strukturierte Entwicklungsparadigma überträgt die Computer-Architektur von Neumanns auf die Gestaltung von Anwendungssystemen und verwendet das Prinzip der getrennten Betrachtung von Funktionen (Programmablauf) und Daten (Speicherung) als Grundlage für die Strukturierung von Anwendungssystemen [BACKUS 1978]. Die Struktu-

[1] Der Begriff „Paradigma“ bezeichnet im Allgemeinen die in einem bestimmten Gegenstandsgebiet jeweils geltenden methodologischen Prinzipien sowie dessen gemeinschaftsstiftende Theorie [KUHN 1976].

rierung erfolgt im Rahmen einer schrittweisen Verfeinerung der zu erfüllenden Aufgabe, d.h. in einem analytischen Top-Down Prozess. Dabei wird typischerweise die entstehende *Funktionsstruktur* [WIRTH 1971:226] als Strukturierungskriterium verwendet. Dagegen basiert das objektorientierte Entwicklungsparadigma auf dem Prinzip der Integration zusammengehöriger Funktionen und Daten zu *Klassen*, die die Einheiten für die Strukturierung von Anwendungssystemen bilden [MEYER 1997:146f.]. Die Strukturierung erfolgt im objektorientierten Paradigma ausgehend von den Klassen, die im Rahmen einer Untersuchung des Anwendungsbereichs identifiziert und in einem synthetischen Bottom-Up Prozess zu einem Gesamtsystem verbunden werden [MEYER 1997:114]. Das komponentenorientierte Entwicklungsparadigma basiert ebenfalls auf dem Prinzip der Integration zusammengehöriger Funktionen und Daten [CHEESMAN und DANIELS 2001:2]. Im Gegensatz zur Objektorientierung stellt es jedoch die Strukturierung von Anwendungssystemen in möglichst abgeschlossene (wieder verwendbare) Einheiten in den Vordergrund [SZYPERSKI, et al. 2002:3]. Um dieses Ziel zu erreichen, trennt die Komponentenorientierung zwischen der Entwicklung von (wieder verwendbaren) *Komponenten*, der sog. Entwicklung im Kleinen, sowie der Entwicklung von Anwendungssystemen unter Verwendung von (vorgefertigten) Komponenten, der sog. Entwicklung im Großen [SAMETINGER 1997:158; GRIFFEL 1998:46]. Der so entstehende, zweigeteilte Entwicklungsprozess besitzt dabei in der Regel sowohl typische Eigenschaften eines analytischen Top-Down als auch eines synthetischen Bottom-Up Prozesses.

2.1.1 Modularitätskriterien

Die Vorteile einer modularen Anwendungsentwicklung (vgl. Abb. 2.2), die durch den Einsatz eines bestimmten Entwicklungsparadigmas realisiert werden sollen, lassen sich im Allgemeinen nur dann erreichen, wenn das jeweilige Strukturierungskonzept einigen grundlegenden (technischen) Anforderungen genügt, die auch als *Modularitätskriterien* bezeichnet werden [MEYER 1997:39-46]. Mit diesen Modularitätskriterien wird vor allem eine die Komplexität reduzierende Zerlegung von Problemstellungen in jeweils einfacher zu handhabende Teile, die möglichst uneingeschränkte Zusammenfügbarkeit einzelner Teile, die unabhängige Analysierbarkeit von Teilen, die Gewährleistung der Kontinuität des entstehenden Gesamtssystems sowie ein kontrollierbares Systemverhalten beim Auftreten eines Fehlers innerhalb eines Teils gefordert [MEYER 1997:40]. Um diesen Anforderungen zu genügen, müssen die bei der Strukturierung entstehenden Teile als Strukturierungseinheiten die *allgemeinen Funktionen* von Entwurfs-, Kompositions-, Analyse- und Abstraktionseinheiten erfüllen, die wegen ihrer Bedeutung zur Bewertung von Entwicklungsparadigmen im Folgenden genauer erläutert werden.

Bei der Strukturierung einer Problemstellung sollten idealerweise solche Teile entstehen, die sich unabhängig voneinander, d.h. als eigenständige *Entwurfseinheiten*, und mit geringerem Aufwand als die ursprüngliche Problemstellung weiter bearbeiten lassen [PARNAS 1972:1054; MEYER 1997:40f.]. Die zunächst geforderte Unabhängigkeit im Hinblick auf die weitere Entwicklung ist sowohl eine wichtige Voraussetzung für die angestrebte *Parallelentwicklung* einzelner Teile als auch die *inkrementelle Anwendungsentwicklung*, die beide wesentlich zur Verkürzung der Entwicklungszeit sowie zur Flexibilisierung des Entwicklungsprozesses beitragen [BROWN 2000:74f.]. Die Forderung nach einem Mechanismus zur Komplexitätsreduktion, der den Strukturierungsprozess begleitet, fördert dagegen vor allem die *methodische Dekomposition* von Entwicklungsaufgaben und dient der Beherrschung der Anwendungssystemkomplexität im Allgemeinen [BROWN 2000:74].

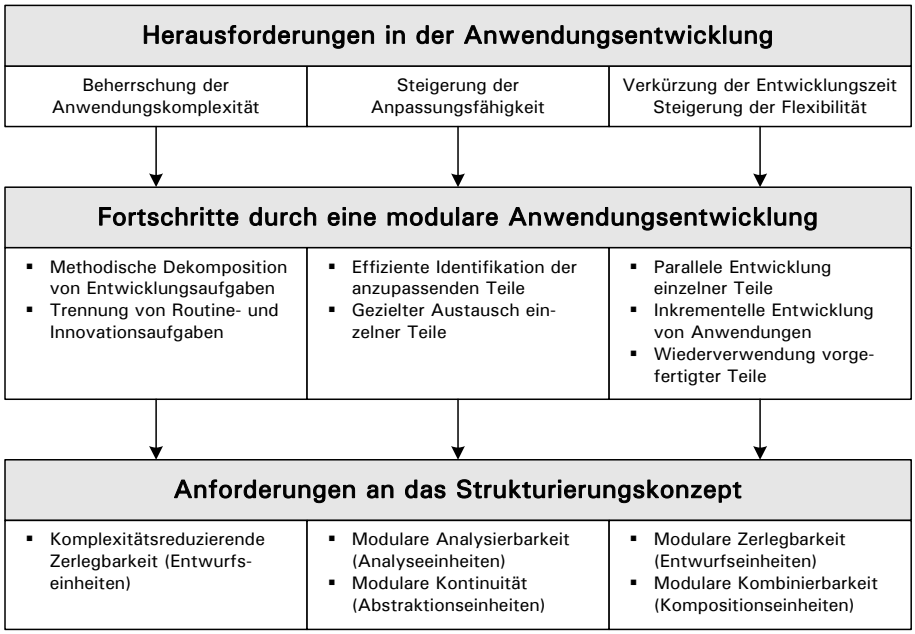


Abb. 2.2: Zielsystem zur Bewertung der Strukturierungskonzepte verschiedener Entwicklungsparadigmen.

Weiterhin ist zu gewährleisten, dass die bei der Strukturierung eines Anwendungssystems identifizierten Teile als *Kompositionseinheiten* möglichst ohne Einschränkungen mit anderen Teilen kombiniert werden können [MEYER 1997:42f.; SZYPERSKI, et al. 2002:143]. Das bedeutet, dass einzelne Teile insbesondere auch in solchen Kombinationen mit anderen Teilen einsetzbar sein sollten, die während der Entwicklung zunächst nicht explizit vorhergesehen und berücksichtigt wurden [TUROWSKI 2003:19]. Die möglichst uneingeschränkte Kombinierbarkeit ist vor allem eine Voraussetzung für die Einführung eines Konzepts zur *Wiederverwendung* einzelner Teile während der Entwicklung von Anwendungssystemen, das eine wirksame Maßnahme zur Verkürzung der Entwicklungszeit und zur Steigerung der Qualität von Anwendungssystemen darstellt [LIM 1994; BROWN 2000:74].

Die einzelnen Teile eines Anwendungssystems sollten darüber hinaus möglichst unabhängig voneinander, also idealerweise als selbstständige *Analyseeinheiten*, betrachtet und auf ihre Eigenschaften hin analysiert werden können, ohne dass hierfür weitere Teile berücksichtigt und einbezogen werden müssen [MEYER 1997:43f.]. Durch die Forderung nach Unabhängigkeit der einzelnen Teile im Hinblick auf ihre Untersuchung und Beurteilung wird einerseits die angestrebte *Wiederverwendung* von Teilen unterstützt, da die unabhängige Analysierbarkeit eine wichtige Grundlage für die effiziente Auswahl von Teilen darstellt. Andererseits leistet die unabhängige Analysierbarkeit auch einen Beitrag zur angestrebten Steigerung der Anpassungsfähigkeit von Anwendungssystemen, da sie die *Identifikation* der (im Falle einer Änderung) anzupassenden Teile erleichtert [MEYER 1997:43; SZYPERSKI, et al. 2002:141f.].

	Funktionen	Module	Klassen	Komponenten
Entwurfseinheiten	o	+	+	+
Kompositionseinheiten	-	-	+	+
Analyseeinheiten	-	+	o	+
Abstraktionseinheiten	-	o	-	+

Legende: + gut unterstützt, o unterstützt, - nicht unterstützt

Abb. 2.3: Unterstützung der Modularitätskriterien durch die verschiedenen Strukturierungskonzepte.

Schließlich ist zu fordern, dass die während der Strukturierung eines Anwendungssystems entstehenden Teile als *Abstraktionseinheiten* einerseits im Hinblick auf die von ihnen bereitgestellte fachliche Funktionalität abgeschlossen sind [FELLNER, et al. 1999:34; HERZUM und SIMS 2000:444; SZYPERSKI, et al. 2002:140f.] und andererseits die bei ihrer jeweiligen Entwicklung getroffenen Entwurfs- bzw. Implementierungsentscheidungen sowie die bei der Ausführung auftretenden Fehlerzustände kapseln [MEYER 1997:44-46]. Diese Forderung zielt vor allem auf eine Verbesserung der *Austauschbarkeit* einzelner Teile, so dass im Falle von Änderungen jeweils möglichst wenige Teile betroffen sind und die Kontinuität des Gesamtsystems gewährleistet bleibt [MEYER 1997:44]. Sie ist eine wesentliche Voraussetzung für die Gewährleistung der effizienten Anpassbarkeit von Anwendungssystemen an technische und fachliche Änderungen [CHEESMAN und DANIELS 2001:2].

Aufgrund ihrer jeweils unterschiedlichen Strukturierungskonzepte sind die gemäß der einzelnen Entwicklungsparadigmen entstehenden Teile jedoch nicht alle gleichermaßen dazu geeignet, die genannten allgemeinen Funktionen zu erfüllen. Vielmehr zeigen sich signifikante Unterschiede (vgl. Abb. 2.3), die im Folgenden genauer untersucht werden. Damit lässt sich die fortwährende Entstehung neuer Paradigmen auch auf eine jeweils ungenügende Erfüllung der Modularitätskriterien durch die bestehenden Ansätze zurückführen.

2.1.2 Strukturierte Entwicklung

Das strukturierte Entwicklungsparadigma ist zunächst als ein Implementierungskonzept entstanden und hat sich im Laufe der Zeit zu einem umfassenden Konzept für den Entwurf und die Strukturierung von Anwendungssystemen weiterentwickelt. Grundlegend für die Entstehung des strukturierten Entwicklungsparadigmas war die Erkenntnis, dass sich jeder Algorithmus durch die (wiederholte und verschachtelte) Verwendung der drei Programmierkonstrukte *Sequenz*, *Iteration* und *bedingte Verzweigung* beschreiben lässt [BÖHM und JACOPINI 1966] sowie die Entstehung der imperativen Programmiersprachen, mit denen sich Anwendungssysteme so lange in jeweils kleinere, weniger komplexe Funktionen unterteilen lassen, bis diese schließlich effizient implementiert werden können [DIJKSTRA 1972; MEYER 1997:45]. Das strukturierte Entwicklungsparadigma greift diese Erkenntnisse auf und entwickelt hieraus ein Konzept für die Zerlegung von Anwendungssystemen. Dabei ist vor allem zwischen einer funktionsorientierten und einer modulorientierten Ausrichtung des strukturierten Entwicklungsparadigmas zu unterscheiden.

Die klassische, *funktionsorientierte Ausrichtung* des strukturierten Entwicklungsparadigmas nutzt für die Zerlegung von Anwendungssystemen ausschließlich das Konzept der *schrittweisen Verfeinerung* ihrer jeweiligen Aufgabenstruktur [WIRTH 1971] und betrachtet die dabei entstehenden *Funktionen* als geeignete Strukturierungseinheiten. Somit erfolgt die Strukturierung eines Anwendungssystems im Rahmen eines Dekompositionsprozesses, während dem die einzelnen, von einem Anwendungssystem zu erfüllenden Aufgaben schrittweise in Teilfunktionen zerlegt und unter Verwendung der bereits genannten Programmierkonstrukte zu einem geeigneten Programmablauf angeordnet werden. Dabei spielen die zwischen den Funktionen jeweils ausgetauschten Daten als reine „Kommunikationseinheiten“ nur eine untergeordnete Rolle [WIRTH 1971:226].

Die ausschließliche Strukturierung von Anwendungssystemen auf Basis der Funktionsstruktur führt in Bezug auf die Modularitätskriterien zu einigen Schwächen. So ist es aufgrund der zahlreichen Wechselwirkungen zwischen Funktionen, der daraus resultierenden komplexen Aufrufstruktur sowie der häufig auftretenden Seiteneffekte praktisch kaum möglich, solche Anwendungssysteme, die lediglich auf der Basis einer funktionalen Dekomposition gegliedert sind, bei Wartungsmaßnahmen in der gewünschten effizienten Weise an Veränderungen anzupassen. Hierfür sind üblicherweise zahlreiche Funktionen zu analysieren und in ihrer Implementierung zu verändern, weshalb einzelne Funktionen weder als eigenständige Abstraktionseinheiten noch als selbstständige Analyseeinheiten angesehen werden können [MEYER 1997:108f.]. Darüber hinaus führt gerade das bei der Strukturierung von Anwendungssystemen eingesetzte Konzept der schrittweisen Verfeinerung

der Funktionsstruktur dazu, dass die einzelnen Funktionen stark an ihren ursprünglichen Problemkontext, nämlich den Ablauf des jeweils zu strukturierenden Anwendungssystems, angepasst sind. Neben den zahlreichen Abhängigkeiten und Wechselwirkungen verhindert insbesondere diese hohe Spezialisierung, dass Funktionen als Kompositionseinheiten effizient in nachfolgenden Entwicklungsprojekten eingesetzt werden können [MEYER 1997:113]. Die Zerlegung der von einem Anwendungssystem zu erfüllenden Aufgaben in einzelne Funktionen dient daher lediglich der Komplexitätsreduktion und der Abgrenzung von Entwurfseinheiten für die Implementierung. Als ein geeignetes Konzept für die Strukturierung von Anwendungssystemen in Komponenten eignet es sich wegen der identifizierten Schwächen indes nicht.[2]

Um diesen Schwächen zu begegnen, forderte Parnas eine Abkehr von der ablauforientierten Strukturierung von Anwendungssystemen und schlug statt dessen eine primär an ihren jeweiligen Datenstrukturen ausgerichtete Zerlegung in einzelne *Module* vor [PARNAS 1972:1056, 1058]. Ein Modul gruppiert diejenigen Funktionen zu einer Einheit, die jeweils auf einer gemeinsamen Datenstruktur arbeiten, und stellt diese über eine *Modulschnittstelle* für den einheitlichen Aufruf zur Verfügung (*Datenkapselung*). Die Implementierung der einzelnen Funktionen sowie die verwendeten Datenstrukturen bleiben dabei verborgen, d.h. sie sind von außerhalb des Moduls nicht einzusehen (*Geheimnisprinzip*). Mit der Forderung nach einem modularen Strukturierungskonzept, das auf der Datenstruktur des jeweiligen Anwendungssystems aufbaut, begann schließlich eine Entwicklung, in deren Rahmen die Bedeutung der Daten für die Strukturierung von Anwendungssystemen im Vergleich zu den Funktionen sukzessive aufgewertet wurde.

Im Rahmen des strukturierten Paradigmas führte diese Entwicklung zu einer stärker modulorientierten Ausrichtung, mit der allerdings häufig nur eine geeignete Zusammenfassung derjenigen Funktionen zu Modulen angestrebt wurde, die zunächst durch eine rein ablauforientierte, funktionale Dekomposition identifiziert wurden. Entsprechende Konzepte zur Verbindung von funktionaler Dekomposition und modularer Organisation sind bspw. Bestandteil der *Structured Design* Methode [YOURDON und CONSTANTINE 1979], bei der einzelne Funktionen im Anschluss an die Dekomposition nach ihrer *Zusammengehörigkeit* (Kohäsion) mit dem Ziel gruppiert werden, die *Abhängigkeiten* zwischen den so entstehenden Modulen möglichst zu minimieren [MEYER 1997:64]. Zu einer Ablösung der für das strukturierte Paradigma charakteristischen funktionalen Dekomposition durch eine

[2] Diese Auffassung liegt dem Ansatz von McIlroy zugrunde, der die Verwendung von Funktionen als geeignete Komponenten für die Anwendungsentwicklung vorschlägt [MCILROY 1968].

primär an der Datenstruktur ausgerichtete Strukturierung kam es demgegenüber nur zögerlich. So wurde den Daten erst im Rahmen der später entwickelten *Structured Analysis* Methoden [DEMARCO 1978; MCMENAMIN und PALMER 1984; YOURDON 1989] eine größere Bedeutung für die Strukturierung von Anwendungssystemen beigemessen. Bei dieser Methode wird die Funktionsstruktur nicht mehr durch eine ausschließlich am Ablauf orientierte funktionale Dekomposition, sondern vielmehr aus einem Modell der zugrunde liegenden *Datenflüsse* abgeleitet [DEMARCO 1978:297-301]. Erst mit den Methoden des *datenstrukturorientierten Entwurfs* [ORR 1977; JACKSON 1983] wurde für die Strukturierung von Anwendungssystemen in Module ausschließlich deren jeweilige *Datenstruktur* genutzt. Bei diesen Ansätzen wird die Ermittlung der Funktionsstruktur lediglich zur *Zuordnung* einzelner Funktionen zu den jeweils identifizierten Modulen durchgeführt.

Als Strukturierungseinheiten bieten Module gegenüber einzelnen Funktionen insbesondere in Bezug auf die Erfüllung der Modularitätskriterien eine Reihe von Vorteilen. So erlaubt das Modulkonzept mit dem zentralen Grundsatz der Datenkapselung und dem Geheimnisprinzip insbesondere Änderungen an der Implementierung einzelner Funktionen oder den zugrunde liegenden Datenstrukturen, ohne dass hiervon weitere Module betroffen werden [PARNAS 1972:1054]. Da einzelne Module wegen des Geheimnisprinzips keine Abhängigkeiten auf die Implementierungen, sondern lediglich auf die Schnittstellen anderer Module aufweisen, lassen sie sich als Abstraktionseinheiten weitgehend unabhängig voneinander austauschen, solange die Schnittstellen erhalten bleiben. Daneben besitzen Module typischerweise weniger Abhängigkeiten untereinander als etwa einzelne Funktionen, da die Grundlage für deren Entstehung gerade das Prinzip der Gruppierung zusammengehöriger Funktionen ist. Aus diesem Grunde sind sie ebenfalls besser als Entwurfseinheiten für eine autonome Weiterentwicklung geeignet [PARNAS 1972:1054]. Schließlich führt der mit dem Modulkonzept eingeführte Schnittstellenbegriff im Prinzip auch zu einer verbesserten eigenständigen Analysierbarkeit von Strukturierungseinheiten [PARNAS 1972:1054]. Dabei ist jedoch zu beachten, dass für eine effiziente Analyse neben den von einem Modul jeweils angebotenen Funktionen auch seine Abhängigkeiten (also Aufrufe modulfremder Funktionen) zu beschreiben sind [DEREMER und KRON 1976:118].

Dennoch besitzt auch das Modulkonzept einige Schwächen, die sich bspw. bei der Anpassung von Anwendungssystemen im Rahmen von Wartungsarbeiten bemerkbar machen. So wird die Modulstruktur üblicherweise zugunsten einer höheren Performanz beim Kompilieren eines Anwendungssystems aufgegeben und dabei eine Eliminierung der jeweiligen Modulschnittstellen angestrebt [PARNAS 1972:1057]. Die somit nur als ein logisches Konzept der Entwicklung genutzte Modulstruktur führt dazu, dass Anwendungssysteme in der Regel in physisch monolithischer Form ausgeliefert werden und deshalb im Falle einer

Wartungsmaßnahme jeweils das komplette Anwendungssystem (also nicht etwa nur die betroffenen Module) erneut zu installieren sein wird [HERZUM und SIMS 2000:22].

Darüber hinaus hat sich vor allem die Wiederverwendung einzelner Module als Kompositionseinheiten im Rahmen nachfolgender Entwicklungsprojekte als schwierig herausgestellt. Dies ist zum einen dadurch bedingt, dass sich durch den Import von Modulschnittstellen für den Aufruf von Methoden anderer Module direkte, starre Abhängigkeiten auf die Schnittstellen einzelner Module ergeben [SZYPERSKI, et al. 2002:40]. So ist es zwar möglich, ein Modul durch ein anderes zu ersetzen, dessen Implementierung (Funktionalität) abweicht – jedoch muss dieses dabei stets die gleiche Schnittstelle besitzen. Dies bedeutet insbesondere, dass die Zahl der bereitgestellten Funktionen sowie die Datenstrukturen ihrer Parameter und Rückgabewerte identisch sein müssen. Wegen dieser Einschränkungen ist eine Wiederverwendung von Modulen in solchen Kombinationen mit anderen, die während der Entwicklung nicht vorhergesehen wurden, in der Praxis nur in einem eingeschränkten Umfang zu realisieren. Weitere Schwierigkeiten entstehen dadurch, dass Module häufig im Anschluss an einen funktionalen Dekompositionsprozess entstanden und somit in ihrer Struktur stark an den jeweiligen Problemkontext angepasst sind. Insofern ist das ihrer Entstehung zugrunde liegende strukturierte Entwicklungsparadigma für die ungenügende Wiederverwendbarkeit von Modulen maßgeblich (mit-) verantwortlich [MEYER 1997:113].

2.1.3 Objektorientierte Entwicklung

Die mangelnde Wiederverwendbarkeit der Entwicklungsergebnisse sowie die anhaltenden Unsicherheiten, die sich durch die getrennte Betrachtung von Funktionen und Daten sowie ihre jeweils wechselnde Rolle für die Strukturierung von Anwendungssystemen ergaben, begünstigten schließlich die Ablösung des strukturierten Paradigmas durch das objektorientierte Paradigma, mit dessen Konzept für die Strukturierung von Anwendungssystemen in Klassen ein Ansatz zur Lösung der verbleibenden Probleme gefunden zu sein schien [MEYER 1997:114].

Ebenso wie das strukturierte Entwicklungsparadigma ist auch die Objektorientierung als ein Konzept der Implementierung entstanden, das sich in der Folge zu einem umfassenden Paradigma für die Strukturierung und Entwicklung von Anwendungssystemen fortentwickelt hat. Grundlegend für die Entstehung des objektorientierten Entwicklungsparadigmas waren vor allem die im Jahre 1966 vorgestellte Programmiersprache Simula mit ihrem neuartigen, auf *Klassen* basierenden Konzept für die Strukturierung von Programmen

[DAHL und NYGAARD 1966] sowie die Einführung der *abstrakten Datentypen* [LISKOV und ZILLES 1974], die dieses Konzept theoretisch untermauern.

Im objektorientierten Paradigma wird ein Anwendungssystem als System kollaborierender *Objekte* aufgefasst, die zur Laufzeit mit Hilfe von Verweisen zueinander in Beziehung stehen und durch den Austausch von Nachrichten miteinander kommunizieren. Jedes Objekt hat eine eindeutige Identität und trägt bestimmte Merkmale in Form von Attributen und Fähigkeiten, wobei die Attribute den jeweiligen Zustand und die Fähigkeiten das Verhalten des Objekts beschreiben. Dabei sind sowohl die Implementierungen der einzelnen Fähigkeiten als auch der Zustand eines Objektes in der Regel gekapselt und dementsprechend von außen nicht direkt einzusehen bzw. zu verändern. Die Manipulation des internen Zustands eines Objekts erfolgt vielmehr ausschließlich durch die Inanspruchnahme seiner jeweiligen Fähigkeiten, die über eine *Objektschnittstelle* verfügbar sind und durch den Versand entsprechender Nachrichten von anderen Objekten aktiviert werden können. Somit übernimmt das objektorientierte Paradigma mit dem Grundsatz der *Datenkapselung* und dem *Geheimnisprinzip* wichtige Ansätze, die sich bereits bei der strukturierten Entwicklung bewährt haben, und entwickelt diese zu zentralen Konzepten weiter [NIERSTRASZ 1989].

Objekte mit gemeinsamen Merkmalen, d.h. mit gleichen Attributen und Fähigkeiten, lassen sich gruppieren und bezüglich der gemeinsamen Struktur durch *Klassen* beschreiben, die als Schemata für ihre Erzeugung zur Laufzeit dienen. Somit lassen sich Objekte grundsätzlich als *Instanzen* einer Klasse auffassen [MEYER 1997:165f.]. Einzelne Klassen bilden als statische Abstraktionen von Objekten im objektorientierten Entwicklungsparadigma die zentralen (und zugleich einzigen) Einheiten für die Strukturierung von Anwendungssystemen [MEYER 1997:24, 209; BROY und SIEDERSLEBEN 2002:5]. Jede Klasse basiert dabei auf einem *abstrakten Datentyp*, der jeweils eine *Datenstruktur* beschreibt und diese Datenstruktur mit den auf ihr ausführbaren *Funktionen* zu einer Einheit zusammenfasst [LISKOV und ZILLES 1974; MEYER 1997:129f.]. Somit ist die Strukturierung von Anwendungssystemen beim objektorientierten Ansatz im Gegensatz zum strukturierten Entwicklungsparadigma konsequent an den jeweils zu manipulierenden Datenstrukturen ausgerichtet [MEYER 1997:116, 684]. Darüber hinaus erfolgt die Strukturierung üblicherweise nicht im Rahmen eines Prozesses der schrittweisen Verfeinerung, sondern vielmehr ausgehend von einer Analyse der beteiligten Klassen, die anschließend in einem *Kompositionsprozess* zu einem geeigneten Gesamtsystem verbunden werden [MEYER 1997:114, 147].

Die während der Strukturierung eines Anwendungssystems identifizierten Klassen sind typischerweise in Spezialisierungshierarchien angeordnet, in deren Rahmen eine *Verer-*

bung von Eigenschaften der übergeordneten auf die jeweils untergeordnete(n) Klasse(n) stattfindet. Dabei ist grundsätzlich zwischen drei Zielen, die mit der Vererbung von Eigenschaften angestrebt werden können, zu unterscheiden [MEYER 1997:459, 494-497; SZYPERSKI, et al. 2002:110]. Mit der *Implementierungsvererbung* (Implementation Inheritance bzw. Subclassing) wird eine Übertragung von Merkmalen und ihrer Implementierung zwischen verschiedenen Klassen angestrebt. Dabei erwirbt die untergeordnete Klasse die Attribute und Fähigkeiten der jeweils übergeordneten Klasse. Demgegenüber zielt die *Schnittstellenvererbung* (Interface Inheritance bzw. Subtyping) lediglich auf eine Übertragung der Schnittstellenvereinbarungen, also vor allem der Signaturen der jeweiligen Fähigkeiten. Mit beiden Formen der Vererbung wird zudem üblicherweise eine *Konformitätsbeziehung* [MEYER 1997:474] zwischen den beteiligten Klassen festgelegt, die eine einseitige *Substituierbarkeit* von Instanzen der übergeordneten durch solche der jeweils untergeordneten, *konformen* Klasse ermöglicht.[3] Weitere mit der Festlegung einer Vererbungsbeziehung verfolgte Ziele finden sich bei [MEYER 1997:822-833].

Während die Implementierungs- und die Schnittstellenvererbung vor allem der Wiederverwendung von Teilen der Implementierung dient, ist die Festlegung von Konformitätsbeziehungen zwischen Objekten zusammen mit dem Konzept der *dynamischen Bindung* von Objekten eine wesentliche Grundlage für die Einführung einer kontrollierten *Polymorphie* (zu dt. *Vielgestaltigkeit*) von Objekten [CARDELLI und WEGNER 1985; MEYER 1997:467-480; SZYPERSKI, et al. 2002:83-108], mit dem das objektorientierte Paradigma eine im Vergleich zu den klassischen modulorientierten Ansätzen erhöhte Flexibilität bei der Kopplung von Strukturierungseinheiten erreicht. So erlaubt die *Inklusions- bzw. Subtyppolymorphie* den Objekten einer bestimmten Klasse, auch als Instanzen einer jeden Klasse aufzutreten, zu denen ihre Klasse in einer Konformitätsbeziehung steht. Damit können Objekte zur Laufzeit nicht nur mit Objekten einer bestimmten, durch die statische Klassenstruktur vorgegebenen Klasse interagieren, sondern auch mit jedem beliebigen Objekt, dessen Klasse in einer Konformitätsbeziehung zu der eigentlich vorgegebenen Klasse steht und diese somit spezialisiert. Zusätzlich ermöglicht die *parametrische Polymorphie* die Implementierung generischer Klassen, die eine bestimmte Funktionalität für eine Reihe von Datenstrukturen zur Verfügung stellen und durch diese parametrisiert werden. Weitere, für den Kontext dieser Arbeit jedoch weniger zentrale Konzepte für die flexible Kopplung von Objekten basieren insbesondere auf dem *Überladen von Funktionen* sowie der *gebundenen Polymorphie* [CARDELLI und WEGNER 1985; SZYPERSKI, et al. 2002:107f.].

[3] Eine prominente Ausnahme stellt die objektorientierte Programmiersprache Smalltalk [GOLDBERG und ROBSON 1983] dar, bei der die Vererbung nicht per se der Festlegung einer Konformitätsbeziehung dient.

Bezug nehmend auf den klassischen, auf Modulen basierenden Ansatz wurde das mit der Objektorientierung eingeführte Klassenkonzept häufig als das überlegene Paradigma für die Strukturierung von Anwendungssystemen bezeichnet [COX 1987; MEYER 1997:210]. Hierfür spricht zunächst die konsequent an den Daten ausgerichtete Identifikation von Klassen, die mit der von Parnas geforderten Vorgehensweise übereinstimmt [PARNAS 1972:1056] und sich mit der Theorie der abstrakten Datentypen dabei zudem auf ein theoretisch fundiertes Konzept stützt [MEYER 1997:144f.; SZYPERSKI, et al. 2002:39]. Darüber hinaus werden auch die mit dem Polymorphismus erreichte höhere Flexibilität bei der Kopplung von Strukturierungseinheiten sowie die sich vor allem aufgrund der Implementierungsvererbung ergebenden Möglichkeiten zur effizienten Wiederverwendung von Implementierungsteilen als Argumente ins Feld geführt [MEYER 1997:494-497].

Dennoch können auch mit dem objektorientierten Paradigma nicht alle aufgestellten Modularitätskriterien erfüllt werden, weshalb ein Vergleich mit dem Modulkonzept eher unterschiedliche Stärken und Schwächen als eine prinzipiell vorhandene Überlegenheit ergibt. So operationalisiert die Theorie der abstrakten Datentypen zwar das Identifizieren von Anwendungssystemteilen, die sich unabhängig voneinander als Entwurfseinheiten weiterentwickeln lassen, führt dabei jedoch im Allgemeinen zu Strukturen mit einer sehr feinen Granularität. Da Klassen (im Gegensatz zu Modulen) stets nur einen abstrakten Datentyp implementieren [MEYER 1997:142] und zudem das einzige Strukturierungskonzept der Objektorientierung darstellen, bedeutet dies, dass bei der Zerlegung eines Anwendungssystems generell eine Vielzahl von Teilen identifiziert wird. Der Entwicklungsprozess, dessen Aufgabe die geeignete Verbindung der einzelnen Teile zu einem Gesamtsystem ist [MEYER 1997:147], besitzt infolgedessen eine hohe Komplexität [BROY und SIEDERSLEBEN 2002:7]. Darüber hinaus weisen Klassen aufgrund ihrer feinen Struktur häufig eine Vielzahl von Abhängigkeiten untereinander auf, weshalb ihre unabhängige Austauschbarkeit in der Praxis eher die Ausnahme darstellt [MILI, et al. 1995:547].

In diesem Zusammenhang wirkt sich gerade auch das Konzept der Implementierungsvererbung als Nachteil aus. So ist im Falle einer Wartung bei der Anpassung von Klassen, die durch eine Implementierungsvererbung mit anderen verbunden sind, davon auszugehen, dass neben der eigentlich von der Wartung betroffenen Klasse auch die zu dieser in Beziehung stehenden Klassen zu überarbeiten sind. Dieses sog. *Problem der zerbrechlichen Basisklasse* (Fragile Base Class Problem [SZYPERSKI, et al. 2002:115-122]) entsteht durch die Tatsache, dass durch eine Implementierungsvererbung direkte Abhängigkeiten auf Teile der Implementierung einer anderen Klasse entstehen, die das Geheimnisprinzip durchbrechen [SNYDER 1986]. Ein weiteres Problem entsteht beim Austausch einzelner Klassen aufgrund der Tatsache, dass auch das objektorientierte Strukturierungskonzept häufig nur

ein logisches Konzept der Entwicklung ist. Dementsprechend besitzt das fertige Anwendungssystem in der Regel eine monolithische Struktur und ist im Wartungsfalle somit als Ganzes neu auszuliefern [HERZUM und SIMS 2000:22].

Schließlich sind Klassen schon aufgrund ihrer feinen Granularität und der daraus resultierenden komplexen Wechselwirkungen sowie den bestehenden Vererbungsbeziehungen nur eingeschränkt als eigenständige Analyseeinheiten nutzbar. Damit bleiben in Bezug auf die Modularitätskriterien hauptsächlich die flexiblen Kopplungsmöglichkeiten, die sich durch das Konzept der Polymorphie ergeben, als wesentliche Vorteile der Objektorientierung zu nennen. Dieses erlaubt eine weit reichende Abkehr vom Grundsatz der Schnittstellengleichheit, so dass spezialisierte Objekte über zusätzliche Fähigkeiten (Funktionen) verfügen und im Rahmen der Vererbung übertragene Fähigkeiten redefinieren können. Bei der Redefinition können neben der Implementierung auch die Typen von Parametern und Rückgabewerten gemäß den Regeln der *Kontravarianz* bzw. *Kovarianz* verändert werden (für eine ausführliche Diskussion vgl. [MEYER 1997:595-604; SZYPERSKI, et al. 2002:90-92]). Dank dieser Vorteile sind Klassen im Vergleich zu Modulen vor allem als die überlegenen Kompositionseinheiten einzustufen. Grenzen werden der Redefinition von Fähigkeiten lediglich durch das Prinzip der *verhaltensmäßigen Konformität* [LISKOV und WING 1994] gesetzt, das die Beibehaltung einer semantisch kompatiblen Funktionalität fordert, sich jedoch im Allgemeinen nicht technisch überprüfen lässt. Einschränkend ist zudem anzumerken, dass eine Änderung der Signatur von Objektfähigkeiten von vielen Programmiersprachen noch immer nicht unterstützt wird [SZYPERSKI, et al. 2002:93].

2.1.4 Komponentenorientierte Entwicklung

Wegen der Schwächen im Hinblick auf die Erfüllung der übrigen Modularitätskriterien eignet sich auch das objektorientierte Paradigma nicht als ein ideales Konzept für die Strukturierung von Anwendungssystemen in Komponenten.[4] Erst mit der Einführung der komponentenorientierten Anwendungsentwicklung scheint heute ein Paradigma gefunden zu sein, das die aufgestellten Modularitätskriterien umfassend zu erfüllen vermag. Das komponentenorientierte Entwicklungsparadigma entwickelte sich im Gegensatz zu den vorgenannten Entwicklungsparadigmen nicht aus einem Konzept der Implementierung, sondern aus einem technischen Konzept für die Organisation von Anwendungssystemen

[4] Diese Auffassung liegt z.B. den Arbeiten von Cox und Meyer zugrunde, die die Verwendung von Klassen als geeignete Komponenten für die Anwendungsentwicklung vorschlagen [COX 1987; MEYER 1997:210].

aus einzelnen Teilen, die ggf. auf verschiedene Rechner verteilt sind. Für seine Entstehung war neben den genannten Schwierigkeiten bei der Entwicklung und Wartung komplexer objektorientierter Anwendungssysteme vor allem die wachsende Bedeutung der verteilten Anwendungssysteme mit ihrer jeweils drei- oder mehrschichtigen Client-Server-Architektur [ORFALI, et al. 1999] grundlegend.

Die Entwicklung verteilter Anwendungssysteme wurde zunächst im Rahmen des objektorientierten Paradigmas mit den sog. *verteilten Objektsystemen* [GRIFFEL 1998:36; HERZUM und SIMS 2000:12-15] unterstützt, die auf speziellen Objektmodellen wie der Common Object Request Broker Architecture (OMG CORBA) [OMG 1997] oder dem Distributed Common Object Model (Microsoft DCOM) [SZYPERSKI, et al. 2002:330-345] basieren. Diese erweitern die klassische Objektorientierung um zusätzliche Konzepte für die verteilte Kommunikation von Objekten und ermöglichen es, auch solche Objekte zu verteilten Objektsystemen zu verbinden, die auf verschiedenen Rechnern beheimatet sind. Verteilte Anwendungssysteme bestehen somit aus einer Vielzahl fein strukturierter, verteilter und miteinander interagierender Objekte [HERZUM und SIMS 2000:14].

Die für die Objektorientierung typische, feine und durch eine Vielzahl von Wechselwirkungen geprägte Objektstruktur wirkt sich wegen des hieraus resultierenden Kommunikationsbedarfs jedoch insbesondere bei verteilten Anwendungssystemen negativ auf das Laufzeitverhalten aus. Mit der wachsenden Bedeutung dieses Anwendungssystemtyps verstärkten sich daher die Bemühungen, nach Strukturierungseinheiten zu suchen, die sich durch eine größere Abgeschlossenheit und einen geringeren Kommunikationsbedarf auszeichnen [GRIFFEL 1998:36]. Dabei entwickelten sich die zuvor erwähnten Objektmodelle zu Komponentenmodellen wie dem CORBA Component Model (OMG CCM) [WANG, et al. 2001; OMG 2002; SZYPERSKI, et al. 2002:247-260] oder dem (Distributed) Component Object Model weiter [EWALD 2001; SZYPERSKI, et al. 2002:329-357], und bilden zusammen mit neu entstandenen Komponentenmodellen wie den Enterprise Java Beans (Sun EJB [BLEVINS 2001; SZYPERSKI, et al. 2002:284-328]), .NET (Microsoft .NET [SZYPERSKI, et al. 2002:357-380]) bzw. XML Web-Services [SZYPERSKI, et al. 2002:223-230] eine wichtige Grundlage für die Komponentenorientierung in ihrer heutigen Form.

Vor allem während der Entstehungszeit war die Komponentenorientierung von einem uneinheitlichen Sprachgebrauch gekennzeichnet. Dementsprechend wurden viele wesentliche Begriffe mit verschiedenen Bedeutungen verwendet und zum Teil nur schlagwortartig definiert. Daneben entstanden vor allem Meinungsunterschiede hinsichtlich der notwendigen und hinreichenden Eigenschaften von Komponenten. Nachfolgend werden daher die wesentlichen Konzepte des komponentenorientierten Entwicklungsparadigmas besprochen,

die in der Literatur heute weitgehend anerkannt werden. Eine Gegenüberstellung heutiger Komponentendefinitionen, die den Komponentenbegriff aus verschiedenen Perspektiven betrachten, findet sich in Abb. 2.4. Im Folgenden werden diese Perspektiven zu einem einheitlichen Gesamtbild integriert.

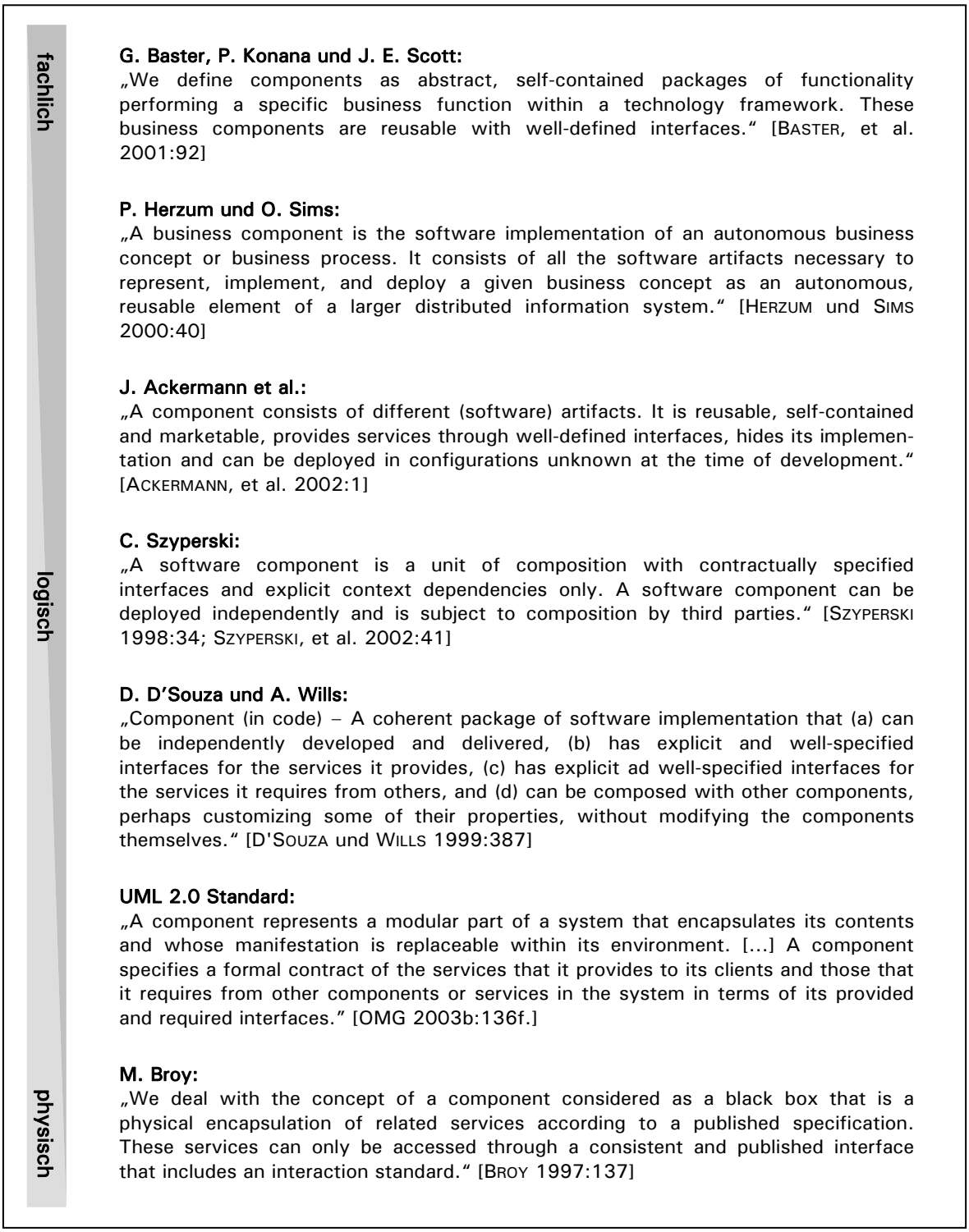


Abb. 2.4: Gegenüberstellung von Komponentendefinitionen aus fachlicher, logischer und physischer Sicht.

Das komponentenorientierte Entwicklungsparadigma [SZYPERSKI 1998; SZYPERSKI, et al. 2002] strebt eine Zerlegung von Anwendungssystemen in möglichst abgeschlossene Teile an und führt hierzu *Komponenten* als zentrale Einheiten für die Strukturierung ein. Jede Komponente implementiert eine Menge zusammengehöriger Daten und Funktionen und stellt letztere an ihren *Komponentenschnittstellen* als Dienste nach außen zur Verfügung. Wie Module und Klassen stellen Komponenten *Schemata* dar, die ihre Implementierung verbergen und auf dem Grundsatz der *Datenkapselung* sowie dem *Geheimnisprinzip* basieren. Im Gegensatz zu Modulen und Klassen bilden Komponenten allerdings nicht nur logische Strukturierungseinheiten, sondern stellen gleichzeitig auch die *physischen Bausteine* der jeweiligen Anwendungssysteme dar [SZYPERSKI, et al. 2002:195]. Dementsprechend werden die Schnittstellen zwischen einzelnen Komponenten beim Kompilieren beibehalten und sind damit auch zur Installations- bzw. Laufzeit (zwischen den verschiedenen Komponenteninstanzen) vorhanden.

Darüber hinaus können Komponenten in Abhängigkeit von dem technischen Komponentenmodell im Allgemeinen mehrere Schnittstellen besitzen. Mit diesem Konzept der *Mehrfachschnittstellen* [GRIFFEL 1998:62-64; SZYPERSKI, et al. 2002:42] lassen sich aus technischer Sicht vor allem die Abhängigkeiten zwischen Komponenten, die sich durch den Import von Komponentenschnittstellen für den Aufruf von Funktionen anderer Komponenten ergeben, auf Teile der Außensicht reduzieren [D'SOUZA und WILLS 1999:397; CHEESMAN und DANIELS 2001:3]. Daneben erleichtert dieses Konzept vor allem die unabhängige Erweiterbarkeit der Komponentenfunktionalität, etwa bei der Entwicklung neuer Versionen [SZYPERSKI, et al. 2002:52f.]. Aus fachlicher Sicht unterstützt es schließlich die getrennte Modellierung verschiedener Rollen, die Komponenten im Kontext eines Anwendungssystems einnehmen können [BROWN 2000:106-109; SZYPERSKI, et al. 2002:42].

Die Strukturierung von Anwendungssystemen in Komponenten erfolgt gemäß dem Grundsatz, die Abhängigkeiten zwischen den einzelnen Komponenten zu minimieren und zusammengehörige Funktionen und Daten möglichst zu jeweils einer Komponente zu gruppieren (Prinzip der *minimalen Abhängigkeit* und der *maximalen Kohäsion* [SZYPERSKI, et al. 2002:40]).[5] Dabei dient die Minimierung von Abhängigkeiten primär dem Ziel, möglichst wieder verwendbare Komponenten zu identifizieren, die sich mit einer überschaubar

[5] Eine solche Gruppierung lässt sich durch die Analyse der Daten- und Funktionsstruktur der jeweiligen Anwendung erreichen. So lassen sich durch die Gegenüberstellung der Daten- und Funktionsstruktur zusammengehörige Anwendungsteile identifizieren [ALBANI, et al. 2003]. Weitere Kriterien zur Identifikation von Komponenten finden sich bei [HERZUM und SIMS 2000:444-448; SZYPERSKI, et al. 2002:139-150].

einfachen Software-Architektur zu einem Anwendungssystem verbinden lassen [MEYER 1997:41]. Die Bündelung zusammengehöriger Daten und Funktionen unterstützt darüber hinaus die Abgrenzung von Komponenten, die auch über eine aus konzeptioneller (fachlicher) Sicht klare, abgeschlossene Funktionalität verfügen [SINZ 1999; HERZUM und SIMS 2000:40; BASTER, et al. 2001:92].

In Abhängigkeit von der jeweils zusammengefassten Funktionalität entstehen während des Strukturierungsprozesses üblicherweise Komponenten mit einer *unterschiedlichen Granularität* [FERSTL, et al. 1997:42]. In seiner Flexibilität bei der Zusammenfassung von Daten und Funktionen ähnelt der komponentenorientierte Strukturierungsprozess damit der Vorgehensweise zur Zerlegung eines Anwendungssystems in Module [SZYPERSKI, et al. 2002:39]. Jedoch besitzen Komponenten eine vergleichsweise komplexe innere Struktur, die typischerweise mehrere Klassen, Module oder sogar weitere Komponenten umfasst [SZYPERSKI, et al. 2002:420]. Eine Komponente stellt als Strukturierungseinheit somit eine inhaltlich flexible Abstraktion über Klassen, Module oder Systeme dar [FRANK 1999:12], die in einem nachfolgenden Entwicklungsschritt weiter zu detaillieren ist. Dementsprechend erfolgt die Strukturierung eines Anwendungssystems bei der komponentenorientierten Entwicklung grundsätzlich in zwei Schritten, wobei zunächst die Zerlegung in Komponenten und die Festlegung der Software-Architektur, die sog. *Entwicklung im Großen*, im Mittelpunkt steht. In einem zweiten Schritt erfolgt, sofern nicht bereits existierende Komponenten in die Entwicklung einbezogen werden können, die *Entwicklung im Kleinen*, die die Zerlegung der zu realisierenden Komponenten in geeignete Implementierungseinheiten sowie die Festlegung der jeweiligen Komponentenarchitektur umfasst [GRIFFEL 1998:46].

Somit nutzt die Komponentenorientierung das von DeRemer und Kron für die Entwicklung großer Anwendungssysteme vorgeschlagene Prinzip der Trennung von Programmierung im Großen und Programmierung im Kleinen [DEREMER und KRON 1976] und entwickelt es zu einem Konzept der *Systemorganisation* weiter. Dabei ist die Entwicklung im Großen, die Systemerstellung, typischerweise durch den *Problemkontext* des jeweils zu entwickelnden Anwendungssystems geprägt, während die Entwicklung im Kleinen, also die Realisierung der einzelnen Komponenten, stärker durch ein *generalisierendes Vorgehen* gekennzeichnet ist [SOMMERVILLE 1992:309-320; SAMETINGER 1997:173f.]. Mit dem komponentenorientierten Entwicklungsparadigma werden jedoch im Wesentlichen Konzepte für die Entwicklung im Großen, d.h. für die Zerlegung von Anwendungssystemen in Komponenten und deren spätere Komposition bereitgestellt. Dementsprechend löst es das objektorientierte Paradigma auch vor allem im Hinblick auf seine Bedeutung für die Strukturierung von Anwendungssystemen ab. Die Entwicklung im Kleinen erfolgt dagegen üblicherweise unter Verwendung des objektorientierten bzw. strukturierten Entwicklungspa-

radigmas. Bezug nehmend auf diesen Aspekt wird die Komponentenorientierung daher bisweilen auch als Ergänzung bzw. Skalierung vorangegangener Entwicklungsparadigmen wie der Objektorientierung bezeichnet [GRIFFEL 1998:32; BROWN 2000:78].

Diese Sichtweise wird durch die Tatsache gestützt, dass die Komponentenorientierung bewährte Konzepte der Objektorientierung wie die *dynamische Bindung* und die *Inklusionspolymorphie* übernimmt, um während der Entwicklung im Großen eine flexible Verbindung von Komponenten zu Gesamtsystemen zu gewährleisten [SZYPERSKI, et al. 2002:467]. Im Gegensatz zur Objektorientierung basiert die Inklusionspolymorphie im komponentenorientierten Paradigma jedoch ausschließlich auf der Möglichkeit, zwischen Komponenten eine *Konformitätsbeziehung* im Rahmen einer *Schnittstellenvererbung* zu deklarieren, die die einseitige Ersetzbarkeit von Komponenten durch jeweils spezialisierte (konforme) Komponenten ermöglicht [GRIFFEL 1998:67f.]. Hingegen wird auf eine Unterstützung der *Implementierungsvererbung* zwischen Komponenten wegen der hierdurch verursachten direkten Abhängigkeiten auf die Implementierungen anderer Komponenten in der Regel verzichtet [SZYPERSKI, et al. 2002:460f.]. Gemäß dem Konzept der Mehrfachschnittstellen reicht darüber hinaus für die Ersetzbarkeit im Allgemeinen bereits aus, dass die ersetzende Komponente zu der ersetzten Komponente bezüglich einer Teilmenge von Schnittstellen konform ist. Sie braucht somit nicht in ihrer gesamten Außensicht mit dieser überein zu stimmen.

Aufgrund der erwähnten Eigenschaften besitzen Komponenten als Strukturierungseinheiten im Hinblick auf die Erfüllung der Modularitätskriterien zahlreiche Vorteile. So unterstützt die flexible Gruppierung von zusammengehörigen Funktionen und Daten zu Komponenten zunächst die Abgrenzung von Teilen, die auch aus konzeptioneller (fachlicher) Sicht abgeschlossen sind und sich deshalb in einem besonderen Maße für die unabhängige Weiterentwicklung als Entwurfseinheiten eignen [GRIFFEL 1998:62-63]. Darüber hinaus erleichtert die Strukturierung eines Anwendungssystems in Komponenten, die über eine jeweils eindeutige Funktionalität verfügen, die Ermittlung der von einer Änderung im Anwendungsbereich jeweils betroffenen Teile. Zusammen mit dem konsequent eingehaltenen Prinzip der Datenkapselung und dem Geheimnisprinzip trägt somit gerade auch der Grundsatz der konzeptionell-fachlichen Abgeschlossenheit zur effizienten Austauschbarkeit von Komponenten bei.

Da Komponenten darüber hinaus zugleich physisch abgeschlossene Bausteine für die zu entwickelnden Anwendungssysteme bilden, lassen sie sich auch nach der erfolgten Auslieferung und Installation noch mit einem vergleichsweise geringen Aufwand identifizieren und einzeln auswechseln [CRNKOVIC 2002:130; SZYPERSKI, et al. 2002:41]. Somit stellen

Komponenten insbesondere im Vergleich zu Modulen oder Klassen die überlegenen Abstraktionseinheiten dar. Allerdings ist anzumerken, dass sich durch die Beibehaltung der Komponentenschnittstellen, die zur Gewährleistung einer unabhängigen Austauschbarkeit nach der Installation notwendig ist, im Allgemeinen die Performanz der entwickelten Anwendungssysteme zur Laufzeit verschlechtert. Diese Verschlechterung entsteht aufgrund der zusätzlichen Indirektionen bei der komponentenübergreifenden Kommunikation.

Als Kompositionseinheiten besitzen Komponenten wegen der aus der Objektorientierung übernommenen Konzepte zur flexiblen Kopplung von Teilen grundsätzlich alle Vorteile, die bereits das Klassenkonzept ausgezeichnet haben. Zwar heben sich Komponenten von diesen insbesondere durch eine höhere Granularität ab und besitzen somit im Allgemeinen eine komplexere Außensicht, die aus einer größeren Zahl von Funktionen besteht. Da diese jedoch entsprechend dem Prinzip der Mehrfachschnittstellen in der Regel auf mehrere Komponentenschnittstellen verteilt sind, ist es trotz der höheren Granularität von Komponenten möglich, die Abhängigkeiten zwischen diesen wirksam zu begrenzen. Darüber hinaus unterstützt dieses Konzept auch die unabhängige Erweiterung von Komponenten um neue Funktionalität. So kann bspw. im Rahmen eines Versionswechsels (während einer Übergangsphase) neben der alten auch eine neue Schnittstelle angeboten werden [GRIFFEL 1998:62; SZYPERSKI, et al. 2002:52f.]. Schließlich trägt die bei der Entwicklung von Komponenten anwendbare generalisierende Vorgehensweise dazu bei, dass diese von einem konkreten Problemkontext weitgehend unabhängig gemacht werden können und sich somit leichter in anderen Entwicklungsprojekten wieder verwenden lassen [GRIFFEL 1998:48]. Die Entwicklung im Kleinen ist bei Anwendung einer solchen Vorgehensweise weitgehend durch das Ziel der systematischen Wiederverwendung geprägt (*Development for Reuse*) [SAMETINGER 1997:158, 172].

Wegen ihrer durchgängigen Abgeschlossenheit lassen sich Komponenten prinzipiell weitgehend unabhängig voneinander als Analyseeinheiten verwenden und auf ihre Eignung für eine bestimmte Anwendung untersuchen. Dabei wirkt es sich zunächst vorteilhaft aus, dass Komponenten üblicherweise auch eine aus fachlich-konzeptioneller Sicht abgeschlossene Funktionalität implementieren, während Module und Klassen aufgrund der Datenkapselung und des Geheimnisprinzips im Allgemeinen nur aus logischer Sicht als eigenständige Einheiten anzusehen sind. Aufgrund ihrer Konzeption als eigenständige Bausteine von Anwendungssystemen sind Komponenten darüber hinaus allerdings auch physisch abgeschlossen. Das bedeutet insbesondere, dass Komponenten in ausführbarer Form, d.h. als *Blackbox* bereitgestellt werden können. Damit ist es im Allgemeinen nicht möglich, im Rahmen von Untersuchungen auf die Implementierungsdetails von Komponenten zurückzugreifen oder diese durch Methoden des Re-Engineering bzw. des Testens mit vertretba-

rem Aufwand und mit der gewünschten Exaktheit zu ermitteln [GARLAN, et al. 1995; WEYUKER 2001]. Um ihrer Funktion als Analyseeinheiten dennoch gerecht zu werden, sind Komponenten deshalb mit einer *Beschreibung* auszuliefern, die neben den jeweils angebotenen Diensten auch die Abhängigkeiten zu den Diensten anderer Komponenten in angemessener, d.h. für den Entwickler geeigneter Form (als Spezifikation) darstellen sollte [D'SOUZA und WILLS 1999:387; SZYPERSKI, et al. 2002:41].

2.2 Entwicklungsmethodik

Die Entwicklung von (betrieblichen) Anwendungssystemen stellt in der Regel eine Aufgabe von hoher Komplexität dar, die ohne eine systematische Untergliederung in Teilaufgaben sowie eine Unterstützung durch spezialisierte Methoden und Werkzeuge nicht zu bewältigen ist. Wie in den anderen Ingenieurwissenschaften wird daher auch für die Anwendungsentwicklung die Schaffung einer umfassenden *Konstruktions- bzw. Entwicklungsmethodik* angestrebt [NAUR und RANDELL 1969], die allgemein ein geplantes, methodisches Vorgehen auf der Basis konkreter Handlungsanweisungen ermöglicht [PAHL und BEITZ 2003:10f.]. Als wesentlicher Bestandteil einer solchen Entwicklungsmethodik ist neben den einzelnen Methoden und Werkzeugen vor allem das *Vorgehensmodell* anzusehen [BOEHM 1988:61], das die Entwicklungsarbeit in verschiedene Phasen untergliedert und diese zusammen mit den jeweils einzusetzenden Methoden, Werkzeugen und Prinzipien einer Methodik inhaltlich zu einem allgemeingültigen, durchgängigen Prozess verbindet [BOEHM 1988:61; MCDERMID und ROOK 1991:15/19-15/20; PRESSMAN 1997:31-33]. Falls dabei auch die Phase der Systemnutzung und Wartung in den Prozess einbezogen wird, wird statt von einem Vorgehensmodell häufig auch von einem *Lebenszyklusmodell* gesprochen [SCHACH 1990:43; PRESSMAN 1997:31-33].

Idealtypisch lassen sich Vorgehensmodelle im Hinblick auf die Art, wie die einzelnen Entwicklungsphasen beschrieben und miteinander verbunden werden, zunächst als *aktivitätsorientiert* oder *ergebnisorientiert* einordnen [DOWSON 1987:37]. Aktivitätsorientierte Vorgehensmodelle beschreiben die einzelnen Entwicklungsphasen vor allem in Form von Aufgaben und durchzuführenden Arbeitsschritten, die anhand von Arbeitsplänen detailliert vorgegeben und miteinander verknüpft werden. Demgegenüber legen ergebnisorientierte Vorgehensmodelle vor allem die Eigenschaften der Ergebnisse einzelner Entwicklungsphasen fest und beschreiben, wie diese aufeinander aufbauen. Auf die Vorgabe konkreter Arbeitsschritte, mit denen einzelne Ergebnisse erzielt werden können, wird dabei weitgehend verzichtet. In der Praxis finden sich hauptsächlich Mischformen dieser beiden Idealtypen, wobei der ergebnisorientierte Anteil bei Vorgehensmodellen wegen seiner höheren

Allgemeingültigkeit in der Regel überwiegt (vgl. hierzu insbesondere die Äußerungen in [ALLEN und FROST 1998:257f.]).

Um die Aufgaben, die es im Rahmen der komponentenorientierten Entwicklung durch eine entsprechende Methodik zu unterstützen gilt, zu verdeutlichen und dabei die Bedeutung von Komponentenspezifikationen herauszuarbeiten, werden im Folgenden die einzelnen Phasen des komponentenorientierten Entwicklungsprozesses im Detail beschrieben. Als Ausgangspunkt für die nachfolgenden Betrachtungen dient zunächst eine Betrachtung der Vorgehensmodelle, die zur Unterstützung der (betrieblichen) Anwendungsentwicklung im Allgemeinen und speziell im Rahmen der Komponentenorientierung vorgeschlagen wurden. Dabei ist anzumerken, dass neben den betrachteten Vorgehensmodellen selbstverständlich weitere Ansätze existieren. So sind insbesondere im Bereich der Objektorientierung zahlreiche weitere Vorgehensmodelle entstanden [BOOCH 1993:229-266; HENDERSON-SELLERS und EDWARDS 1994:267-270; WILLIAMS 1996:41; MEYER 1997:926-928], die aufgrund ihrer Spezialisierung auf das objektorientierte Entwicklungsparadigma jedoch nicht weiter untersucht werden. Eine ausführliche Darstellung der Genealogie objektorientierter Vorgehensmodelle findet sich bei [NOACK und SCHIENMANN 1999].

2.2.1 Vorgehensmodelle

Im Laufe der Jahre wurde eine Vielzahl unterschiedlicher Vorgehensmodelle vorgeschlagen (vgl. [DAVIS, et al. 1988]), von denen vor allem das *Wasserfallmodell* [ROYCE 1970] und das *Spiralmodell* [BOEHM 1988] als klassische, idealtypische Vorgehensmodelle eine herausragende Bedeutung erlangt haben. Das Wasserfallmodell unterstellt in seiner ursprünglichen Form ein rein *sequenzielles* Vorgehen bei der Anwendungsentwicklung, bei dem einzelne Phasen streng nacheinander durchlaufen werden. Insbesondere ist nach dem Wasserfallmodell also jede Entwicklungsphase vor dem Beginn der darauf folgenden Phase vollständig abzuschließen, so dass die jeweils entstehenden Entwicklungsergebnisse stets als Ganzes weitergegeben werden. Mit den eingeführten Entwicklungsphasen und seiner Prozessstruktur trägt das Wasserfallmodell vor allem zur systematischen Untergliederung des Entwicklungsprozesses bei.

Dennoch ist es in der Praxis vor allem wegen der Tatsache, dass fertig gestellte Software-Artefakte aufgrund der sequenziellen Vorgehensweise erst zu einem späten Zeitpunkt verfügbar werden und Änderungen an den Ergebnissen der einzelnen Phasen, die sich aus dem weiteren Verlauf ergeben, nur mit hohem Aufwand zu realisieren sind, in die Kritik geraten [BOEHM 1988:63; MEYER 1997:925; SAMETINGER 1997:153]. Im Gegensatz zum Was-

serfallmodell beschreibt das Spiralmodell den Anwendungsentwicklungsprozess daher als einen grundsätzlich *zyklischen* Ablauf, bei dem einzelne Phasen bzw. Arbeitsschritte wiederholt durchlaufen werden können.

Mit seiner flexiblen Ablaufstruktur unterstützt das Spiralmodell sowohl die *iterative Entwicklung*, bei der die einzelnen Entwicklungsergebnisse sukzessive verfeinert und verbessert werden [D'SOUZA und WILLS 1999:513], als auch die *inkrementelle Entwicklung*, bei der das Gesamtsystem in mehrere Teile zerlegt wird und in einem evolutionären Prozess entsteht [SCHACH 1990:59; D'SOUZA und WILLS 1999:513f.]. Seit ihrer ursprünglichen Einführung haben die beiden klassischen, idealtypischen Vorgehensmodelle zahlreiche Modifikationen erfahren. So wurde bspw. das Wasserfallmodell um Konzepte des *Prototyping* [BUDDE, et al. 1992] sowie des *phasenübergreifenden Rückschritts* erweitert und so dem Spiralmodell angenähert. Weitere Modifikationen, wie etwa die Einführung *alternativer Pfade*, auf denen die Entwicklungsphasen jeweils durchlaufen werden können, zielen vor allem auf eine Steigerung der Flexibilität von Vorgehensmodellen [HARMSEN, et al. 1994; ORTNER 1998:329; D'SOUZA und WILLS 1999:510; BROWN 2000:153].

Die speziell zur Unterstützung der komponentenorientierten Entwicklung vorgeschlagenen Vorgehensmodelle greifen die bewährten Konzepte der klassischen Vorgehensmodelle auf und spezialisieren diese im Hinblick auf das komponentenorientierte Paradigma. Sie heben sich von klassischen Vorgehensmodellen insbesondere durch die Beschreibung eines speziellen hybriden Entwicklungsprozesses ab, der konzeptionell zwischen der Entwicklung von *Anwendungssystemen* mit Komponenten (der sog. Entwicklung im Großen) auf der einen, sowie der Entwicklung von *Komponenten* (der sog. Entwicklung im Kleinen) auf der anderen Seite unterscheidet [CHAMBERS 1996:76; SAMETINGER 1997:157f.; ALLEN und FROST 1998:261f.; OBERNDORF, et al. 1998; D'SOUZA und WILLS 1999:517-521; HERZUM und SIMS 2000:246f.]. Einige komponentenorientierte Vorgehensmodelle verwenden zur Beschreibung dieses Entwicklungsprozesses zwei ineinander greifende, inhaltlich getrennte *Makroprozesse*, von denen der eine die Anwendungsentwicklung unter Nutzung von Komponenten und der andere das Vorgehen bei der Komponentenentwicklung beschreibt. Durch die so realisierte Entkopplung der beiden Prozesse ergibt sich insbesondere die Möglichkeit, auf deren jeweils unterschiedliche Anforderungen einzugehen.

So beschreibt Sametinger mit seinem *Twin-Life-Cycle* Modell (vgl. Abb. 2.5 links unten) ein Vorgehen, dass primär auf die Entwicklung wieder verwendbarer Komponenten und deren anschließende Nutzung im Rahmen der Anwendungsentwicklung zielt [SAMETINGER 1997:157f.]. Dementsprechend steht hauptsächlich die Komponentenentwicklung im Mittelpunkt, die als *Development for Reuse* systematisch auf die Wiederverwendung ausge-

richtet ist [SAMETINGER 1997:158, 172]. Hierzu schlägt Sametinger neben der *Generalisierung* von Komponenten, die im Rahmen einzelner Anwendungsentwicklungsprojekte identifiziert wurden, vor allem die Durchführung der *Domänenanalyse und -implementierung* vor, mit der sich ganze Anwendungsbereiche methodisch in Komponenten zerlegen lassen [SAMETINGER 1997:160-168].

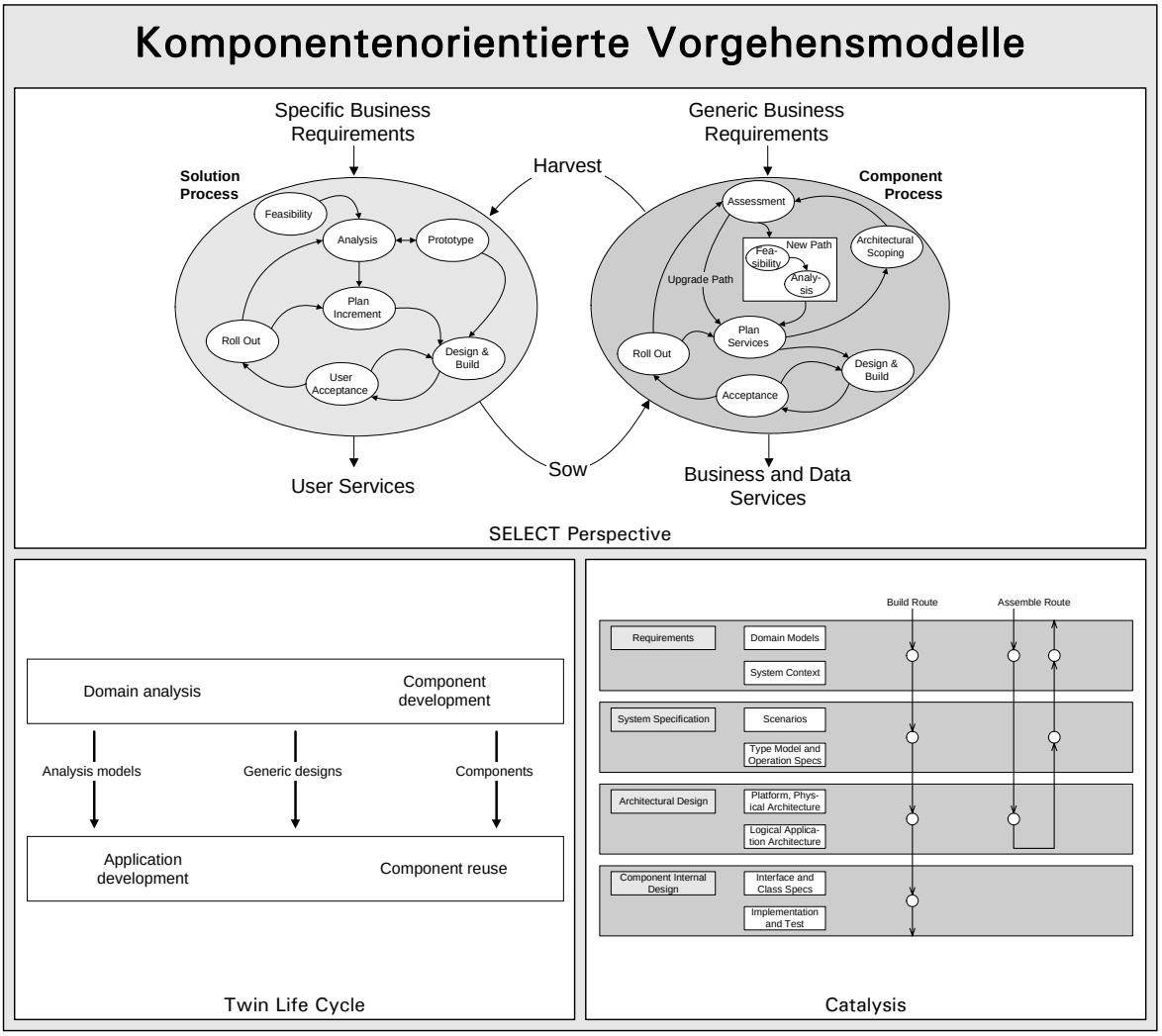


Abb. 2.5: Gegenüberstellung von Vorgehensmodellen für die komponentenorientierte Anwendungsentwicklung (in Anlehnung an [SAMETINGER 1997:158; ALLEN und FROST 1998:264, 275, 309; D'SOUZA und WILLS 1999:512, 517]).

Die einzelnen Projekte der Anwendungsentwicklung greifen dann primär auf die Komponenten zurück, die zuvor im Rahmen der Domänenanalyse bzw. der Komponentenentwicklung bereitgestellt wurden. Die Entwicklung von Anwendungssystemen ist somit als *Development with Reuse* oder *Reuse-Driven Development* ausgelegt [SAMETINGER 1997:158, 185-187]. Dies bedeutet, dass entweder in Abhängigkeit von der zu realisierenden Funktionalität bei der Anwendungsentwicklung möglichst geeignete Komponenten einzubezie-

hen sind (Development with Reuse) oder gar die Funktionalität des jeweiligen Anwendungssystems in Abhängigkeit von den einsetzbaren Komponenten festzulegen und dabei ggf. einzuschränken ist (Reuse-Driven Development). In jedem Fall ist die Anwendungsentwicklung im Twin-Life-Cycle Modell als synthetischer Bottom-Up Prozess stark an bereits den vorhandenen, wieder verwendbaren Komponenten ausgerichtet.

Der Einsatz von Methoden wie der Domänenanalyse, die zur Bereitstellung wieder verwendbarer Komponenten eine umfassende Strukturierung des jeweiligen Anwendungsbereichs durchführen, wird vor allem wegen des zu erwartenden hohen Aufwands und der langen Entwicklungszeit kritisiert [MILI, et al. 1995:544; ALLEN und FROST 1998:206f., 306f.]. Allen und Frost schlagen aus diesem Grund mit ihrem Vorgehensmodell (vgl. Abb. 2.5 oben), das als Bestandteil der *SELECT Perspective* Methodik entwickelt wurde und ebenfalls zwischen einem Anwendungsentwicklungs- (*Solution Process*) sowie einem Komponentenentwicklungsprozess (*Component Process*) unterscheidet [ALLEN und FROST 1998:261-265], eine andere Strategie zur Komponentenfindung vor. Dieser Strategie zufolge wird im Rahmen eines Anwendungsentwicklungsprojekts nicht ausschließlich auf die Ergebnisse vorangehender Komponentenentwicklungsprojekte zurückgegriffen, so dass nicht nur bereits vorhandene Komponenten wieder verwendet werden. Vielmehr bildet jedes Anwendungsentwicklungsprojekt gleichzeitig einen potenziellen Ausgangspunkt für zukünftige Komponentenentwicklungsprojekte. In diesen werden projektübergreifend nutzbare Teile des gerade zu realisierenden Anwendungssystems generalisiert und zu wieder verwendbaren Komponenten weiterentwickelt [ALLEN und FROST 1998:307].

Während der Entwicklung stellt ein genereller architektonischer Rahmen, der für alle Komponentenentwicklungsprojekte verbindlich ist, die Entstehung aufeinander abgestimmter Komponenten sicher [ALLEN und FROST 1998:312-315]. Darüber hinaus wird die inkrementelle Entwicklung von Komponenten unterstützt, so dass diese flexibel um weitere Dienste zu ergänzen sind, falls sich die Notwendigkeit hierzu aus den jeweiligen Anwendungsentwicklungsprojekten ergibt [ALLEN und FROST 1998:261]. Auf diese Weise verbindet das Vorgehensmodell der SELECT Perspective Methodik mit seinen beiden zusammenwirkenden Teilprozessen ein Verfahren zur Anwendungsentwicklung nach einem synthetischen Bottom-Up Prinzip, das zumindest teilweise an den verfügbaren Komponenten ausgerichtet ist, mit einem Verfahren zur Komponentenentwicklung, das eine inkrementelle und zielgerichtete Bereitstellung wieder verwendbarer Komponenten im Bedarfsfall ermöglicht. Der Bedarf an Komponenten wird dabei vor allem anhand der zum jeweiligen Zeitpunkt in einem Unternehmen laufenden bzw. geplanten Anwendungsentwicklungsprojekte und somit unter Berücksichtigung eines *mittelfristigen Planungshorizonts* bestimmt.

Im Gegensatz zu den beiden eben genannten Ansätzen schlagen D'Souza und Wills mit ihrem im Rahmen der *Catalysis* Methodik bereitgestellten Vorgehensmodell (vgl. Abb. 2.5 unten rechts) zwei unterschiedliche Vorgehensweisen vor, mit denen sich jeweils der gesamte Prozess der komponentenorientierten Entwicklung abdecken lässt [D'SOUZA und WILLS 1999:510-512]. Die beiden Vorgehensweisen sind als alternative Pfade eines gemeinsamen Vorgehensmodells ausgelegt und besitzen somit grundsätzlich die gleiche Phasenstruktur [D'SOUZA und WILLS 1999:517f.]. Jedoch weichen sie vor allem in der zeitlichen Anordnung der einzelnen Entwicklungsphasen voneinander ab. So wird mit der sog. *Assemble Route* zum einen eine Vorgehensweise nach einem synthetischen Bottom-Up Prinzip vorgegeben, die die Entwicklung von Anwendungssystemen aus bereits vorhandenen Komponenten beschreibt.

Zum anderen wird mit der sog. *Build Route* eine weitere Vorgehensweise angegeben, die sich am Wasserfallmodell orientiert und die Entwicklung eines Anwendungssystems einschließlich der Komponenten, aus denen es aufgebaut ist, nach einem analytischen Top-Down Prinzip beschreibt. Dabei werden die einzelnen Komponenten eines Anwendungssystems so entworfen und implementiert, dass sie für einen bestimmten Problemkontext optimal ausgelegt sind. Wegen dieses Vorgehens sind die entstehenden Komponenten jedoch üblicherweise nur in eingeschränktem Maße als Kompositionseinheiten wieder verwendbar und dienen somit hauptsächlich als Entwurfs- und Abstraktionseinheiten. Die mit der *Build Route* angegebene Vorgehensweise wurde insbesondere von Cheesman und Daniels im Rahmen ihrer *UML Components* Methodik aufgegriffen und als Grundlage für ein spezielles Vorgehensmodell verwendet, das sie für die Entwicklung flexibel anpassbarer, komponentenorientierter Anwendungssysteme empfehlen [CHEESMAN und DANIELS 2001:26-28]. Demgegenüber wird die Entwicklung und systematische Einbeziehung wieder verwendbarer Komponenten, die von ihrem Vorgehensmodell nicht explizit abgedeckt wird, als zweitrangig betrachtet [CHEESMAN und DANIELS 2001:2].

Den meisten Vorgehensmodellen gemeinsam ist die explizite Forderung nach einer iterativ-inkrementellen Vorgehensweise bei der Entwicklung komponentenorientierter Anwendungssysteme, bei der die einzelnen Komponenten sukzessive zu verfeinern und im Rahmen verschiedener, ggf. parallel ausgeführter Inkremente bereitzustellen sind [ALLEN und FROST 1998:261; D'SOUZA und WILLS 1999:512-514; HERZUM und SIMS 2000:251; CHEESMAN und DANIELS 2001:28-30]. Die frühen Phasen des Entwicklungsprozesses bzw. des jeweiligen Makroprozesses sind hingegen möglichst umfassend durchzuführen, um das für die Planung der einzelnen Inkremente notwendige Verständnis der Anwendungssystem- bzw. Komponentenfunktionalität zu erlangen [ALLEN und FROST 1998:264f., 272, 308]. Ein inkrementelles Vorgehen ist für diese Phasen somit nicht vorgesehen.

2.2.2 Entwicklungsphasen

Insbesondere im Hinblick auf ihre Strukturierung in einen oder mehrere Teilprozesse, die Ausdifferenzierung einzelner Entwicklungsphasen sowie deren zeitlich-inhaltliche Verknüpfung weisen die betrachteten komponentenorientierten Vorgehensmodelle bisweilen jedoch deutlichere Unterschiede auf. Dennoch lassen sich zur Beschreibung des komponentenorientierten Entwicklungsprozesses im Wesentlichen die in Abb. 2.6 schematisch dargestellten Phasen *Voruntersuchung* (Planning bzw. Feasibility Study [ALLEN und FROST 1998:275]), *Fachentwurf* (Analysis [ALLEN und FROST 1998:275] bzw. Domain Modeling [D'SOUZA und WILLS 1999:512]), *Systementwurf* (Application Design [SAMETINGER 1997:186f.; ALLEN und FROST 1998:275] bzw. Logical Application Architecture Design [D'SOUZA und WILLS 1999:512]), *Komponentenentwurf* (Component Internal Design [ALLEN und FROST 1998:309; D'SOUZA und WILLS 1999:512]), *Implementierung* (Implementation [D'SOUZA und WILLS 1999:512] bzw. Component Building [ALLEN und FROST 1998:309]), *Konfigurierung* (System Building [ALLEN und FROST 1998:275], Physical Application Architecture [D'SOUZA und WILLS 1999:512] bzw. Integration [SAMETINGER 1997:186f.]), *Stabilisierung* (Acceptance Testing [ALLEN und FROST 1998:275]) und *Einführung* (Installation bzw. Roll Out [ALLEN und FROST 1998:275]) unterscheiden. Hinzu kommt ggf. die Phase der *Nutzung* (Use), die die Beschreibung des Lebenszyklus vervollständigt und dabei die im Allgemeinen längste Phase darstellt.

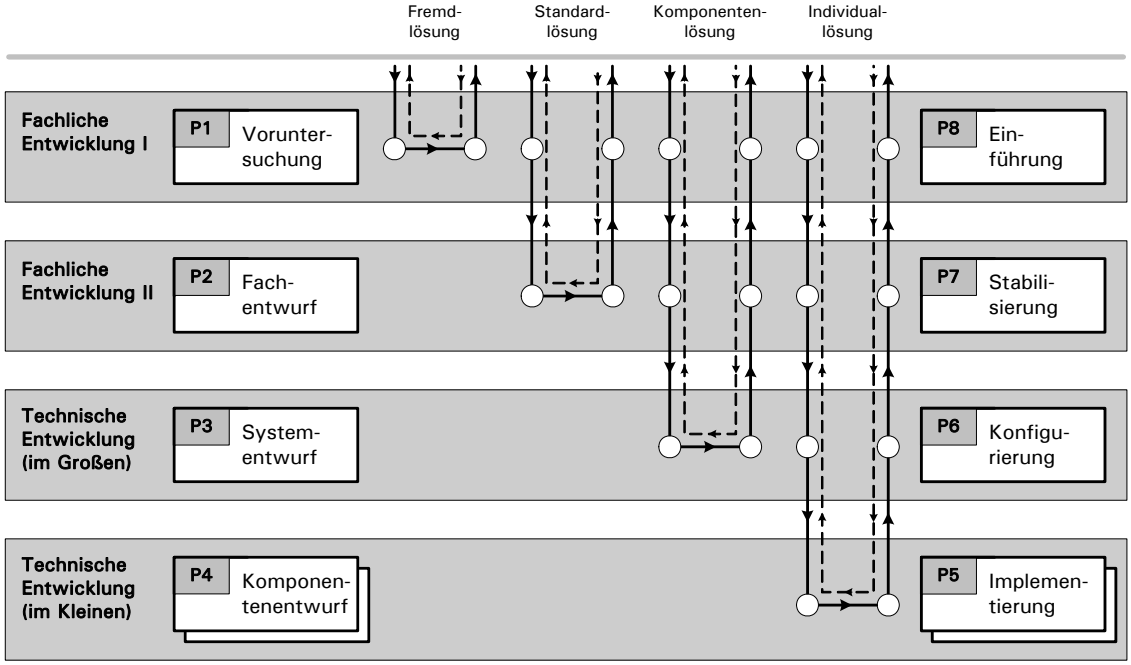


Abb. 2.6: Vereinheitlichtes Vorgehensmodell für die komponentenorientierte Anwendungsentwicklung mit flexibler Anordnung von Entwicklungsphasen (in der Abbildung als P1 – P8 bezeichnet).

Im Rahmen der zu Beginn durchzuführenden *Voruntersuchung* werden die Anforderungen an das zu entwickelnde Anwendungssystem sowie der Ist-Zustand des Problembereichs erfasst. Dabei sind neben dem Ziel, der Vision [CHEESMAN und DANIELS 2001:71], vor allem die funktionalen und nicht-funktionalen (Qualitäts-) Merkmale des zu entwickelnden Anwendungssystems festzulegen. Weitere Aufgaben der Voruntersuchung umfassen Wirtschaftlichkeitsbetrachtungen, eine Analyse des bei der Einführung bzw. Nicht-Einführung des zu entwickelnden Anwendungssystems bestehenden Risikos sowie die Entwicklung grundlegender Lösungskonzepte und Bewertung von Alternativen [PRESSMAN 1997:269-274; ALLEN und FROST 1998:277-281]. Bei der Voruntersuchung ist insbesondere die Abstimmung des geplanten Entwicklungsprojekts mit den übergeordneten Zielen des Unternehmens im Rahmen der *strategischen Informationsplanung* zu gewährleisten [PRESSMAN 1997:257-261]. Falls eine Entscheidung zugunsten der Durchführung des Entwicklungsprojekts gefällt wird, sind darüber hinaus ein grober Projektplan und Regelungen bezüglich der Projektdokumentation zu erstellen. Das Ergebnis der Voruntersuchung ist das *Pflichtenheft*, das im Wesentlichen die eben erwähnten Teile umfasst. Es dient als Grundlage für die weitere Durchführung des Entwicklungsprojekts oder dessen Fremdvergabe an Dritte.

Im *Fachentwurf* wird anschließend das problemorientierte, fachliche Konzept für das zu schaffende Anwendungssystem entwickelt. Mit diesem wird die fachliche Funktionalität, also die Summe der fachlichen Leistungen eines Anwendungssystems festgelegt. Das *Fachkonzept* ist dabei möglichst unabhängig von der Frage der späteren systemtechnischen Realisierung zu erstellen [DAVIS 1993:192; ORTNER 1997:20] und sollte nicht nur grundlegende Anforderungen wie Korrektheit, Vollständigkeit, Eindeutigkeit und Konsistenz erfüllen, sondern darüber hinaus in einer auch für die Systementwicklung geeigneten präzisen Form, also einer *Spezifikationssprache*, erfasst sein (vgl. [LISKOV und BERZINS 1986:3f.; DAVIS 1993:181]). Der Schwerpunkt des Fachentwurfs ist die Klärung und Aufbereitung, d.h. die Rekonstruktion derjenigen *Fachbegriffe* aus dem Anwendungsbereich, die für die mit dem geplanten Anwendungssystem zu unterstützende Informationsverarbeitung relevant sind [ORTNER 1997:20, 49-51; D'SOUZA und WILLS 1999:548; CHEESMAN und DANIELS 2001:68-70]. Darüber hinaus sind während des Fachentwurfs vor allem *fachliche Testfälle* zu erstellen, mit denen das entwickelte Anwendungssystem auf seine materiale Korrektheit, d.h. seine Übereinstimmung mit der fachlichen Praxis im Anwendungsbereich überprüft werden kann [MITTERMEIR 1990:254]. Diese gehen zusammen mit den geklärten Fachbegriffen in das Fachkonzept ein.

Die zu rekonstruierenden Fachbegriffe des jeweiligen Anwendungsbereichs lassen sich im Hinblick auf ihr Bezugsobjekt einteilen in solche, die sich auf *Dinge* (Informationsobjekte bzw. Entitäten), *Handlungen* (Funktionen bzw. Aufgaben) oder *Abläufe* (Prozesse bzw.

Anwendungsfälle) beziehen [ORTNER 1997:84f.; SCHIENMANN 1997:111-113; SCHEER 1998:37]. Um das Ziel der Klärung und Aufbereitung von Fachbegriffen zu erreichen, arbeiten im Fachentwurf üblicherweise verschiedene Beteiligte (Anwender, Fachexperten, Analytiker, Entwickler etc.) in einem sog. *Joint Application Design* (JAD [ALLEN und FROST 1998:287]) zusammen. Ferner lässt sich der Fachentwurf durch die Nutzung von *Inhaltsstandards* wie bspw. fachkonzeptionellen *Referenzmodellen* beschleunigen, die allgemeingültige Begriffsdefinitionen und Zusammenhänge für einen Anwendungsbereich beschreiben. Ein Überblick zum Gebiet der Referenzmodellierung findet sich bei [SCHWEGMANN 1999:53-80; BECKER und SCHÜTTE 2004:65-170]. Zum Vorgehen bei der Rekonstruktion von Begriffen vgl. bspw. [ORTNER 1997; D'SOUZA und WILLS 1999:543-568]. Das ggf. unter Einsatz dieser Methoden erstellte Fachkonzept stellt die Grundlage für den Systementwurf oder die Auswahl einer geeigneten Standard-Software dar, mit der sich die Entwicklungsaufgabe ggf. ohne Eigenentwicklung lösen lässt.

Im Rahmen des *Systementwurfs* erfolgt die Festlegung der *Architektur* des Anwendungssystems, die die Strukturierung des Anwendungssystems in Komponenten sowie deren Eigenschaften und Verbindung zu einem Gesamtsystem beschreibt [BASS, et al. 1998:27; BOOCH, et al. 1999:31; IEEE 2000:9; CLEMENTS, et al. 2003:2-4]. Die Strukturierung erfolgt ggf. in einem rekursiven Prozess, bei dem zunächst Komponenten mit grober Granularität voneinander abgegrenzt und anschließend solange weiter verfeinert werden, bis entweder eine geeignete Komponente wieder verwendet werden kann oder die erreichte Granularität eine effiziente Realisierung ermöglicht [HERZUM und SIMS 2000:38; ATKINSON, et al. 2004:97f.]. Mit der logischen Architektur, der sog. *Software-Architektur*, ist sowohl der logische *Aufbau* des Anwendungssystems in Form eines Bauplans als auch der *Ablauf* in Form eines Prozesses festzulegen [BOOCH, et al. 1999:31; SIEDERSLEBEN 2004:145]. In Abhängigkeit von der Priorität, mit der wieder verwendbare Komponenten in die Anwendungsentwicklung einzubeziehen sind, lassen sich dabei zwei idealtypische Vorgehensweisen zum Entwurf einer Software-Architektur unterscheiden. Beim Vorgehen nach dem sog. *Bottom-Up Prinzip* (Reuse-Driven Development [SAMETINGER 1997:186]) wird die Software-Architektur primär anhand vorgefertigter Komponenten festgelegt, die sich bezogen auf den Problemkontext des Anwendungssystems als wieder verwendbar erweisen. Dabei wird zugunsten einer höheren Wiederverwendungsquote in Kauf genommen, dass die entstehende Architektur aufgrund von Inkompatibilitäten zwischen einzelnen Komponenten und funktionalen Konflikten, die durch das mehrfache Anbieten gleichartiger Dienste entstehen, kompromittiert wird. Im Extremfall kann dieses Vorgehen sogar dazu führen, dass die während der Voruntersuchung ermittelten Anforderungen zugunsten einer möglichst umfangreichen Wiederverwendung zumindest teilweise aufgegeben werden [SAMETINGER 1997:186; BOEHM und ABTS 1999:135; BROWNSWORD, et al. 2000:49].

Demgegenüber erfolgt die Festlegung der logischen Architektur nach dem sog. *Top-Down Prinzip* (Development with Reuse [SAMETINGER 1997:186]) zunächst weitgehend ohne eine Berücksichtigung vorhandener Wiederverwendungsmöglichkeiten. Vielmehr werden ausgehend von den im Fachentwurf rekonstruierten Informationsobjekten, fachlichen Aufgaben und Prozessen einzelne Teile des Anwendungssystems voneinander abgegrenzt, die unter konzeptionell-fachlichen und architektonisch-logischen Gesichtspunkten abgeschlossen sind und somit als eigenständige Komponenten angesehen werden können [HERZUM und SIMS 2000:444-448]. Erst nachdem die logische Architektur vollständig entworfen ist, wird nach vorgefertigten Komponenten gesucht, die zu der jeweils festgelegten Struktur passen und sich somit in die Anwendungsentwicklung einbeziehen lassen [SAMETINGER 1997:186; BROWN 2000:110]. Durch diese Vorgehensweise ist gewährleistet, dass sich eine Architektur ergibt, die die Anforderungen aus der Voruntersuchung optimal erfüllt. Jedoch kann bei einer Entwicklung nach dem Top-Down Prinzip meist nur eine geringere Zahl von Komponenten im Rahmen einer Wiederverwendung erfolgreich in die Anwendungsentwicklung einbezogen werden [SAMETINGER 1997:186].

Bei der Festlegung der logischen Architektur ist insbesondere auch die nach außen sichtbare Struktur und das Verhalten, d.h. die Außensicht der einzelnen Komponenten zu beschreiben [BASS, et al. 1998:27; BOOCH, et al. 1999:31]. Die dabei getroffenen Festlegungen werden als *Komponentenspezifikationen* festgehalten. Sie legen jeweils einen *Soll-Zustand* fest, der von den einzusetzenden Komponenten einzuhalten ist, und stellen somit wesentliche Teilergebnisse des Systementwurfs dar [BROWN 2000:102-104]. Mit einer Spezifikation werden insbesondere die von den Komponenten an den jeweiligen Schnittstellen anzubietenden Dienste, d.h. das, *was* die jeweilige Komponente vollbringt, präzise und möglichst vollständig beschrieben [BOOCH, et al. 1999:31; BROWN 2000:102; CHEESMAN und DANIELS 2001:5]. Bei einem Vorgehen nach dem Top-Down Prinzip sind dabei in der Regel sämtliche Komponenten des zu entwickelnden Anwendungssystems zu spezifizieren. Die erstellten Spezifikationen dienen unter anderem als Grundlage für die Auswahl wieder verwendbarer Komponenten bzw. die Realisierung der spezifizierten Komponenten. Bei einem Vorgehen nach dem Bottom-Up Prinzip sind hingegen nur diejenigen Komponenten zu spezifizieren, die eine noch nicht im Rahmen der Wiederverwendung abgedeckte Funktionalität realisieren. Die Beschreibung der wieder verwendeten Komponenten ist hingegen (im Idealfall) in deren Dokumentation enthalten.

Neben der logischen Architektur, also der Software-Architektur, umfasst das Ergebnis des Systementwurfs als vollständiges *Systemkonzept* auch physische Aspekte der Anwendungssystemarchitektur wie die Beschreibung der technischen Infrastruktur (die sog. Hardware-Architektur [SIEDERSLEBEN 2004:145]), die Verteilung der Komponenten auf

verschiedene Rechner oder die Entscheidung für eine Implementierungsplattform [BROWN 2000:88; CLEMENTS, et al. 2003:167-169]. Bei der Festlegung der Anwendungssystemarchitektur kann ggf. auf vordefinierte *Referenzarchitekturen* und *Architekturstile* [SHAW und GARLAN 1996:191, 196-198; MONROE, et al. 1997] zurückgegriffen werden. Mit diesen lassen sich Anwendungssysteme in spezifische Komponenten strukturieren, die nach einem allgemeingültigen Muster zu einem Gesamtsystem verbunden sind. Daneben kann bei der Spezifikation von Komponenten ggf. auf *Schnittstellenstandards* zurückgegriffen werden. Ein Beispiel für einen solchen Schnittstellenstandard aus dem Bereich der Finanzbuchhaltung findet sich bei [OMG 2001].

Das Systemkonzept bildet die Grundlage für die Konfigurierung von Anwendungssystemen aus vorhandenen Komponenten und die ggf. notwendige Entwicklung einzelner Komponenten. Letztere beginnt mit dem *Komponentenentwurf*, in dessen Rahmen in Abhängigkeit von dem jeweils einzusetzenden Entwicklungsparadigma zunächst die interne, logische *Komponentenarchitektur* entworfen wird. Als Ausgangspunkt für die Festlegung der Komponentenarchitektur dient die während des Systementwurfs in Form einer Spezifikation vorgegebene Außensicht, die durch eine Menge von Modulen bzw. Klassen zu realisieren ist. Während des Komponentenentwurfs ist demnach eine Strukturierung der zu entwickelnden Komponente in Implementierungseinheiten durchzuführen. Dabei ist insbesondere festzulegen, *wie* die Komponente die von ihr anzubietenden Dienste zu vollbringen hat. Darüber hinaus ist die Funktionalität der Komponente ggf. unter Verwendung einer geeigneten Methode zu generalisieren, um ihre Wiederverwendbarkeit auch in anderen Problemkontexten zu gewährleisten [SAMETINGER 1997:173f.]. Bei der Strukturierung einer Komponente empfiehlt sich die Verwendung von *Entwurfsmustern* [PREE 1994; GAMMA, et al. 1995], die bewährte Modul- oder Klassenstrukturen zur Lösung spezifischer Problemstellungen beschreiben [BALZERT 1999:115]. Das Ergebnis des Komponentenentwurfs ist ein *Komponentenkonzept*, das alle notwendigen Vorgaben für die Implementierung der Komponente beinhaltet.

Während der *Implementierung* werden die nunmehr in ihrer Außen- und Innensicht spezifizierten Komponenten unter Verwendung einer Programmiersprache schrittweise ausformuliert und ihre physischen *Programmstrukturen* mittels eines Übersetzers erzeugt. Nach der Definition von DeRemer und Kron entspricht die Implementierung der *Programmierung im Kleinen*, da sie der Realisierung von einzelnen Teilen des Anwendungssystems dient und hierzu schwerpunktmäßig auf deren jeweils beschriebene Innensicht zurückgreift [DEREMER und KRON 1976:114]. Sie lässt sich in ihrer Effizienz vor allem durch den Einsatz programmiersprachenabhängiger *Idiome*, die konkrete Algorithmen für die Lösung spezifischer Programmieraufgaben bereitstellen, sowie von *Modul- bzw. Klassenbibliothe-*

ken, die bereits implementierte Programmfragmente beinhalten, steigern. Nach Abschluss der Implementierung stehen schließlich fertig gestellte *Komponenten* für die Anwendungsentwicklung zur Verfügung. Sie sind im Hinblick auf ihre nach außen sichtbare Struktur und das Verhalten zu dokumentieren, um einen effizienten Zugriff auf diese Informationen während der Anwendungsentwicklung zu gewährleisten. Die *Komponentendokumentation* beschreibt somit den nach der Implementierung erreichten *Ist-Zustand*, der den Vorgaben des während des Systementwurfs festgelegten Soll-Zustands entspricht, jedoch nicht unbedingt mit diesen identisch ist.

Die nunmehr vorhandenen Realisierungen werden im Rahmen der *Konfigurierung* zu einem Gesamtsystem, d.h. dem eigentlich zu entwickelnden Anwendungssystem verbunden. Maßgeblich für den Prozess der Verbindung ist dabei die im Rahmen des Systementwurfs erstellte Anwendungssystemarchitektur. Diese wird im Rahmen der Konfigurierung unter Verwendung spezieller Verbindungssprachen für Komponenten, die entweder als primär aufbauorientierte (strukturbasierte) Architekturbeschreibungssprachen [SHAW und GARLAN 1996:129-146] oder ablauforientierte (prozessbasierte) Verbindungssprachen [NIERSTRASZ 1999:317] ausgelegt sind, schrittweise realisiert, d.h. in eine physische *Anwendungssystemstruktur* transformiert. Dabei sind zwischen einzelnen Komponenten auftretende Inkompatibilitäten und funktionale Konflikte durch die Entwicklung und Zwischenschaltung von *Adaptern* zu beseitigen. Darüber hinaus sind ggf. weitere Anpassungen an einzelnen Komponenten, bspw. durch die Justierung von *Parametern*, vorzunehmen [MILI, et al. 1995:554]. Nach der Definition von DeRemer und Kron entspricht die Konfigurierung der *Programmierung im Großen*, da sie die Verbindung der einzelnen Anwendungsteile unter Verwendung einer speziellen Verbindungssprache zum Gegenstand hat und dabei ausschließlich deren jeweils beschriebene Außensicht verwendet [DEREMER und KRON 1976:114f.]. Auch die Konfigurierung lässt sich durch die Verwendung von *Idiomen*, die konkrete Algorithmen für die Verbindung von Komponenten beinhalten, in ihrer Effizienz steigern.

Mit Abschluss der Konfigurierung ist das zu entwickelnde *Anwendungssystem* schließlich weitgehend fertig gestellt. Es ist im Rahmen der *Stabilisierung* unter Betriebsbedingungen zu testen und auf seine materiale Korrektheit, d.h. die Übereinstimmung mit der fachlichen Praxis im Anwendungsbereich zu überprüfen. Nach der Freigabe für den Betrieb ist das Anwendungssystem im Rahmen der *Einführung* auf dem Zielsystem zu installieren und in die Organisation des Unternehmens zu integrieren. Hieran schließt sich mit der *Nutzung* die im Allgemeinen längste Phase im Lebenszyklus eines Anwendungssystems an. Während dieser Phase kann sich im Laufe der Zeit ein Bedarf zur Weiterentwicklung, ein sog. Anpassungsbedarf, ergeben. Eine solche Weiterentwicklung kann einerseits aus einer *War-*

tung bestehen, bei der einzelne Komponenten gegen neuere Versionen auszutauschen sind. Andererseits kann es durch Veränderungen im Anwendungsbereich jedoch auch notwendig werden, *Erweiterungen* bzw. *Änderungen* am entwickelten Anwendungssystem vorzunehmen. In diesem Fall werden einzelne, ggf. zusätzliche Komponenten neu entwickelt bzw. wieder verwendet und in das Anwendungssystem integriert. Zur Durchführung von Erweiterungen und Änderungen kann es daher notwendig werden, einzelne Phasen des Entwicklungsprozesses oder sogar den gesamten Entwicklungszyklus erneut zu durchlaufen.

Mit der Möglichkeit, Anpassungen im Rahmen der Nutzungsphase durch den Austausch von Komponenten vorzunehmen, ergeben sich vor allem für das *Konfigurationsmanagement*, das zu den phasenübergreifend durchzuführenden Querschnittsfunktionen des Entwicklungsprozesses zählt, zahlreiche neue Anforderungen [LARSSON und CRNKOVIC 1999]. Daneben bilden die *Qualitätssicherung*, das *Konfigurationsmanagement* und die *Dokumentation* weitere wichtige Querschnittsfunktionen. Zusätzlich zu dem im Rahmen dieses Abschnitts beschriebenen Entwicklungsprozess existiert in der Regel auch ein komplementärer Managementprozess, der das *Projekt- und Risikomanagement* zum Gegenstand hat. Dieser Prozess beeinflusst den Entwicklungsprozess nachhaltig und ist dementsprechend grundsätzlich im Zusammenhang mit ihm zu betrachten.

Abschließend ist zu erwähnen, dass auch die Entwicklung von Komponenten für den anonymen (unternehmensinternen oder -externen) Markt, bei der nicht die Entwicklung eines einzelnen Anwendungssystems, sondern die Entwicklung einer Menge von miteinander kombinierbaren Komponenten im Mittelpunkt steht, einem prinzipiell ähnlichen Prozessschema folgt [MILI, et al. 1995:541; BROWN 2000:159]. Unterschiede ergeben sich vor allem durch die Tatsache, dass statt dem für ein einziges Anwendungssystem relevanten Ausschnitt aus dem Anwendungsbereich üblicherweise ein größerer, ggf. sogar der gesamte Anwendungsbereich abgedeckt wird. Dementsprechend wird statt des Fachentwurfs meist eine *Domänenanalyse*, anstelle des Systementwurfs ein umfassender *Domänenentwurf* und statt der Implementierung einzelner Komponenten eine vollständige *Domänenimplementierung* vorgenommen [CZARNECKI und EISENECKER 2000:57]. Eine Konfigurierung der entwickelten Komponenten erfolgt dabei nur zu Testzwecken oder dann, wenn diese gemeinsam als Komponente gröberer Granularität auf den Markt zu bringen sind.

2.2.3 Methoden und Werkzeuge

Mit der Einführung der Komponentenorientierung in die (betriebliche) Anwendungsentwicklung entsteht eine Vielzahl neuer Entwicklungsaufgaben, die es durch geeignete Me-

thoden und Werkzeuge, d.h. mit einer *Methodik*, zu unterstützen gilt. In Abhängigkeit von ihrem Gegenstand lassen sich diese Aufgaben entweder der Komponentenentwicklung, die die Realisierung von (wieder verwendbaren) Komponenten zum Ziel hat, oder der Anwendungsentwicklung, in deren Mittelpunkt die Konfigurierung von Anwendungssystemen aus Komponenten steht, zuordnen. Dementsprechend sind zur Unterstützung der Entwicklungsarbeit zwei spezialisierte *Teilmethodiken* zu schaffen [GRIFFEL 1998:46], von denen jede eine Menge aufeinander abgestimmter Methoden und Werkzeuge bereitstellt [MEYER 1997:664; SZYPERSKI, et al. 2002:457]: eine auf die Komponentenentwicklung sowie eine auf die Anwendungsentwicklung mit Komponenten spezialisierte Teilmethodik, wobei letztere im Folgenden auch als *Kompositionsmethodik* bezeichnet wird.

Zur Realisierung von Komponenten wird in der Regel eine bewährte Implementierungstechnik wie die funktionale oder objektorientierte Programmierung eingesetzt. Dementsprechend weist die Komponentenentwicklung eine starke Ähnlichkeit zur strukturierten bzw. objektorientierten Anwendungsentwicklung auf und lässt sich an vielen Stellen durch Methoden und Werkzeuge unterstützen, die bereits im Zusammenhang mit diesen Entwicklungsparadigmen entstanden sind [GRIFFEL 1998:46]. Unterschiede ergeben sich bei der Komponentenentwicklung zum einen durch die ggf. anzustrebende *Generalisierung*, mit der die Wiederverwendung von Komponenten in verschiedenen Problemkontexten sichergestellt werden soll. Deswegen sind als Bestandteile einer Methodik zur Unterstützung der Komponentenentwicklung zusätzliche Methoden und Werkzeuge bereitzustellen, mit denen die Entwicklung von generalisierten Komponenten unterstützt werden kann. Hierfür finden sich vor allem im Rahmen der *Domänenanalyse* und *Domänenimplementierung* [PRIETO-DÍAZ 1990; SAMETINGER 1997:160-168; CZARNECKI und EISENECKER 2000:19-59] sowie der *Produktlinienentwicklung* [JACOBSON, et al. 1997] wichtige Ansätze, die als Bestandteile in die angestrebte Entwicklungsmethodik aufzunehmen sind.

Zum anderen setzt die Entwicklung von Komponenten auf einer zuvor erstellten *Komponentenspezifikation* auf, die die nach außen sichtbare Struktur sowie das Verhalten der jeweiligen Komponente als *Soll-Zustand* verbindlich vorgibt und bei der Realisierung entsprechend einzuhalten ist. Um diese Anforderung besser zu unterstützen, sind als Bestandteile einer Methodik zur Unterstützung der Komponentenentwicklung vor allem spezielle Methoden zu entwickeln, mit denen sich die Konformität zu der spezifizierten Außensicht während des Entwurfs der internen Komponentenarchitektur und der anschließenden Implementierung sicherstellen bzw. verifizieren lässt. Entsprechende Ansätze hierfür lassen sich aus dem bekannten Konzept der *Verfeinerung einer Spezifikation* (Refinement) ableiten [D'SOUZA und WILLS 1999:214-233, 274-284]. So wird bspw. in [D'SOUZA und WILLS 1999:639-668] ein allgemeingültiges Vorgehen beschrieben, mit dem eine Komponenten-

spezifikation unter Verwendung des objektorientierten Paradigmas systematisch realisiert werden kann. Die dort genannten Ansätze sind jedoch auf den einzusetzenden Spezifikationsrahmen und dessen konkrete Vorgaben zur Beschreibung der Außensicht von Komponenten anzupassen. Darüber hinaus sind sie durch Werkzeuge zu unterstützen, mit denen sich Teile der internen Komponentenarchitektur aus der Spezifikation der Außensicht generieren lassen.

Der zur Beschreibung der Außensicht von Komponenten einzusetzende Spezifikationsrahmen hat somit einen wesentlichen Einfluss auf die Komponentenentwicklung und deren Unterstützung durch geeignete Methoden bzw. Werkzeuge. Gleichermaßen beeinflusst er jedoch auch den Anwendungsentwicklungsprozess, da dessen erfolgreiche Durchführung maßgeblich von der Verfügbarkeit von Informationen über die zu konfigurierenden Komponenten abhängt. Dabei stellen Komponentenspezifikationen in der Regel die einzige auswertbare Informationsquelle dar. Eine Beschaffung von Informationen durch eine Untersuchung der physischen Komponenten während der Anwendungsentwicklung scheidet dagegen aufgrund der Tatsache, dass diese vom Hersteller üblicherweise als Blackbox ausgeliefert werden und daher aufwändige bzw. wenig exakte Methoden des Testens oder Reverse Engineering einzusetzen wären, ebenso aus wie die Verifikation der faktischen Korrektheit von Spezifikationen durch die Anwendungsentwickler [GARLAN, et al. 1995; WEYUKER 2001]. Aus dem gleichen Grund beeinflusst der zur Beschreibung von Komponenten eingesetzte Spezifikationsrahmen nicht nur die Durchführung der Anwendungsentwicklung, sondern auch die Schaffung zahlreicher Methoden und Werkzeuge, mit denen die Anwendungsentwicklung unterstützt werden soll. Da diese nur jeweils solche Informationen über Komponenten nutzen können, die als Bestandteil einer Komponentenspezifikation verfügbar sind, hängt ihre Funktionalität vom eingesetzten Spezifikationsrahmen ab.

Der Spezifikationsrahmen, nach dessen Vorgaben die nach außen sichtbaren Eigenschaften von Komponenten beschrieben werden, stellt somit auch ein zentrales Element der Kompositionsmethodik dar [GARLAN, et al. 1995; D'SOUZA und WILLS 1999:233; BROWN 2000:102; HERZUM und SIMS 2000:20; APPERLY, et al. 2001; CHEESMAN und DANIELS 2001:5; SPEED, et al. 2001; WEYUKER 2001; SZYPERSKI, et al. 2002:36, 50, 412f.; WALLNAU 2003:2, 10]. Mit ihm müssen deshalb neben den Eigenschaften, die für die Komponentenentwicklung relevant sind, auch diejenigen Eigenschaften von Komponenten in angemessener Weise beschrieben werden können, die zur Durchführung der Anwendungsentwicklung bzw. von Wartungsarbeiten benötigt werden [BROWN 2000:103; HOUSTON und NORRIS 2001:244]. Um die sich daraus ergebenden Anforderungen an den Spezifikationsrahmen zu bestimmen, werden im Folgenden zunächst die zu unterstützenden Aufgaben des Anwendungsentwicklungsprozesses näher betrachtet. Dabei kann aufgrund der

Tatsache, dass eine Kompositionsmethodik derzeit noch weitgehend im Entstehen begriffen ist, bei der vorgenommenen Betrachtung zwar kein Anspruch auf Vollständigkeit erhoben werden. Zu den zentralen Bausteinen einer Kompositionsmethodik (vgl. Abb. 2.7) gehören jedoch vor allem Methoden und Werkzeuge für die *Zertifizierung* von Komponenten, die *Katalogisierung* bzw. die *Suche* von Komponenten, die Durchführung *statischer Kompatibilitätstests*, die *Adaptergenerierung*, die *Ableitung der Eigenschaften* von Anwendungssystemen auf Basis der Eigenschaften ihrer Komponenten sowie die *Komposition* von Komponenten [ZAREMSKI und WING 1997:334; CRNKOVIC 2002:132f.].

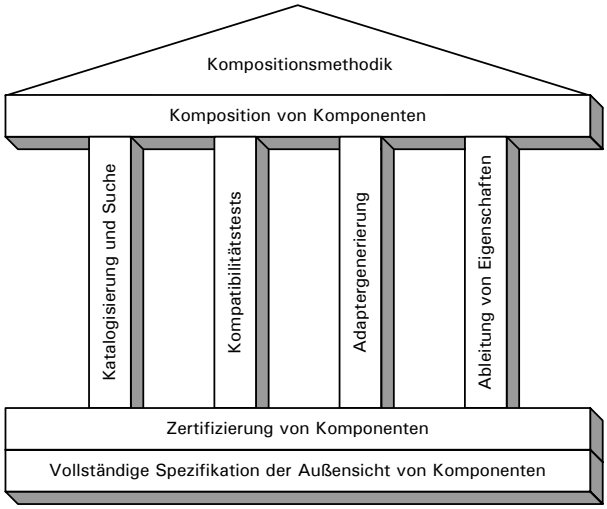


Abb. 2.7: Bausteine einer Kompositionsmethodik für die Anwendungsentwicklung mit Komponenten.

Die Entwicklung von Methoden und Werkzeugen zur *Zertifizierung* von Komponenten dient hauptsächlich der Steigerung des Vertrauens in die jeweils als Spezifikation bereitgestellten Informationen. Im Rahmen der Zertifizierung wird folglich vor allem die Übereinstimmung von Komponenten mit ihrer jeweils spezifizierten Außensicht untersucht und durch ein entsprechendes Zertifikat bescheinigt [IEEE 1991; APPERLY, et al. 2001:769; FLYNT und DESAI 2001:701; MORRIS, et al. 2001]. Damit wird ein gewisser Ausgleich für die zuvor beschriebenen Schwierigkeiten bei der Verifizierung einer Spezifikation durch die Anwendungsentwickler geschaffen.

Demgegenüber stellt die Entwicklung leistungsfähiger Methoden und Werkzeuge zur Katalogisierung von Komponenten sowie zur Suche nach ihnen vor allem eine wesentliche Voraussetzung für den Aufbau von Verzeichnissen und Bibliotheken (sog. Repositorien) dar [SAMETINGER 1997:178f.; MILI, et al. 1998]. Mit diesen lässt sich während des Systementwurfs eine effiziente Auswahl von Komponenten für die Wiederverwendung realisieren, weshalb sie ein wichtiges Hilfsmittel für die komponentenorientierte Anwendungsentwicklung bilden [SAMETINGER 1997:178; VITHARANA, et al. 2003:649f.]. Hierzu wer-

den Komponenten während der *Katalogisierung* zunächst bezüglich ihrer Eigenschaften nach einem *Klassifikationsschema* angeordnet [PRIETO-DÍAZ und FREEMAN 1987; PRIETO-DÍAZ 1991; SAMETINGER 1997:179-184] und in einem Repositorium abgelegt [VITHARANA, et al. 2003:651f.]. Die Qualität der mit der Katalogisierung von Komponenten erzielbaren Ergebnisse hängt maßgeblich vom *Umfang* der bereitgestellten Informationen über Komponenten ab, die zur Klassierung verwendet werden können.

Während der *Suche* werden die Eigenschaften der in einem Repositorium angeordneten Komponenten dann mit den Kriterien verglichen, die im Rahmen einer Anfrage formuliert wurden. Dabei kann grundsätzlich zwischen textbasierten, klassifikationsbasierten und spezifikationsbasierten Vorgehensweisen unterschieden werden, die vor allem im Hinblick auf die Komplexität und die Qualität der ermittelten Ergebnisse voneinander abweichen [FRAKES und POLE 1994; MILI, et al. 1995:547, 550]. Textbasierte Suchmethoden überprüfen die katalogisierten Informationen über Komponenten lediglich auf das Vorhandensein von Schlüsselwörtern, die im Rahmen einer Abfrage vorgegeben wurden. Dagegen unterstützen klassifikationsbasierte Methoden ein differenzierteres Vorgehen bei der Suche nach Schlüsselwörtern, etwa indem sie die Vorgabe unterschiedlicher Schlüsselwörter erlauben und mit diesen jeweils unterschiedliche Facetten (Aspekte) der katalogisierten Information durchsuchen [PRIETO-DÍAZ 1991; MILI, et al. 1995:550f.].

Die besten Ergebnisse lassen sich in der Regel jedoch durch die Vorgabe einer Referenzspezifikation erzielen, die im Rahmen einer Suche mit den Spezifikationen der einzelnen Komponenten detailliert verglichen wird [MILI, et al. 1995:551f.]. Somit sind spezifikationsbasierte Suchmethoden den anderen meist überlegen. Im Gegensatz zu diesen Methoden besitzen sie jedoch eine in der Regel deutlich höhere Komplexität, die durch den detaillierten Vergleich von Spezifikationen entsteht [MILI, et al. 1995:552]. Daher kommen bei der Entwicklung von Suchmethoden hauptsächlich *relaxierende Vergleichsverfahren* zum Einsatz [ZAREMSKI und WING 1995:147; ZAREMSKI und WING 1997:334]. Diese besitzen eine deutlich geringere Komplexität, liefern jedoch üblicherweise nur eine als Vorauswahl potenziell geeigneter Komponenten, die im Folgenden genauer zu untersuchen sind [KONTIO 1996]. Die Qualität der mit spezifikationsbasierten Suchmethoden erzielbaren Ergebnisse hängt jedoch nicht nur vom eingesetzten Vergleichsverfahren, sondern vor allem auch vom Umfang und der (effizienten) statischen *Vergleichbarkeit* der jeweils einzubeziehenden Informationen ab [MILI, et al. 1995:552].

Eine wesentliche Abhängigkeit von der Vergleichbarkeit der zur Verfügung gestellten Informationen besteht auch bei den Methoden und Werkzeugen, mit denen *statische Kompatibilitätstests* durchgeführt werden können. Ähnlich wie spezifikationsbasierte Suchmetho-

den nutzen diese die in Komponentenspezifikationen enthaltenen Informationen zur Ermittlung der Kompatibilität, indem sie einen detaillierten Vergleich zweier Spezifikationen durchführen [ZAREMSKI und WING 1995; YELLIN und STROM 1997; ZAREMSKI und WING 1997; BRADA 2001:2; PLASIL und VISNOVSKY 2002].[6] Im Unterschied zur Vorgehensweise bei der Suche nach Komponenten, bei der die effiziente Ermittlung einer Menge potenziell geeigneter Komponenten im Mittelpunkt steht, kommen im Rahmen von Kompatibilitätstests jedoch möglichst *exakte Vergleichsverfahren* zum Einsatz [ZAREMSKI und WING 1995:150f.; ZAREMSKI und WING 1997:340-342], mit denen sich eine umfassende und korrekte Aussage über die Kompatibilität zweier Spezifikationen ermitteln lässt.

Mit den eingesetzten Vergleichsverfahren lässt sich bestimmen, ob eine Spezifikation als Ganzes oder teilweise zu einer anderen konform ist, wobei im Allgemeinen zwischen verschiedenen Konformitätsklassen unterschieden werden kann [LARSSON und CRNKOVIC 1999:235f.; BRADA 2001:4f.]. Basierend auf der ermittelten Aussage zur Konformität lässt sich dann bspw. feststellen, ob eine Komponente entsprechend den Vorgaben einer Referenzspezifikation aus dem Systementwurf in ein Anwendungssystem integriert werden kann (*Verwendbarkeit*), ob zwei Komponenten während der Konfigurierung miteinander verbunden werden können (*Kombinierbarkeit*) oder eine Komponente im Rahmen von Wartungsarbeiten durch eine andere ersetzt werden kann (*Ersetzbarkeit*) [LARSSON und CRNKOVIC 1999:235; BRADA 2001:1; CHEESMAN und DANIELS 2001:164].

Eine Vielzahl der im Rahmen von Kompatibilitätstests entdeckten Inkompatibilitäten zwischen Komponenten lässt sich durch die Zwischenschaltung geeigneter Adapter beseitigen [YELLIN und STROM 1994:176; ZAREMSKI und WING 1995:147f.; YELLIN und STROM 1997:293; ZAREMSKI und WING 1997:334], deren Entwicklung somit eine wichtige, durch geeignete Methoden und Werkzeuge zu unterstützende Teilaufgabe des komponentenorientierten Entwicklungsprozesses darstellt. Zur Unterstützung dieser Aufgabe sind vor allem Verfahren für eine systematische und zumindest teilweise automatisierbare *Adaptergenerierung* [YELLIN und STROM 1997:293] zu entwickeln, mit denen zunächst der vorhandene Anpassungsbedarf aus den Spezifikationen zweier Komponenten im Rahmen einer Analyse berechnet werden kann. In Abhängigkeit von den jeweils ermittelten Ergebnissen ist dann entweder zu entscheiden, ob entsprechende Maßnahmen zur Ableitung eines geeigneten Adapters zur Anwendung gebracht werden können, oder festzustellen,

[6] Hingegen basieren *dynamische Kompatibilitätstests* vor allem auf Integrationstests, die mit der zu testenden Komponente während der Komposition durchgeführt werden. Zur Kritik an dynamischen Testmethoden und ihren Ergebnissen vgl. [WEYUKER 2001; OVERHAGE und THOMAS 2004:110f.].

dass keine Anpassung ausgeführt werden kann [YELLIN und STROM 1997:321; CHEESMAN und DANIELS 2001:164].

Eine ausführliche Untersuchung von Techniken, die bei der Entwicklung von Methoden zur Generierung von Adaptern eingesetzt werden können, sowie eine Zusammenstellung von Anforderungen, die bei der Entwicklung von Methoden zur Anpassung von Komponenten generell zu erfüllen sind, findet sich bei [BOSCH 2000]. Neben der Beseitigung von Inkompatibilitäten zwischen Komponenten sind als weitere Aufgabenbereiche für die Adaptergenerierung vor allem die gezielte Veränderung der Funktionalität und der Qualität des Gesamtsystems zu nennen. So lassen sich durch die Zwischenschaltung von Adaptern bspw. *funktionale Konflikte* beheben, die bei redundant angebotenen Diensten verschiedener Komponenten entstehen [FERSTL, et al. 1997:26; STETS, et al. 1999:60; TUROWSKI 2003:168-174]. Maßgeblich hängt der durch eine Adaptergenerierung erzielbare Erfolg jedoch generell vom Umfang der jeweils als Spezifikation bereitgestellten Informationen über die anzupassenden Komponenten sowie von ihrer (effizienten) *Verwendbarkeit zur Berechnung* des Anpassungsbedarfs ab [YELLIN und STROM 1997:293].

Die effiziente Verwendbarkeit von Komponentenspezifikationen im Rahmen von Berechnungen ist ebenso eine Voraussetzung für die Entwicklung von Methoden und Werkzeugen, mit denen eine *Ableitung der Eigenschaften* des zu entwickelnden Anwendungssystems auf Basis der Eigenschaften der zu verbindenden Komponenten durchgeführt werden kann. Hierdurch wird es möglich, die aus der Verbindung von Komponenten resultierenden Eigenschaften des Gesamtsystems im Voraus, d.h. vor der Konfigurierung zu ermitteln [WALLNAU, et al. 2001; DUMITRASCU, et al. 2003:3-5; WALLNAU 2003:10f.]. So lässt sich bereits während des Systementwurfs verifizieren, ob eine komponentenorientierte Anwendungssystemarchitektur in der Lage sein wird, die während der Voruntersuchung bzw. des Fachentwurfs ermittelten Anforderungen zu erfüllen [MILI, et al. 1995:552]. Die Vorhersageverfahren sind dabei prinzipiell gleichermaßen dazu geeignet, sowohl die Strukturierung eines Anwendungssystems in Komponenten, die sich bei einem Top-Down Vorgehen ergibt, als auch die Auswahl einer Komponentenmenge bei einem Bottom-Up Vorgehen im Hinblick auf ihre Eignung für die Anwendungsentwicklung zu überprüfen.

Zur Bestimmung von Systemeigenschaften werden die in den Spezifikationen enthaltenen Informationen über Komponenten gemäß der Architektur des jeweiligen Anwendungssystems miteinander verknüpft und unter Verwendung eines *Vorhersagemodells*, das als zentrales Element mit der jeweils anzuwendenden Methode bereitzustellen ist, ausgewertet [LAU und ORNAGHI 2001; REUSSNER und SCHMIDT 2002; WALLNAU 2003:29-31]. Die mit den zu entwickelnden Methoden erzielbare Vorhersagequalität ist dabei grundsätzlich von

der *Kompositionalität*, d.h. der erwartungstreuen Zusammensetzbarkeit der jeweils spezifizierten Komponenteneigenschaften abhängig. Sie ist eine wichtige Voraussetzung, um durch die Verknüpfung einzelner Komponenteneigenschaften Rückschlüsse auf die Eigenschaften des entstehenden Systems ziehen zu können [LEAVENS 1991; WALLNAU 2003:59-63]. Darüber hinaus wird die Vorhersagequalität wesentlich von den jeweils verwendeten Vorhersagemodellen beeinflusst.

Derzeit steht vor allem die Entwicklung geeigneter Modelle und Verfahren, mit denen sich die nicht-funktionalen (Qualitäts-) Eigenschaften von Anwendungssystemen vorhersagen lassen, im Mittelpunkt des wissenschaftlichen Interesses [WALLNAU, et al. 2001; DUMITRASCU, et al. 2003:1f.; WALLNAU 2003:29]. Daneben gilt jedoch auch die zuverlässige Vorhersage der bei der Komposition von Komponenten entstehenden funktionalen Eigenschaften, d.h. der Funktionalität, als ein noch zu erforschendes Aufgabengebiet [LAU und ORNAGHI 2001; REUSSNER und SCHMIDT 2002]. Eine grundsätzliche Herausforderung bei der Entwicklung von Methoden zur Bestimmung von Systemeigenschaften stellt die Berücksichtigung von Veränderungen dar, die sich durch das Zwischenschalten von Adaptern während der Konfigurierung ergeben. Diese sind durch die eingesetzten Vorhersagemodelle zu berücksichtigen und zu prognostizieren.

Zur Unterstützung der komponentenorientierten Anwendungsentwicklung sind schließlich auch Methoden und Werkzeuge zu schaffen, mit denen sich die *Komposition* von Komponenten unterstützen lässt. Hierzu zählt bspw. die Entwicklung spezieller *Verbindungssprachen*, wobei prinzipiell zwischen primär aufbauorientierten (strukturbasierten) Architekturbeschreibungssprachen [LUCKHAM, et al. 1995; SHAW und GARLAN 1996:129-146; MEDVIDOVIC und TAYLOR 1997; CARRIÈRE 1998] und stärker ablauforientierten (prozessbasierten) Verbindungssprachen [PINTADO 1995; NIERSTRASZ 1999:317] unterschieden werden kann. Als Grundlage für den Einsatz dieser Sprachen im Rahmen einer methodischen Komposition sind vor allem entsprechende *Komponenten- bzw. Prozessmodelle* zu entwickeln, mit denen sich die Verknüpfung von Komponenten abbilden lässt [SATTLER 1997]. Während der Komposition werden die als Spezifikationen verfügbaren Informationen über die einzelnen Komponenten ausgewertet und für eine (automatisierte) Verknüpfung gemäß dem jeweiligen Modell verwendet. Im Mittelpunkt des wissenschaftlichen Interesses steht dabei vor allem die Entwicklung spezieller *Komponententypsysteme*, mit denen der Kompositionsprozess abgebildet und erklärt werden kann [SECO und CAIRES 2000; LEE und XIONG 2001; WALLNAU 2003:62f., 65]. Dementsprechend ist darauf zu achten, dass sich die spezifizierten Eigenschaften von Komponenten im Idealfall stets auch im Rahmen entsprechender Typkalküle verwenden lassen, wodurch sich neben der *Kompositionalität* in der Regel weitere Anforderungen ergeben [WALLNAU 2003:63].

2.3 Spezifikationsrahmen

Nachdem die zentrale Bedeutung, die die Einführung eines (standardisierten) Spezifikationsrahmens für die komponentenorientierte Entwicklung besitzt, hervorgehoben und die mit dem Spezifikationsrahmen zu unterstützenden Entwicklungsaufgaben näher beschrieben wurden, ist nunmehr die Entwicklung des UNSCOM Spezifikationsrahmens als eigenständiger Ansatz zu begründen. Hierzu werden zunächst die grundlegenden Anforderungen, die an einen zur Beschreibung der Außensicht von Komponenten einzusetzenden Spezifikationsrahmen zu stellen sind, beschrieben und als Kriterien zur Beurteilung bereits existierender Ansätze verwendet. Die dabei identifizierten Defizite der bestehenden Ansätze bilden die Motivation für die Erarbeitung eines Lösungskonzepts, mit dem diese beseitigt oder zumindest vermindert werden können. Dieses Lösungskonzept bildet die Grundlage für die Entwicklung des UNSCOM Spezifikationsrahmens, der in den nachfolgenden Kapiteln im Detail vorgestellt wird.

2.3.1 Grundlegende Anforderungen

Bei der Entwicklung nach dem komponentenorientierten Paradigma kommt der *Spezifikation von Komponenten*, die sowohl zur Durchführung der Komponentenentwicklung als auch der Anwendungsentwicklung benötigt wird, eine herausragende Bedeutung zu: im Hinblick auf die Komponentenentwicklung ist die Spezifikation dabei *präskriptiv*, d.h. sie definiert die Vorgaben, die bei einer anforderungsgerechten Realisierung von Komponenten einzuhalten sind. Bezogen auf den Prozess der Anwendungsentwicklung ist sie hingegen *deskriptiv*, d.h. sie stellt die benötigten Informationen über die (wieder) zu verwendenden Komponenten in expliziter Form bereit. Allgemein lässt sich der Begriff „Spezifikation“ in Anlehnung an [LISKOV und BERZINS 1986:3] zunächst wie folgt definieren:

> Definition 2.1: Spezifikation
>
> Eine Spezifikation beschreibt die von einem Programm in einem bestimmten Bedingungsrahmen bereitgestellten Funktionen in einer präzisen, von ihrer jeweiligen Realisierung (Implementierung) unabhängigen Form. Sie gibt an, *was* ein Programm vollbringt, ohne dabei näher darauf einzugehen, *wie* es dies erreicht.

Diese Definition lässt sich auf den Komponentenbegriff, wie er in den vorigen Abschnitten eingeführt und benutzt wurde, spezialisieren. Entsprechend ergibt sich in Anlehnung an [BEUGNARD, et al. 1999:38; D'SOUZA und WILLS 1999:202; ACKERMANN, et al. 2002:3] folgende Definition für die Spezifikation einer Komponente:

Definition 2.2: Spezifikation einer Komponente (Komponentenspezifikation)

Die Spezifikation einer Komponente beschreibt deren Außensicht in einer präzisen, von der jeweiligen Realisierung (Implementierung) unabhängigen Form, d.h. sie beschreibt die Dienste, die eine Komponente in einem bestimmten Bedingungsrahmen bereitstellt. Sie gibt an, *was* eine Komponente vollbringt, ohne dabei näher darauf einzugehen, *wie* sie dies erreicht.

Die Spezifikation einer Komponente wird bisweilen auch als *Repräsentation*, *Dokumentation* oder *Beschreibung* einer Komponente bezeichnet, wobei die verwendeten Wörter prinzipiell als Synonyme betrachtet werden können. Dabei hebt der Terminus „Repräsentation" vor allem den abstrakten Modellcharakter der dargestellten Außensicht hervor. Als Repräsentation werden daher hauptsächlich *präskriptive* Beschreibungen bezeichnet, die bspw. im Rahmen des Systementwurfs erstellt werden. Im Gegensatz hierzu unterstreicht die Verwendung des Bezeichners „Dokumentation" ausdrücklich die *deskriptive* Funktion, bei der Eigenschaften einer vorhandenen Komponente beschrieben werden. Somit eignet sich neben dem Terminus „Spezifikation" vor allem das Wort „Beschreibung", um die Darstellung der Außensicht einer Komponente unabhängig von ihrem Verwendungszweck zu charakterisieren. Im Rahmen dieser Arbeit wird jedoch der Terminus „Spezifikation" bevorzugt, da dieser im Allgemeinen eine höhere Präzision gegenüber einer Beschreibung impliziert. Dabei soll nicht geleugnet werden, dass auch der Bezeichner „Spezifikation" gelegentlich zu unbeabsichtigten Auslegungen führt, da er häufig in Verbindung mit dem Einsatz formaler Sprachen verwendet wird. Diese spielten bei der Entwicklung des UNSCOM Spezifikationsrahmens jedoch nur eine untergeordnete Rolle. Zur ausführlichen Diskussion der verschiedenen Bezeichner und ihrer jeweiligen Implikationen vgl. [CLEMENTS, et al. 2003:8f.]

Aufbauend auf der zuvor genannten Definition 2.2, in der vor allem die *präzise* Form betont wird, wurden im Rahmen der klassischen Anwendungsentwicklung weitere wichtige Eigenschaften von Spezifikationen definiert, die sich auf Komponentenspezifikationen übertragen lassen. Demgemäß ist die Außensicht einer Komponente durch eine Spezifikation vor allem in *korrekter*, *vollständiger*, *konsistenter*, *eindeutiger*, *verständlicher*, *prüfbarer*, *kommentierter*, *leicht veränderbarer* und *begründbarer* Weise zu beschreiben, damit sie als Anforderungsdefinition für die *Komponentenentwicklung* verwendet werden kann (zur Begründung vgl. [THAYER und DORFMAN 1990; DAVIS 1993:181; PRESSMAN 1997:304f.]). Aus dem Ziel, die erstellten Komponentenspezifikationen auch als Informationsquelle im Rahmen des *Anwendungsentwicklung* verwenden zu können, ergeben sich darüber hinaus zusätzliche zu erfüllende Eigenschaften. Insbesondere sind Komponentenspezifikationen zu diesem Zweck in *einheitlicher, vergleichbarer* und *zusammensetzbarer* Weise zu verfassen (zur Begründung vgl. Abschnitt 2.2.3).

Um sicherzustellen, dass die erstellten Spezifikationen über die gewünschten Eigenschaften verfügen, ist der Spezifikationsprozess durch *methodische Vorgaben* zu unterstützen, die zu einem *Spezifikationsrahmen* bzw. Rahmenwerk zusammengefasst werden:

> Definition 2.3: Spezifikationsrahmen
>
> Ein Spezifikationsrahmen bestimmt durch verschiedene Vorgaben, welche Eigenschaften eines Software-Artefakts in welcher Form zu beschreiben und wie die einzelnen Teilbeschreibungen zu erarbeiten sowie anschließend zu einem Gesamtergebnis zu verbinden sind.

Grundsätzlich lassen sich dabei Vorgaben unterscheiden, mit denen der *Inhalt* einer Spezifikation, die bei der Beschreibung einzusetzenden *Notationen* (Spezifikationssprache) oder die bei der Ermittlung der zu beschreibenden Informationen anzuwendende *Vorgehensweise* (Spezifikationsmethode) festgelegt wird. Ähnlich wie bei Vorgehensmodellen sind ergebnisorientierte Vorgaben, mit denen der Inhalt von Spezifikationen sowie die einzusetzenden Notationen normiert werden, aufgrund ihrer höheren Allgemeingültigkeit in der Regel leichter durchzusetzen als aktivitätsorientierte Vorgaben, mit denen die anzuwendende Vorgehensweise beschrieben wird. Eine Vielzahl von Vorschlägen zur Spezifikation von Komponenten befasst sich aus diesem Grund primär mit der Entwicklung ergebnisorientierter Vorgaben (vgl. bspw. [HAN 1998; BEUGNARD, et al. 1999; HOUSTON und NORRIS 2001; ACKERMANN, et al. 2002]).

Aus den gewünschten Eigenschaften von Komponentenspezifikationen lassen sich die folgenden *grundlegenden Anforderungen* an einen Spezifikationsrahmen bzw. die in ihm enthaltenen Vorgaben ableiten:

- **Angemessenheit.** Um den bei der Spezifikation zu betreibenden Aufwand angemessen zu begrenzen, sollten die gemäß den Vorgaben des Spezifikationsrahmens erstellten Komponentenspezifikationen sowohl *bei der Komponenten- als auch der Anwendungsentwicklung einsetzbar* sein. Daher muss der Spezifikationsrahmen zunächst eine Beschreibung der Außensicht von Komponenten mit möglichst *hoher Präzision* sicherstellen [LISKOV und BERZINS 1986:3; DAVIS 1993:181]. Darüber hinaus sind jedoch auch die *Durchführbarkeit von statischen Tests und Berechnungen* mit den entstehenden Spezifikationen zu gewährleisten. Da sich inhaltliche Vorgaben im Allgemeinen umso weniger für eine effiziente Auswertung eignen, je komplexer der mit ihnen beschreibbare Sachverhalt ist, muss der Spezifikationsrahmen folglich ein *Gleichgewicht zwischen der Ausdrucksmächtigkeit und der Auswertbarkeit* von Komponentenspezifikationen anstreben. Dabei wird die realisierbare Ausdruckskraft vor allem durch die Berücksichtigung der Forderungen nach statischer Vergleichbarkeit und Kompositionalität (erwartungstreuer Zusammensetzbarkeit) von Spezifikationen beeinträchtigt.

- **Vollständigkeit.** Der Spezifikationsrahmen sollte durch seine Vorgaben die Beschreibung *aller nach außen sichtbaren Eigenschaften* von Komponenten gewährleisten, die zur *Durchführung der Entwicklungsarbeit* benötigt werden. Dabei ist sicherzustellen, dass mit der Spezifikation einerseits alle notwendigen Informationen für eine *anforderungsgerechte Komponentenentwicklung* bereitgestellt werden [LISKOV und BERZINS 1986:3]. Andererseits ist auch die geplante Verwendung von Spezifikationen während der *Anwendungsentwicklung* zu berücksichtigen. So muss es insbesondere möglich sein, mit ihnen die *Eignung einer Komponente* zu beurteilen bzw. durch Anpassungsmaßnahmen herzustellen [BROWN 2000:103; HOUSTON und NORRIS 2001:244f.]. Auch wenn die genannten Verwendungen bei der Entwicklung des Spezifikationsrahmens berücksichtigt werden, bleibt allerdings festzuhalten, dass sich die Erfüllung der Forderung nach Vollständigkeit im Allgemeinen nicht absolut, sondern nur mit relativer Sicherheit, bspw. anhand der jeweils unterstützten Aufgaben belegen lässt.

- **Abgestimmtheit.** Der Spezifikationsrahmen sollte eine explizite Einordnung der zu spezifizierenden Sachverhalte vornehmen und dabei auch die Zusammenhänge zwischen den einzelnen Bestandteilen einer Spezifikation klären [DAVIS 1993:181; D'SOUZA und WILLS 1999:321]. Hierdurch wird zum einen die anzustrebende *Konsistenz* (Widerspruchsfreiheit) von Komponentenspezifikationen unterstützt. Zum anderen tragen die geklärten Zusammenhänge vor allem zu einer besseren *Veränderbarkeit* des Spezifikationsrahmens bei, da bestehende Abhängigkeiten zwischen seinen einzelnen Teilen im Falle einer Änderung berücksichtigt werden können. Darüber hinaus sollte der Spezifikationsrahmen bei der Einordnung der zu spezifizierenden Sachverhalte sicherstellen, dass Informationen nur einmal beschrieben werden. Damit wird vor allem zur angestrebten *Widerspruchsfreiheit* von Spezifikationen beigetragen.

- **Universalität.** Der Spezifikationsrahmen ist möglichst *unabhängig von Komponententechnologien oder speziellen Komponentenarten* als universelles, generisches Instrument zur Beschreibung der Außensicht von Komponenten auszulegen [HOUSTON und NORRIS 2001:244]. Die Erfüllung dieser Anforderung sichert vor allem die Verwendbarkeit des Spezifikationsrahmens als Grundlage für die Entwicklung generischer Methoden und Werkzeuge. Hierdurch wird die Entstehung einer *allgemeingültigen*, *universell anwendbaren Methodik* begünstigt, mit der sich der komponentenorientierte Entwicklungsprozess unabhängig von konkreten Technologien unterstützen lässt.

- **Verbindlichkeit.** Der Spezifikationsrahmen sollte Vorgaben enthalten, mit denen der *Inhalt* und das *Format* von Komponentenspezifikationen verbindlich festgelegt werden. Die entsprechenden Vorgaben sind notwendig, um die *Einheitlichkeit* von Kom-

ponentenspezifikationen zu gewährleisten, die eine wesentliche Voraussetzung für deren Verwendbarkeit durch andere Methoden und Werkzeuge ist. Darüber hinaus lässt sich auch der Spezifikationsprozess mit diesen Vorgaben unterstützen, da sie gleichzeitig als *Richtlinien zur Spezifikation* von Komponenten dienen. Somit trägt die Forderung nach verbindlichen Vorgaben auch zur besseren *Begründbarkeit* von Komponentenspezifikationen bei. Dabei sollte auf eine Normierung der bei der Spezifikation anzuwendenden *Vorgehensweisen*, die ebenfalls zur Unterstützung des Spezifikationsprozesses geeignet wäre, jedoch aufgrund der weniger hohen Allgemeingültigkeit verzichtet werden.

- **Verständlichkeit.** Die gemäß dem Spezifikationsrahmen entstehenden Komponentenspezifikationen sollten sowohl *maschinell auswertbar* als auch für menschliche Entwickler *lesbar* sein [BROWN 2000:102]. Die Forderung nach der maschinellen Auswertbarkeit ist vor allem eine Voraussetzung für deren Verwendbarkeit durch Werkzeuge. Eine Konsequenz, die sich daraus ergibt, ist der Einsatz möglichst *formaler Notationen*, deren Syntax und Semantik präzise definiert ist. Gleichzeitig sollten Komponentenspezifikationen jedoch auch effizient von Entwicklern auszuwerten sein. Dies ist notwendig, um den prinzipiell nur teilweise automatisierbaren Entwicklungsprozess durchgängig zu unterstützen. Zudem ist eine Unterstützung durch Werkzeuge für den komponentenorientierten Entwicklungsprozess derzeit nicht gegeben, so dass die Auswertung von Spezifikationen durch Entwickler im Vergleich zu deren Verwendung in Werkzeugen überwiegt. Bei der Entwicklung des Spezifikationsrahmens ist deshalb ein *Gleichgewicht zwischen Formalität und Lesbarkeit* von Komponentenspezifikationen anzustreben. Denn obgleich formale Spezifikationen prinzipiell auch von Entwicklern ausgewertet werden können [HALL 1990], ergeben sich dabei in der Praxis häufig Schwierigkeiten. Deshalb ist die Lesbarkeit formaler Spezifikationen zumindest durch *Kommentare* zu verbessern [HALL 1990:16f.].

- **Veränderbarkeit.** Der Spezifikationsrahmen sollte über eine *modulare Struktur* verfügen, so dass zusätzliche Vorgaben bezüglich der Spezifikation von Komponenten abwärtskompatibel als neue Module hinzugefügt werden können. Hierdurch wird zunächst zur *Einheitlichkeit* von Komponentenspezifikationen beigetragen und sichergestellt, dass auch Methoden und Werkzeuge, die auf einer älteren Version des Spezifikationsrahmens aufbauen, solche Komponentenspezifikationen verwenden können, die gemäß den Vorgaben einer neueren Version erstellt wurden. Gleichzeitig zielt diese Anforderung auf eine Steigerung der *Flexibilität des Spezifikationsrahmens*, so dass sich zusätzliche Vorgaben bei Bedarf mit möglichst geringem Aufwand integrieren lassen. Ein solcher Erweiterungsmechanismus ist gerade deswegen zu implementieren, da

die Vollständigkeit einer Komponentenspezifikation meist nicht absolut, sondern nur relativ zu den jeweils unterstützten Aufgaben garantiert werden kann. Somit ist es möglich, dass sich mit der Entdeckung und Erforschung neuer Aufgaben ggf. auch ein Änderungsbedarf am Spezifikationsrahmen ergibt.

- **Praktikabilität.** Bei der Spezifikation von Komponenten sollten möglichst *etablierte*, *industriefähige Notationen* zum Einsatz kommen. Zwar lässt sich durch die Verwendung derartiger Notationen meist nur ein suboptimales Ergebnis im Vergleich zum jeweiligen Stand der Wissenschaft erzielen. Dessen ungeachtet ist die Aufstellung dieser praktischen Anforderung jedoch eine wichtige Maßnahme zur Steigerung der *Akzeptanz des Spezifikationsrahmens* in der (betrieblichen) Anwendungsentwicklung, die auch eine Voraussetzung für die angestrebte Unterstützung des Spezifikationsrahmens durch Werkzeuge aus der Software-Industrie darstellt.

Die Erfüllung dieser grundlegenden Anforderungen durch den eingesetzten Spezifikationsrahmen ist eine wichtige Voraussetzung dafür, dass sich Komponentenspezifikationen erstellen lassen, die über die zu Beginn genannten Eigenschaften verfügen. Von den existierenden Vorschlägen zur Spezifikation von Komponenten werden sie jedoch nicht in ausreichendem Maße berücksichtigt. Vielmehr weisen die meisten der bislang entwickelten Spezifikationsrahmen bezüglich der aufgestellten Anforderungen signifikante Defizite auf, weshalb deren Eignung für eine durchgängige Unterstützung des komponentenorientierten Entwicklungsprozesses in Frage zu stellen ist.

2.3.2 Existierende Ansätze und Defizite

Wegen ihrer zentralen Bedeutung für die komponentenorientierte Entwicklung wurde die Spezifikation von Komponenten bereits vielfach untersucht und durch Vorschläge konkretisiert. Die verschiedenen Vorschläge lassen sich im Hinblick auf ihre Allgemeingültigkeit zunächst in solche einteilen, die die Schaffung eines möglichst aufgabenübergreifend einsetzbaren, generischen Spezifikationsrahmens zum Gegenstand haben, und aufgabenspezifische Empfehlungen, deren Vorgaben bezüglich der Spezifikation von Komponenten lediglich der Bewältigung einer bestimmten Aufgabe dienen.

2.3.2.1 Aufgabenspezifische Ansätze

Zu den *aufgabenspezifischen Spezifikationsvorschlägen* zählen bspw. die in [NIERSTRASZ 1993; YELLIN und STROM 1997:295-298] entwickelten Vorgaben zur Spezifikation von Interaktionsprotokollen, die der Prüfung der Protokollkompatibilität sowie der anschlie-

ßenden Generierung von Protokolladaptern dienen. Daneben zählen das in [VITHARANA, et al. 2003:651-653] zur Katalogisierung und Suche von Komponenten vorgeschlagene Dokumentationsschema und die zur Vorhersage der Funktionalität von Anwendungssystemen in [LAU und ORNAGHI 2001] vorgeschlagene formale Notation, mit der die in Komponentenschnittstellen enthaltenen Signaturen spezifiziert werden können, ebenso zu den aufgabenspezifischen Vorschlägen wie die in [WALLNAU 2003:33-42] zur Vorhersage der Qualität von Anwendungssystemen genannten nicht-funktionalen Merkmale von Komponenten und zahlreiche weitere Vorschläge, die an dieser Stelle nicht betrachtet werden.

Die zur Bewältigung konkreter Aufgaben entwickelten Spezifikationsvorschläge sind für die angestrebte durchgängige Unterstützung des komponentenorientierten Entwicklungsprozesses in der Regel nicht geeignet. Zum einen verstoßen diese Vorschläge vor allem gegen die geforderte *Vollständigkeit*, da mit ihnen nur der jeweils für die Aufgabenstellung relevante Teil einer Komponente beschrieben werden kann. Zum anderen lässt sich auch durch die Kombination mehrerer aufgabenspezifischer Vorschläge üblicherweise kein geeigneter, d.h. die eingangs aufgestellten Anforderungen erfüllender Spezifikationsrahmen schaffen. Da sich die einzelnen Vorschläge in Bezug auf ihren jeweiligen Inhalt vielfach überlappen, jedoch zur Spezifikation meist unterschiedliche Konzepte verwenden, wären bei einer solchen Vorgehensweise zahlreiche Komponenteneigenschaften wiederholt zu beschreiben. Der durch eine Kombination entstehende Spezifikationsrahmen würde deshalb die geforderte *Abgestimmtheit* nicht gewährleisten. Allerdings können aus den zahlreichen aufgabenspezifischen Vorschlägen und ihren jeweils verbindlichen Vorgaben wichtige Hinweise für die Entwicklung bzw. Bewertung eines aufgabenübergreifend einsetzbaren Spezifikationsrahmens gewonnen werden.

2.3.2.2 Ansätze mit UML

Für eine solche aufgabenübergreifende Verwendung wurden vor allem die Spezifikationsrahmen konzipiert, die mit der *SELECT Perspective* [ALLEN und FROST 1998:177-207], der *Catalysis* [D'SOUZA und WILLS 1999:581-615] und der *UML Components* [CHEESMAN und DANIELS 2001:59-63, 121-145] Methodik bereitgestellt werden.[7] Diese Spezifikationsrahmen nutzen die von der *Unified Modeling Language* (UML) in der Version 1.3 [OMG 1999] bereitgestellten Notationen und stellen für die Beschreibung der Außensicht von Komponenten jeweils eine Menge angepasster Modellierungskonstrukte bereit. Dabei er-

[7] Weitere, jedoch weniger verbreitete Vorschläge zur Spezifikation von Komponenten mit UML finden sich bei [STEVENS und POOLEY 2000; HOUSTON und NORRIS 2001]. Einen einfachen Prozess für die Spezifikation von Komponenten gemäß der Catalysis Methodik beschreibt [BROWN 2000:245-266].

folgt die Anpassung der UML Modellierungskonstrukte für die Beschreibung von Komponenten hauptsächlich durch die Einführung spezieller *Namenskonventionen* und *Stereotypen* [CRNKOVIC 2002:132] (zur Erläuterung vgl. [CHEESMAN und DANIELS 2001:39]).

Gemäß dem Ansatz der SELECT Perspective Methodik sind Komponenten als *Pakete* zu modellieren, mit denen sich jeweils mehrere *Services* als konstituierende Elemente zu einer Einheit gruppieren und bestehende *Abhängigkeiten* zu den Services anderer Komponenten beschreiben lassen [ALLEN und FROST 1998:179-181]. Ein Service umfasst typischerweise mehrere zusammengehörige *Funktionen* und ist durch ein stereotypisiertes Klassenkonstrukt darzustellen. Somit repräsentiert jeder Service eine *Schnittstelle* der zu beschreibenden Komponente [ALLEN und FROST 1998:181]. Durch die Beschreibung bestehender Abhängigkeiten zu den Services anderer Komponenten lässt sich darüber hinaus der Bedingungsrahmen spezifizieren, in dem eine Komponente ihre Dienste jeweils anbietet. Jedoch wird in Zusammenhang mit der Repräsentation von Komponentenschnittstellen durch angepasste Klassenkonstrukte explizit auch eine Empfehlung für eine objektorientierte Realisierung der Komponentenspezifikation gegeben [ALLEN und FROST 1998:182]. Damit verstößt der Spezifikationsrahmen nicht nur gegen das aufgestellte Gebot der *Universalität*, sondern verletzt zudem die *Definition von Komponentenspezifikationen*, die prinzipiell eine von der Realisierung unabhängige Form verlangt (vgl. Definition 2.2). Obgleich für den mit der SELECT Perspective Methodik vorgeschlagenen Ansatz ansonsten dieselben Vor- und Nachteile gelten wie für die anderen nachfolgend betrachteten beiden Ansätze, ist er schon aus diesem Grund nicht in der Lage, die eingangs aufgestellten Anforderungen zu erfüllen.

Im Gegensatz dazu zielen sowohl die Catalysis als auch die UML Components Methodik mit ihren Spezifikationsrahmen auf eine von der Realisierung unabhängige Beschreibung der Außensicht von Komponenten [D'SOUZA und WILLS 1999:36; CHEESMAN und DANIELS 2001:22]. Bei beiden Ansätzen steht die Beschreibung der Schnittstellen im Mittelpunkt, die eine Komponente für den Aufruf ihrer Dienste nach außen zur Verfügung stellt (sog. Angebotsschnittstellen) bzw. für den Aufruf von Diensten anderer Komponenten verwendet (sog. Nachfrageschnittstellen) [D'SOUZA und WILLS 1999:387; CHEESMAN und DANIELS 2001:59f.]. Dabei dient die Beschreibung der Nachfrageschnittstellen vor allem der Spezifikation des Bedingungsrahmens, in dem eine Komponente ihre Dienste jeweils zur Verfügung stellt. Die Beschreibung der Komponentenschnittstellen basiert auf dem Konzept des *vertragsbasierten Entwurfs* (Design by Contract [MEYER 1992; MEYER 1997:331-406]), und umfasst somit nicht nur die Signaturen der aufgeführten Dienste, sondern auch die für ihren Aufruf jeweils relevanten Bedingungen, die in Form von *Zusiche-*

rungen beschrieben werden und vom Nachfrager bzw. Anbieter entsprechend zu erfüllen sind.

Der mit der Catalysis Methodik bereitgestellte Spezifikationsrahmen nutzt für die Repräsentation von Komponentenschnittstellen sog. *Typmodelle* (vgl. Abb. 2.8 links), die gemäß dem Konzept des vertragsbasierten Entwurfs um Vor- und Nachbedingungen für die einzelnen Dienste ergänzt werden [D'SOUZA und WILLS 1999:203, 597; BROWN 2000:262-265]. Ergänzend zu den Vor- und Nachbedingungen kann zur Beschreibung von Aufrufreihenfolgen ein Zustandsdiagramm verwendet werden [D'SOUZA und WILLS 1999:136f.]. Durch das mit der Catalysis Methodik eingeführte Typmodell wird das auf dem Klassenkonstrukt basierende UML Typkonstrukt so erweitert, dass eine integrierte Darstellung der an der Schnittstelle ausgetauschten Daten als graphisches Modell ermöglicht wird [D'SOUZA und WILLS 1999:75f.]. Demgegenüber lassen sich Komponentenschnittstellen mit dem von der UML Components Methodik bereitgestellten Spezifikationsrahmen als stereotypisierte UML *Typen* darstellen (vgl. Abb. 2.8 rechts), wodurch eine proprietäre Erweiterung des UML Standards vermieden wird [CHEESMAN und DANIELS 2001:52f.].

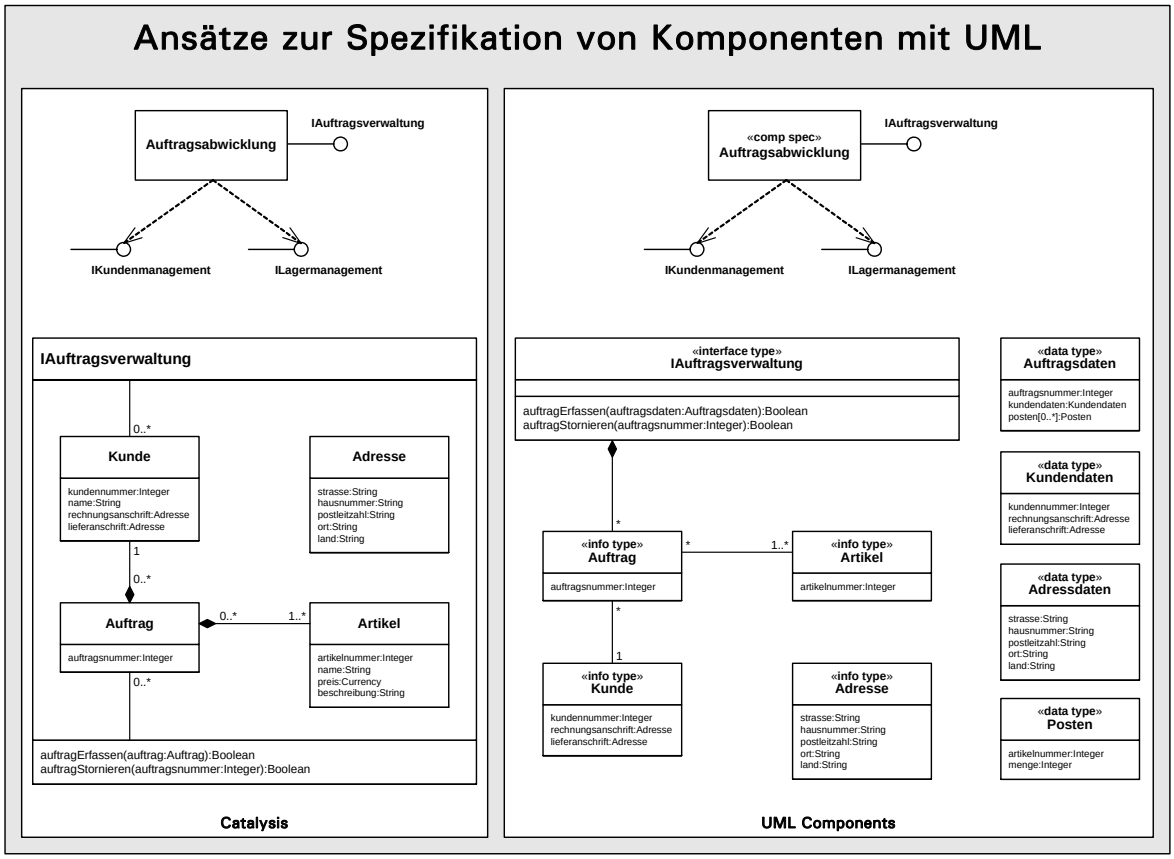


Abb. 2.8: Gegenüberstellung der Modellierungskonstrukte von Catalysis und UML Components (in Anlehnung an [D'SOUZA und WILLS 1999:202f.; CHEESMAN und DANIELS 2001:55-57]).

Die Beschreibung einer Schnittstelle wird dabei ebenfalls um Zusicherungen für die einzelnen Dienste ergänzt [CHEESMAN und DANIELS 2001:55]. Zur graphischen Darstellung der ausgetauschten Daten wird zudem ein spezielles *Informationsmodell* verwendet, mit dem die Beschreibung jeder Schnittstelle vervollständigt wird [CHEESMAN und DANIELS 2001:55, 57]. Zusätzlich sind die einzelnen modellierten Schnittstellen gemäß dem Spezifikationsrahmen der UML Components Methodik zu einer *Komponentenspezifikation* zusammenzufassen. Dies geschieht unter Verwendung eines weiteren Konstrukts, mit dem die einzelnen Schnittstellen als Angebots- bzw. Nachfrageschnittstellen zu klassifizieren sind [CHEESMAN und DANIELS 2001:59-61]. Dieses Konstrukt kann bei Bedarf um *Zusicherungen* ergänzt werden, die zwischen den einzelnen Schnittstellen gelten und so zur Widerspruchsfreiheit der einzelnen Spezifikationsteile beitragen [CHEESMAN und DANIELS 2001:141f.].

Mit den inhaltlichen Vorgaben und den vorgenommenen Erweiterungen der UML Notationen unterstützen die beiden Spezifikationsrahmen eine von der Realisierung unabhängige Beschreibung der Dienste, die an den Schnittstellen von Komponenten angeboten bzw. nachgefragt werden. Sie erfüllen somit die Forderungen nach *Verbindlichkeit* und *Universalität*. Gleichzeitig stellt die zur Spezifikation eingesetzte UML eine in der (betrieblichen) Anwendungsentwicklung bewährte und häufig genutzte Notation dar, weswegen die Spezifikationsrahmen auch den eingangs aufgestellten Anforderungen bezüglich der *Praktikabilität* genügen. Als semiformale Spezifikationssprache erfüllt die zur Beschreibung verwendete UML zudem das Kriterium der *Verständlichkeit*. So stellt sie aufgrund ihrer weiten Verbreitung für die Entwickler einerseits eine Standardsprache dar, deren graphische Konstrukte effizient genutzt und ausgewertet werden können. Andererseits lässt sich die UML hinreichend formalisieren, um eine Auswertung der erstellten Spezifikationen durch Werkzeuge zu ermöglichen – etwa zur Generierung von Teilen der Implementierung. Die standardisierten Notationen der UML garantieren dabei eine Anwendbarkeit zur Unterstützung zahlreicher unterschiedlicher Aufgaben, so dass der geforderten *Angemessenheit* von Komponentenspezifikationen Rechnung getragen wird. Allerdings ist anzumerken, dass eine Vielzahl der UML Erweiterungen, die von den genannten Spezifikationsrahmen vorgenommen wurden, mit Einführung der UML 2.0 [OMG 2003b] überflüssig geworden ist. Diese unterstützt die Spezifikation von Komponenten inzwischen durch standardisierte Konstrukte, die mittelfristig auch von den genannten Spezifikationsrahmen zu nutzen sind.

Darüber hinaus sind die zuvor beschriebenen, auf der UML basierenden Spezifikationsrahmen schwerpunktmäßig auf die Unterstützung der Komponentenentwicklung ausgelegt [D'SOUZA und WILLS 1999:583; CHEESMAN und DANIELS 2001:20, 24]. Sie vernachlässigen infolgedessen insbesondere die Anforderungen, die sich aus dem geplanten Einsatz der

erstellten Spezifikationen während der Anwendungsentwicklung ergeben. Zwar wird in den Ausführungen zu den Spezifikationsrahmen der Catalysis und der UML Components Methodik auch die Auswahl und Begutachtung von Komponenten während der Anwendungsentwicklung als ein mögliches Einsatzgebiet für Spezifikationen erwähnt [D'SOUZA und WILLS 1999:233, 583; CHEESMAN und DANIELS 2001:23]. Beide lassen jedoch die Spezifikation der Komponentenfunktionalität aus fachlich-konzeptioneller Sicht, die für eine Unterstützung dieser Aufgaben erforderlich ist [BROWN 2000:104f.; VITHARANA, et al. 2003:652f.], unberücksichtigt. Auch eine Beschreibung der nicht-funktionalen (Qualitäts-) Eigenschaften von Komponenten ist mit den zuvor genannten Spezifikationsrahmen nicht möglich. Vielmehr gestatten diese ausschließlich die Beschreibung funktionaler Komponenteneigenschaften aus einer logischen Perspektive, die von der eingesetzten UML effektiv unterstützt wird. Somit ergibt sich in Bezug auf die geforderte *Vollständigkeit* eine Vielzahl von Defiziten.

Abschließend ist zu kritisieren, dass keiner der genannten Spezifikationsrahmen eine explizite Einordnung der zu spezifizierenden Sachverhalte in Form eines Klassifikations- oder Metaschemas vornimmt. Deshalb sind die Zusammenhänge zwischen den einzelnen Spezifikationsteilen häufig unklar und nur schwer zu einem abgestimmten Gesamtbild zu integrieren. Darüber hinaus wird vielfach zugelassen, dass ein Sachverhalt in mehreren Spezifikationsteilen beschrieben werden kann (vgl. z.B. die vorhandenen Möglichkeiten zur Spezifikation von Aufrufreihenfolgen durch Vor- und Nachbedingungen bzw. Zustandsautomaten [D'SOUZA und WILLS 1999:136f.; CHEESMAN und DANIELS 2001:42]). In Abhängigkeit von der Sorgfalt, mit der bei der Spezifikation vorgegangen wird, können auf diese Weise redundante und ggf. sogar widersprüchliche Ergebnisse entstehen. Somit wird die Forderung nach *Abgestimmtheit* von den genannten Spezifikationsrahmen nicht erfüllt. Aufgrund der unklaren Zusammenhänge ist es außerdem nur schwer möglich, Teile der jeweiligen Spezifikationsrahmen zu ändern bzw. neue Vorgaben zu integrieren. Das Kriterium der *Veränderbarkeit* wird von ihnen daher ebenfalls nicht ausreichend berücksichtigt.

2.3.2.3 Ansätze mit Architekturbeschreibungssprachen

Im Gegensatz zu den Spezifikationsrahmen, die auf der UML basieren, steht bei *Architekturbeschreibungssprachen* (engl. Architecture Description Languages, ADL) vor allem die Unterstützung der Komposition von Komponenten während der Anwendungsentwicklung im Mittelpunkt. Mit ihnen lassen sich Komponenten, Konnektoren und die sich aus der Verbindung von Komponenten und Konnektoren ergebenden Kompositionen spezifizieren [SHAW und GARLAN 1996:196-198; MEDVIDOVIC und TAYLOR 1997; CLEMENTS, et al. 2003:145]. Dabei sind vor allem die Signaturen der Dienste, die von Komponenten an ihren Schnittstellen bereitgestellt bzw. nachgefragt werden, sowie Protokolle für die Interak-

tionen zwischen einzelnen Komponenten zu beschreiben. Zur Spezifikation wird typischerweise eine textbasierte *formale Notation* verwendet, deren Syntax und Semantik exakt definiert ist [CLEMENTS, et al. 2003:147]. Durch dieses formale Konzept werden die Bestandteile einer Komponentenspezifikation präzise festgelegt, weshalb Architekturbeschreibungssprachen das Kriterium der *Verbindlichkeit* erfüllen. Darüber hinaus werden sämtliche Elemente einer Architekturbeschreibung in ein klar definiertes Verhältnis zueinander gesetzt, sodass auch der Forderung nach *Abgestimmtheit* entsprochen wird. Schließlich ermöglicht das abstrakte formale Konzept eine von konkreten Implementierungstechnologien unabhängige Beschreibung der Anwendungssystemarchitektur. Damit genügen Architekturbeschreibungssprachen außerdem der Anforderung nach *Universalität*.

Durch den formalen Charakter der Spezifikationssprache ergeben sich jedoch auch einige Defizite, z.B. im Hinblick auf das Kriterium der *Verständlichkeit*. So sind die erstellten Komponentenspezifikationen zwar effizient durch Werkzeuge auszuwerten [CLEMENTS, et al. 2003:145]. Für den Entwickler gestaltet sich diese Aufgabe indes den Ausführungen in [HALL 1990] zum Trotz meist deutlich schwieriger. Nicht zuletzt deshalb haben Architekturbeschreibungssprachen in der Praxis der (betrieblichen) Anwendungsentwicklung bislang keine herausragende Bedeutung erlangt, so dass eine Empfehlung zugunsten der Architekturbeschreibungssprachen außerdem der geforderten *Praktikabilität* widersprechen würde. Gleichzeitig wird aufgrund der komplexen formalen Konzepte gerade die *Veränderbarkeit* von Architekturbeschreibungssprachen erschwert, da meist nur ein kleiner Personenkreis über die dazu notwendigen Fähigkeiten verfügt. Schließlich ist anzumerken, dass auch Architekturbeschreibungssprachen meist nur die Beschreibung funktionaler Komponenteneigenschaften aus einer logischen Perspektive unterstützen. Qualitative Eigenschaften lassen sich hingegen in der Regel ebenso wenig beschreiben wie die Funktionalität von Komponenten aus fachlich-konzeptioneller Sicht.[8] Aus diesem Grund genügen Architekturbeschreibungssprachen ebenso wie die anderen genannten Ansätze nicht dem Kriterium der *Vollständigkeit*. Da sie überdies nur auf eine Unterstützung der Anwendungsentwicklung ausgelegt sind, ist in Zusammenhang mit der Komponentenentwicklung auch ihre *Angemessenheit* zumindest in Frage zu stellen.

2.3.2.4 Vertragsebenenorientierte Ansätze

Auf eine möglichst vollständige Beschreibung der Außensicht von Komponenten, die sich sowohl während der Anwendungs- als auch der Komponentenentwicklung verwenden

[8] Eine Ausnahme stellt die Architekturbeschreibungssprache Rapide dar, mit der sich zumindest die zeitliche Abstimmung (das sog. Timing) von Komponenten beschreiben lässt [LUCKHAM, et al. 1995].

lässt, zielen vor allem diejenigen Ansätze, die auf dem Konzept des vertragsbasierten Entwurfs aufsetzen und verschiedene, für die Beschreibung von Komponenteneigenschaften relevante *Vertragsebenen* identifizieren. So werden in der Literatur verschiedene inhaltliche Aspekte, die es bei der Erstellung von Software-Verträgen für Komponenten zu berücksichtigen gilt, genannt [SZYPERSKI 1998:45-47; SZYPERSKI, et al. 2002:55-57]. Aus diesen lässt sich ein allgemeines Schema ableiten, mit dem der Inhalt von Software-Verträgen für Komponenten klassifiziert werden kann [HAN 1998; BEUGNARD, et al. 1999:38-40]. Das entstehende Vertragsschema unterscheidet zunächst vier Vertragsebenen, die als *Syntaxebene* (Syntactic Level), *Verhaltensebene* (Behavioral Level), *Abstimmungsebene* (Synchronization Level) und *Qualitätsebene* (Quality-of-Service Level) bezeichnet werden (vgl. Abb. 2.9 links).

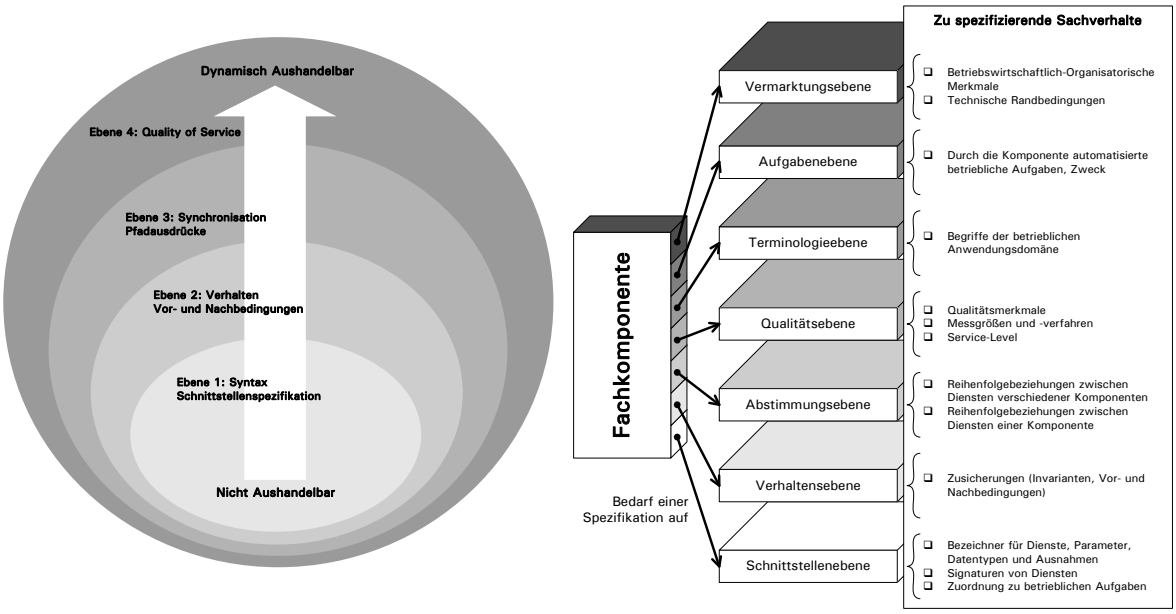


Abb. 2.9: Vertragsebenen zur Spezifikation von Komponenten (in Anlehnung an [BEUGNARD, et al. 1999:39; ACKERMANN, et al. 2002:4]).

Während auf der Syntax- und der Verhaltensebene ähnlich wie bei den zuvor genannten Ansätzen die jeweiligen *Signaturen* bzw. die *Vor- und Nachbedingungen* der Dienste beschrieben werden, die Bestandteil der Komponentenschnittstellen sind [BEUGNARD, et al. 1999:38-40], beinhalten die letzten beiden Vertragsebenen zusätzliche Informationen. So sind auf der Abstimmungsebene zunächst die zwischen den einzelnen Diensten existierenden Abhängigkeiten zu beschreiben, aus denen sich die zulässigen Aufrufreihenfolgen ableiten lassen. Auf der Qualitätsebene werden schließlich die nicht-funktionalen Eigenschaften der Dienste spezifiziert [BEUGNARD, et al. 1999:40]. Mit diesen Vertragsebenen lassen sich wichtige zusätzliche Anforderungen formulieren, die bei der Komponentenentwicklung zu berücksichtigen sind. Darüber hinaus tragen gerade diese zusätzlichen Informatio-

nen zu einer effizienteren Überprüfung der Eignung von Komponenten im Rahmen von Anwendungsentwicklungsprojekten bei.

Das vorgeschlagene Vertragsschema wurde in verschiedenen Arbeiten aufgegriffen und, vor allem in Zusammenarbeit mit einem Arbeitskreis der Gesellschaft für Informatik (GI), zu einem Memorandum für die Spezifikation von Komponenten des betrieblichen Anwendungsbereichs – sog. *Fachkomponenten* – weiterentwickelt [CONRAD und TUROWSKI 2000; TUROWSKI 2001; ACKERMANN, et al. 2002]. Dabei wurden eine *Terminologieebene* sowie eine *Aufgabenebene* zur Beschreibung der Komponentenfunktionalität aus fachlich-konzeptioneller Sicht als weitere Vertragsebenen hinzugefügt [ACKERMANN, et al. 2002:3f.]. Aufbauend auf dem so erweiterten Vertragsschema (vgl. Abb. 2.9 rechts) wurde ein Spezifikationsrahmen für die Beschreibung von Fachkomponenten entwickelt, der konkrete inhaltliche Vorgaben für die auf den einzelnen Ebenen zu beschreibenden Komponenteneigenschaften beinhaltet und gleichzeitig eine Reihe von *textbasierten Notationen* zur Beschreibung festlegt [ACKERMANN, et al. 2002:5-23].

Der mit dem Memorandum vorgeschlagene Spezifikationsrahmen nutzt zur Beschreibung von Sachverhalten hinreichend präzise, vorzugsweise bewährte und industriefähige Notationen. Er genügt damit vor allem den Kriterien der *Praktikabilität* und *Verständlichkeit*. Darüber hinaus gewährleistet der ebenenorientierte Aufbau eine vergleichsweise leichte *Veränderbarkeit*. Gegenüber den vorgenannten Ansätzen hebt sich der Spezifikationsrahmen weiterhin dadurch ab, dass er eine umfassendere Beschreibung der Außensicht von Komponenten anstrebt und dabei auch viele derjenigen Komponenteneigenschaften einbezieht, die während der Anwendungsentwicklung nachgefragt werden. So unterstützt er die Beschreibung der Funktionalität von Komponenten aus fachlich-konzeptioneller Sicht und die Spezifikation nicht-funktionaler (Qualitäts-) Eigenschaften gleichermaßen. Damit entspricht er der Forderung nach *Vollständigkeit* eher als die bislang genannten Ansätze.

Obwohl das Streben nach Vollständigkeit eines der obersten Ziele bei der Schaffung dieses Spezifikationsrahmens war [ACKERMANN, et al. 2002:3], liegt der Identifikation und Beschreibung neuer Vertragsebenen jedoch nur ein empirisches Vorgehen zugrunde, mit dem sich das erreichte Ergebnis nicht in intersubjektiv nachvollziehbarer Weise begründen lässt. Dabei wurden wiederholt neue Ebenen identifiziert und bereits vorhandene Ebenen diversifiziert bzw. eliminiert (vgl. hierzu die in [CONRAD und TUROWSKI 2000:180; TUROWSKI 2001:278; ACKERMANN, et al. 2002:4] dargestellten Vertragsebenen). Gleichzeitig erfüllt der Spezifikationsrahmen auch das Gebot der *Verbindlichkeit* nicht vollständig, da der zu beschreibende Inhalt und die einzusetzenden Notationen nicht für alle Vertragsebenen festgelegt wurden. Vor allem für die Beschreibung der nicht-funktionalen Komponen-

teneigenschaften auf der Qualitätsebene wird lediglich ein allgemeines Vorgehen beschrieben, das weder Festlegungen bezüglich der zu beschreibenden Eigenschaften, noch im Hinblick auf die zur Spezifikation einzusetzende Notation beinhaltet [ACKERMANN, et al. 2002:14f.]. Insofern lässt der Spezifikationsrahmen dem Entwickler bei der Beschreibung nicht-funktionaler Eigenschaften zahlreiche unerwünschte Freiräume.

Ferner ist vor allem zu kritisieren, dass auch bei der Entwicklung dieses Spezifikationsrahmens keine explizite Einordnung der zu spezifizierenden Sachverhalte durch ein Klassifikations- oder Metaschema vorgenommen wurde. Deshalb wird das eingangs aufgestellte Gebot der *Abgestimmtheit* ebenfalls nicht erfüllt. Mit der Einführung zusätzlicher Vertragsebenen wurden die Möglichkeiten, denselben Sachverhalt auf verschiedenen Ebenen mehrfach und dabei ggf. widersprüchlich zu beschreiben, indes sogar noch verstärkt. So können Interaktionsprotokolle gemäß dem Spezifikationsrahmen sowohl auf der Verhaltensebene (durch die Vorgabe von Vor- und Nachbedingungen) als auch der Abstimmungsebene (durch die Vorgabe von Aufrufreihenfolgen) beschrieben werden [CHEESMAN und DANIELS 2001:42]. Hinter der Kritik an der fehlenden Einordnung einzelner Spezifikationsteile tritt die Beobachtung zurück, dass der Spezifikationsrahmen wegen seiner Spezialisierung auf die Beschreibung von Fachkomponenten auch die Forderung nach *Universalität* nicht erfüllt. Dieses Defizit lässt sich durch einige geringfügige Veränderungen beheben (vgl. [OVERHAGE 2002:11f.]).

Zu erwähnen bleibt allerdings, dass der Spezifikationsrahmen des Memorandums weder das grundlegende Konzept der *Mehrfachschnittstellen* von Komponenten noch eine durchgängige Beschreibung der existierenden Abhängigkeiten zu den Diensten anderer Komponenten unterstützt. Daher lässt sich der *Bedingungsrahmen*, in dem eine Komponente ihre Dienste anbietet, nicht in einer Weise beschreiben, die mit den anderen Ansätzen vergleichbar ist. Stattdessen wurde das Konzept der Inter-Komponenten-Abstimmung eingeführt, mit dem Einschränkungen für die Komposition von Komponenten beschrieben werden können. Durch die Vorgabe von Einschränkungen wird jedoch der Freiheitsgrad bei der Verbindung von Komponenten zu Gesamtsystemen und damit die gewünschte Kombinierbarkeit von Komponenten in unvorhergesehenen Szenarien herabgesetzt. Schon deshalb besitzt der Spezifikationsrahmen nur eine eingeschränkte *Angemessenheit*.

2.3.2.5 Zusammenfassung

Insgesamt ist festzuhalten, dass keiner der genannten Spezifikationsrahmen in der Lage ist, sämtliche der eingangs aufgestellten grundlegenden Anforderungen optimal zu erfüllen (vgl. Abb. 2.10). Vielmehr besitzen die einzelnen Ansätze unterschiedliche Stärken und Schwächen, die sich hauptsächlich aus ihren jeweils voneinander abweichenden *inhaltli-*

chen Vorgaben zur Spezifikation von Komponenten ergeben. Gleichzeitig sind bei den betrachteten Spezifikationsrahmen jedoch auch signifikante Unterschiede im Hinblick auf die zur Beschreibung vorgeschlagenen Notationen festzustellen. Dabei kann allgemein zwischen Ansätzen, die *textbasierte Notationen* verwenden, und solchen, die *graphische Diagrammsprachen* bevorzugen, unterschieden werden.

	Aufgabenspez Ansätze	SELECT Perspective	Catalysis	UML Components	Architektur-beschreibung	Vertrags-ebenen	Memorandum
Angemessenheit	+	+	+	+	o	o	o
Vollständigkeit	-	-	-	-	-	o	o
Abgestimmtheit	-	-	-	-	+	-	-
Universalität	o	-	+	+	+	+	-
Verbindlichkeit	+	o	o	o	+	-	o
Verständlichkeit	o	+	+	+	-	o	+
Veränderbarkeit	-	-	-	-	-	+	+
Praktikabilität	o	+	+	+	-	o	+

Legende: + erfüllt, o berücksichtigt, - nicht berücksichtigt

Abb. 2.10: Unterstützung der Anforderungen durch die verschiedenen Spezifikationsrahmen.

Mit ihren jeweils abweichenden Vorgaben zur Repräsentation unterscheiden sich die einzelnen Spezifikationsrahmen also nicht nur bezüglich der zu beschreibenden Komponenteneigenschaften, sondern auch im Hinblick auf ihre Eignung für die Benutzung durch verschiedene Zielgruppen während des Entwicklungsprozesses. So wird eine graphische Darstellung von Komponentenspezifikationen im Allgemeinen von den am Entwurf beteiligten Analytikern bevorzugt, während textbasierte Repräsentationen Vorteile bei der Realisierung von Komponenten sowie der Verwendung von Komponentenspezifikationen in Entwicklungswerkzeugen besitzen.

2.3.3 Lösungskonzept

Zur Schaffung eines universell einsetzbaren Spezifikationsrahmens, der die grundlegenden Anforderungen vollständig erfüllt, bedarf es eines Ansatzes, der die spezifischen Stärken der vorgestellten Ansätze zusammenführt und die verbleibenden Defizite behebt. Hierzu werden die einzelnen Vorschläge zur Spezifikation von Komponenten im Folgenden von einem *vereinheitlichenden Ansatz* zusammengeführt. Der in dieser Arbeit dargestellte Spezifikationsrahmen verbindet jedoch nicht nur die bereits bestehenden Vorschläge zu einer – aufgrund der bestehenden Überlappungen wenig konsistenten – Obermenge. Vielmehr

wurde mit seiner Entwicklung die systematische Rekonstruktion eines Spezifikationsrahmens angestrebt, die auf den Ergebnissen der bereits existierenden Vorschläge aufsetzt und diese zu einem Gesamtansatz integriert, dabei jedoch die vorhandenen Ansätze zugleich ersetzen soll. Entsprechend der mit seiner Entwicklung verfolgten Absicht trägt dieser Spezifikationsrahmen den Namen UNIFIED SPECIFICATION OF COMPONENTS (UNSCOM).

Der darzustellende Spezifikationsrahmen basiert auf einem allgemeingültigen konzeptionellen *Komponentenmodell*, das von konkreten Technologien unabhängig ist. Es konkretisiert das komponentenorientierte Entwicklungsparadigma und schafft zugleich ein *einheitliches Begriffsverständnis*. Darüber hinaus werden mit Angebots- und Nachfrageschnittstellen, Komponententypen sowie Dienst- und Komponentenverträgen geeignete Konzepte eingeführt, mit denen sich die *allgemeinen Funktionen* von Komponenten als Strukturierungseinheiten (vgl. hierzu Abschnitt 2.1.1) in das Komponentenmodell abbilden und unterstützen lassen. Damit werden schon bei der Modellbildung die verschiedenen Verwendungen von Komponentenspezifikationen, die sich aus den allgemeinen Funktionen während des Entwicklungsprozesses ergeben, und ihre jeweiligen Anforderungen berücksichtigt. Die Modellbildung trägt so wesentlich zur geforderten *Angemessenheit* des Spezifikationsrahmens bei. Mit der Nutzung eines allgemeingültigen Komponentenmodells wird ferner die *Universalität* des zu schaffenden Spezifikationsrahmens sichergestellt. So stellen das einheitliche Verständnis des komponentenorientierten Entwicklungsparadigmas und die im Rahmen des Modells eingeführten allgemeingültigen Konzepte sicher, dass der darauf aufsetzende UNSCOM Spezifikationsrahmen unabhängig von konkreten Technologien und Implementierungsparadigmen zur Beschreibung der Außensicht von Komponenten eingesetzt werden kann. Zugleich lassen sich mit dem Spezifikationsrahmen verschiedene Komponentenarten beschreiben, die im Rahmen des Komponentenmodells identifiziert und einander gegenübergestellt werden.

Schließlich bildet das Komponentenmodell mit seinen Konzepten auch die analytische Grundlage für die systematische Bestimmung der zu beschreibenden Komponenteneigenschaften. Mit seinen aus der Allgemeinen Systemtheorie abgeleiteten Modellperspektiven, die jeweils verschiedene Beschreibungsaspekte repräsentieren, trägt das Komponentenmodell zur Begründung der *Vollständigkeit* des UNSCOM Spezifikationsrahmens bei. Um eine möglichst umfassende Identifikation von Komponenteneigenschaften zu gewährleisten, werden zusätzlich die verschiedenen Abstraktionsebenen des Entwicklungsprozesses in die Beschreibung einbezogen und den Modellperspektiven gegenübergestellt. Die dabei identifizierten Spezifikationsebenen decken die in den vertragsbasierten Ansätzen jeweils identifizierten Vertragsebenen ab, so dass der UNSCOM Spezifikationsrahmen die zuvor genannten Ansätze hinsichtlich des Inhalts von Komponentenspezifikationen umfasst. Gleichzei-

tig unterstützen die voneinander abgegrenzten Spezifikationsebenen den modularen Aufbau des Spezifikationsrahmens, mit dem auch das Gebot der *Veränderbarkeit* erfüllt wird.

Für die identifizierten Spezifikationsebenen werden im Rahmen eines gemeinsamen, ebenenübergreifenden *Metaschemas* normative inhaltliche Vorgaben geschaffen und zu einem Gesamtzusammenhang integriert. Das Metaschema bildet als eigenständiges *Inhaltsmodell* einen zentralen Bestandteil des UNSCOM Spezifikationsrahmens und stellt im Vergleich zu den bereits existierenden Spezifikationsvorschlägen eine wesentliche Weiterentwicklung dar. Es trägt einerseits zur Erfüllung der Forderungen nach *Abgestimmtheit* und (inhaltlicher) *Verbindlichkeit* bei. Andererseits unterstützt die Einführung des Metaschemas eine getrennte Betrachtung von inhaltlichen Vorgaben und solchen Vorgaben, die sich auf die zur Beschreibung einzusetzenden Notationen beziehen. Durch diese Trennung von Inhalt und Repräsentation eröffnet sich die Möglichkeit, zur Darstellung eines Sachverhalts jeweils mehrere Notationen zuzulassen. Die inhaltliche Konsistenz der verschiedenen Darstellungen bleibt gewährleistet, da die Ausdruckskraft der einzelnen Notationen äquivalent und dabei zugleich durch das Metaschema inhaltlich normiert ist.

Bei der Entwicklung des UNSCOM Spezifikationsrahmens wurde eine *Vielzahl von Darstellungsformen* statt eines einzigen Formats unterstützt, wobei zunächst ein textbasiertes, ein auf der UML 2.0 [OMG 2003b] basierendes graphisches sowie ein XML Format entwickelt wurden. Die Einführung der verschiedenen Formate entspricht den unterschiedlichen Präferenzen der am Entwicklungsprozess beteiligten Zielgruppen und damit dem Gebot der *Verständlichkeit*. Zugleich dient die Einführung der verschiedenen Formate dem Ziel, die bereits existierenden Vorschläge zur Spezifikation von Komponenten mit ihren unterschiedlichen Formaten zu umfassen. Dabei wird in Kauf genommen, dass die geforderte *Verbindlichkeit* bezüglich der einzusetzenden Notationen nur eingeschränkt erfüllt wird. Dank der bestehenden inhaltlichen Verbindlichkeit ist jedoch sichergestellt, dass die unterschiedlichen Darstellungen effizient ineinander übersetzt werden können. Abschließend bleibt anzumerken, dass bei der Entwicklung der einzelnen Formate möglichst auf bewährte und industriefähige Notationen zurückgegriffen wurde. Beim Einsatz formaler Notationen wurde zudem stets auf eine Ergänzung durch natürlichsprachliche Kommentare geachtet. Damit berücksichtigt der UNSCOM Spezifikationsrahmen auch die Anforderungen, die sich aus den Geboten der *Verständlichkeit* sowie der *Praktikabilität* ergeben.

Mit dem UNSCOM Spezifikationsrahmen ist es somit möglich, die eingangs gestellten grundlegenden Anforderungen zu erfüllen. Dabei greift der Spezifikationsrahmen an vielen Stellen die Ergebnisse der einzelnen, bereits bestehenden Vorschläge auf und verwendet diese weiter. Er genügt somit den Anforderungen, die an einen *vereinheitlichenden* Ansatz

zu stellen sind. Demgegenüber bleibt vor allem seine durchgängige Verwendbarkeit im Rahmen einer Methodik zur Unterstützung der komponentenorientierten Entwicklung noch in der Praxis zu überprüfen. Aufgrund der Tatsache, dass eine solche Methodik erst im Entstehen begriffen ist, kann hierüber derzeit noch kein abschließendes Urteil gefällt werden. Dessen ungeachtet lassen sich jedoch anhand von Plausibilitätsbetrachtungen bereits einige Hinweise für seine Verwendbarkeit geben. Hierzu werden bei der Darstellung des Spezifikationsprozesses an einigen Stellen konkrete Anwendungen für den UNSCOM Spezifikationsrahmen im Rahmen eines Ausblicks auf die allmählich entstehende Entwicklungsmethodik benannt und näher erläutert.

Die mit dem UNSCOM Spezifikationsrahmen umgesetzte Schaffung von Vorgaben bezüglich des *Inhalts* und der *Repräsentation* ist allerdings nur eine wichtige Maßnahme, um die Spezifikation von Komponenten zu unterstützen. Darüber hinausgehend sind vor allem Prozessschemata zu definieren, mit denen sich die die Anwendung des Spezifikationsrahmens während des Entwicklungsprozesses beschreiben lässt. Ferner sind Methoden und Werkzeuge zu entwickeln, die die Spezifikation der verschiedenen Sachverhalte unterstützen. Die Darstellung des UNSCOM Spezifikationsrahmens wird deshalb um ein allgemeingültiges, ergebnisorientiertes Vorgehensmodell ergänzt, das den Lebenszyklus einer Spezifikation beschreibt. Dabei werden an einigen Stellen Hinweise auf Methoden gegeben, die zur Unterstützung der Spezifikationsarbeit eingesetzt werden können. Wegen der meist geringeren Allgemeingültigkeit solcher Methoden wird auf eine umfassende Ausarbeitung jedoch verzichtet und stattdessen auf einschlägige Literatur verwiesen.

2.4 Fallstudie

Die Ausführungen zum UNSCOM Spezifikationsrahmen nehmen zur Veranschaulichung auf die Ergebnisse einer Fallstudie Bezug, in deren Rahmen ein komponentenorientiertes *Verkaufssystem* entworfen wurde, das die automatisierte Abwicklung der Bestellungen von Endkunden für einen Internetversandhändler realisiert. Das entwickelte betriebliche Anwendungssystem stellt hierfür einerseits sog. Hilfsdienste zur Registrierung von Kunden und zur Abfrage der Lieferbarkeit einzelner Artikel zur Verfügung. Den Schwerpunkt der Funktionalität bilden indes die darüber hinaus bereitgestellten Dienste zur Erfassung von Bestellungen, die aus Sicht des Versandhändlers Abnehmer- bzw. *Kundenaufträge* darstellen [BECKER und SCHÜTTE 2004:396f., 399f.], sowie zu deren anschließender Bearbeitung. Das entworfene Anwendungssystem unterstützt damit den *Distributionsprozess* [BECKER und SCHÜTTE 2004:396], beschränkt sich jedoch auf die Funktionsbereiche *Kundenmanagement*, *Artikelmanagement*, *Verkauf*, *Warenausgang* und *Fakturierung*. Es deckt folglich

nicht den gesamten Prozess ab, zu dem neben den genannten Funktionsbereichen auch die Sortiment- und Preisgestaltung, die zusammen mit dem Kundenmanagement Bestandteile des (Absatz-) *Marketing* sind, sowie die *Debitorenbuchhaltung* gehören [BECKER und SCHÜTTE 2004:396f.].

Das Kundenmanagement umfasst schwerpunktmäßig die Verwaltung der *Kundenstammdaten* einschließlich der Neuerfassung eines Kunden, der Aktualisierung seiner Daten und der Löschung der erfassten Kundendaten bei Beendigung einer Geschäftsbeziehung. Zu den Kundendaten zählen dabei sog. Grunddaten, spezielle Marketing- und Verkaufsdaten sowie die für die Durchführung der (Debitoren-) Buchhaltung erforderlichen Informationen [BECKER und SCHÜTTE 2004:399f., 413f.]. In analoger Weise fokussiert das Artikelmanagement auf die Verwaltung von *Artikelstammdaten*, die die Neuerfassung von Artikeln nach ihrer Aufnahme in das Sortiment, die Aktualisierung der Daten von Artikeln sowie die Löschung von Artikeln aus dem Sortiment umfasst. Zu den verwalteten Artikeldaten zählen im Wesentlichen sog. Grunddaten, spezielle Marketing- sowie die für die Beschaffung, Lagerhaltung und Distribution erforderlichen Informationen [BECKER und SCHÜTTE 2004:253-257].

Der Verkauf fasst alle (betrieblichen) Aktivitäten zusammen, die zur Durchführung eines Umsatzvorgangs mit Endkunden erforderlich sind. Dabei steht vor allem die Abwicklung von *Kundenaufträgen* im Mittelpunkt, die die Annahme von Aufträgen, deren anschließende Bearbeitung, die Stornierung bereits in Bearbeitung befindlicher Aufträge sowie die Bearbeitung von Reklamationen und Retouren umfasst [BECKER und SCHÜTTE 2004:431]. Für die Bearbeitung von Kundenaufträgen stellen vor allem der Warenausgang und die Fakturierung wichtige Hilfsprozesse dar. Die Aufgabe des Warenausgangs ist dabei die Erfüllung eines Kundenauftrags durch eine *Lieferung*, mit der die gewünschte Menge an Gütern in der vereinbarten Zeit und Qualität an den Endkunden überstellt wird [BECKER und SCHÜTTE 2004:454]. Der Warenausgang umfasst die Kommissionierung, d.h. die Zusammenstellung der bestellten Artikel aus dem Lager sowie die Ausstellung des Lieferscheins [BECKER und SCHÜTTE 2004:454]. Er ist um Dienste zur Abfrage des aktuellen Lagerbestands der angebotenen Artikel sowie zur mengenmäßigen Erfassung von Veränderungen im Lagerbestand zu ergänzen. Die Fakturierung führt schließlich eine Bewertung der jeweils erfolgten Lieferungen durch und erstellt dabei die *Rechnung* für den Endkunden [BECKER und SCHÜTTE 2004:483]. Neben der Rechnungserstellung gehört zu ihren Aufgaben auch die die Einbuchung von offenen Forderungen in die Debitorenbuchhaltung, mit der sich anschließend der Ausgleich von Forderungen durch den Kunden überwachen lässt [BECKER und SCHÜTTE 2004:483].

Es bleibt anzumerken, dass im Rahmen dieser Arbeit kein vollständiger, in der Praxis universal einsetzbarer Anwendungssystementwurf für die angesprochenen Funktionsbereiche durchgeführt werden kann. Dabei würde speziell deren Komplexität, die bspw. durch das SAP R/3 System (vgl. [BECKER, et al. 2000]) umfassend abgebildet wird, dem beabsichtigten Einsatz zu Illustrationszwecken entgegenstehen. Trotz der vorzunehmenden Vereinfachungen eignet sich das im Rahmen der Fallstudie konzipierte Verkaufssystem vor allem aufgrund der Tatsache als Anwendungsbeispiel, dass das Lagergeschäft (also die Beschaffung und Distribution [BECKER und SCHÜTTE 2004:259]) zu den bereits umfassend untersuchten und strukturierten Gegenstandsbereichen der betrieblichen Anwendungsdomäne gehört. Dabei erweist es sich als wichtiger Vorteil, dass der Entwurf des Anwendungssystems an einem Referenzmodell (dem sog. *Handels-H-Modell* [BECKER und SCHÜTTE 2004:42-48]) ausgerichtet werden kann, das die Architektur von Handelsinformationssystemen in umfassender und allgemeingültiger Weise abbildet. Zudem ist davon auszugehen, dass selbst fachfremde Anwendungsentwickler aufgrund ihrer alltäglichen Erfahrungen mit dem gewählten Anwendungsbeispiel zumindest aus der Sicht eines bestellenden Endkunden vertraut sein dürften, was zu seiner allgemeinen Verständlichkeit beiträgt.

2.5 Notationstechnische Konventionen

Vor allem bei der Einführung der verschiedenen Formate des UNSCOM Spezifikationsrahmens in Kapitel 4 wird die Syntax einiger sprachlicher Ausdrücke unter Rückgriff auf die Symbole einer Backus-Naur-Form (BNF) erklärt. Die verwendeten Symbole haben dabei folgende Bedeutung:

Ausdruck1 | Ausdruck2	alternative Ausdrücke
(Ausdruck)	Vorrangsklammerung
[Ausdruck] bzw. (Ausdruck)?	optionaler Ausdruck
(Ausdruck)*	n-fache Ausdruckswiederholung ($n \geq 0$)
(Ausdruck)$^{+}$	n-fache Ausdruckswiederholung ($n \geq 1$)
Symbol	Terminalsymbol
Symbol	zu ersetzendes Symbol (Pseudoterminalsymbol)
/Ausdruck/	Ausdruck mit BNF Symbolen innerhalb eines UML Ausdrucks

3 Konzeptionelle Grundlegung

Bevor mit der Darstellung konkreter Vorgaben hinsichtlich des Inhalts und des Formats von Komponentenspezifikationen begonnen wird, werden in diesem Kapitel die dafür notwendigen Grundlagen gelegt, die zugleich das theoretische Fundament des UNSCOM Spezifikationsrahmens bilden. Hierzu werden in Abschnitt 3.1 im Rahmen eines konzeptionellen Komponentenmodells zunächst allgemein gültige, formale Definitionen für den Komponentenbegriff und die zentralen Konzepte des komponentenorientierten Entwicklungsparadigmas entwickelt. Das mit dem Komponentenmodell geschaffene Verständnis des Komponentenbegriffs und seiner charakteristischen Merkmale unterstützt insbesondere die systematische Bestimmung der zu beschreibenden Komponenteneigenschaften. Gleichzeitig bildet es den Ausgangspunkt für die Entwicklung zusätzlicher Konzepte, mit denen sich die in der Gegenstandsbestimmung genannten allgemeinen Funktionen von Komponenten als Strukturierungseinheiten in das entwickelte Komponentenmodell integrieren lassen.

Mit der Entwicklung dieser zusätzlichen Konzepte wird sichergestellt, dass die allgemeinen Funktionen von Komponenten als Strukturierungseinheiten im Komponentenmodell und dem darauf aufbauenden Spezifikationsrahmen in angemessener Weise berücksichtigt und unterstützt werden. Diese Konzepte bilden deshalb ebenfalls zentrale Grundlagen für den später darzustellenden UNSCOM Spezifikationsrahmen. Neben der Einführung von *Angebots- und Nachfrageschnittstellen* als Bestandteile des Komponentenmodells in Abschnitt 3.1 wird dabei in Abschnitt 3.2 ein *Typsystem* für Komponenten entworfen, das das Komponentenmodell in Bezug auf die statische Analysierbarkeit von Komponenteneigenschaften ergänzt. Darüber hinaus wird in Abschnitt 3.3 das Konzept des *vertragsbasierten Entwurfs* (Design by Contract [MEYER 1992; MEYER 1997:331-406]) erweitert und für die Beschreibung von Komponenten angepasst. Die Bestimmung der für die Entwicklung relevanten und deshalb zu spezifizierenden Komponenteneigenschaften in Abschnitt 3.4 schließt die Betrachtung der Grundlagen ab und leitet zur Darstellung des UNSCOM Spezifikationsrahmens über.

3.1 Komponentenmodell

Für die Darstellung der Vorgaben zur Beschreibung der Außensicht von Komponenten im vierten Kapitel ist vor allem ein geklärtes Verständnis des Komponentenbegriffs, seiner

charakteristischen Merkmale sowie der mit ihm in unmittelbarem Zusammenhang stehenden zentralen Konzepte des komponentenorientierten Entwicklungsparadigmas von grundlegender Bedeutung [CRNKOVIC 2002:129]. Aus diesem Grund sind zunächst Definitionen für die zentralen Konzepte des komponentenorientierten Entwicklungsparadigmas zu entwickeln und zu einem Gesamtbild, einem *konzeptionellen Komponentenmodell* zusammenzufügen. Um eine technologieunabhängige, allgemein gültige Betrachtung der wesentlichen Zusammenhänge zu ermöglichen, abstrahiert dieses Komponentenmodell von derzeit am Markt verfügbaren Komponentenmodellen wie dem OMG CCM [WANG, et al. 2001; OMG 2002; SZYPERSKI, et al. 2002:247-260], Sun EJB [BLEVINS 2001; SZYPERSKI, et al. 2002:284-328] oder Microsoft .NET [SZYPERSKI, et al. 2002:357-380].

Das konzeptionelle Komponentenmodell basiert im Wesentlichen auf dem *Komponenten-Konnektor-Modell*, das zur Beschreibung der logischen Architektur von komponentenorientierten Anwendungssystemen entwickelt wurde [SHAW und GARLAN 1996:196-198; CLEMENTS, et al. 2003:103-123] und nicht zuletzt wegen seiner Verwendung im Rahmen der Catalysis Methodik [D'SOUZA und WILLS 1999:383, 410-414] sowie des UML 2.0 Komponentendiagramms [OMG 2003b:133] eine herausragende Bedeutung als technologisches Referenzmodell der komponentenorientierten Anwendungsentwicklung erlangt hat. Das Komponenten-Konnektor-Modell führt zur Beschreibung der logischen Architektur von komponentenorientierten Anwendungssystemen im Wesentlichen vier Konzepte ein (vgl. Abb. 3.1): *Komponenten*, *Konnektoren, Ports* und *Kompositionen* [SHAW und GARLAN 1996:197].

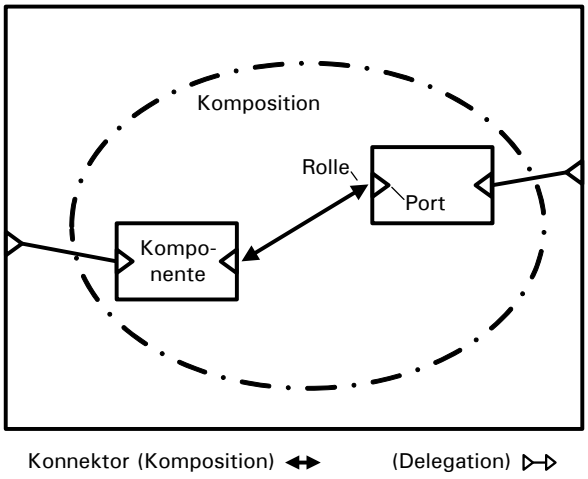


Abb. 3.1: Komponenten-Konnektor-Modell zur Beschreibung der logischen Architektur von komponentenorientierten Anwendungssystemen (in Anlehnung an [SHAW und GARLAN 1996:197]).

Es zeichnet sich außerdem durch ein vergleichsweise allgemeines, abstraktes Begriffsverständnis sowie durch die grundlegende *Trennung von Verarbeitungs- und Interaktionsas-*

pekten als Architekturprinzip aus. Der Verarbeitungsaspekt wird dabei durch die Komponenten repräsentiert, deren Aufgabe die Durchführung einer kooperativen Informationsverarbeitung ist [SHAW und GARLAN 1996:197]. Zu diesem Zweck bieten Komponenten jeweils bestimmte Dienste für die Nutzung durch andere Komponenten an und fragen während der Informationsverarbeitung ggf. auch Dienste von diesen nach. Im Modell werden die von einer Komponente jeweils angebotenen bzw. nachgefragten Dienste zu *Schnittstellen* zusammengefasst und als *Ports* dargestellt [D'SOUZA und WILLS 1999:410; OMG 2003b:138]. Die Ports bilden die Ausgangspunkte der Interaktionen zwischen den Komponenten, die im Modell als *Konnektoren* dargestellt und somit ebenfalls als eigenständige Objekte von Software-Architekturen aufgefasst werden [SHAW und GARLAN 1996:197; D'SOUZA und WILLS 1999:410f.]. Konnektoren repräsentieren den Interaktionsaspekt und verbinden mehrere Komponenten zu einer größeren Ganzheit, die im Komponenten-Konnektor-Modell als *Komposition* bezeichnet wird [SHAW und GARLAN 1996:197] und ihrerseits wiederum als (zusammengesetzte) Komponente angesehen werden kann, sofern sie Ports zur Durchführung von Interaktionen mit der Umwelt besitzt und die übrigen Kriterien der Komponentendefinition erfüllt.

Mit dem Komponenten-Konnektor-Modell lässt sich die *Konfiguration* eines komponentenorientierten Anwendungssystems als (Software-) Architektur darstellen, die die Art, Anzahl und Hauptmerkmale der Einheiten beschreibt, aus denen das Anwendungssystem jeweils aufgebaut ist [IEEE 1983]. Gemäß den Vorgaben des Komponenten-Konnektor-Modells sind als konstituierende Einheiten eines Anwendungssystems jedoch nicht nur die einzelnen *Komponenten* zu beschreiben, sondern auch deren *Anordnung*, die durch die Konnektoren festgelegt wird. Im Allgemeinen lassen sich komponentenorientierte Anwendungssysteme daher als geordnete Ganzheiten von miteinander verknüpften Komponenten, als *Komponentensysteme*, auffassen. Als solche können sie unter Verwendung der Konzepte der allgemeinen Systemtheorie, die eine generische Methodik zur Beschreibung von Systemen liefert, formal definiert werden [TUROWSKI 2003:22f.]. Dabei lässt sich das beschriebene Komponenten-Konnektor-Modell zugleich systemtheoretisch begründen.

3.1.1 Allgemeine Systemtheorie

Die *allgemeine Systemtheorie* beschäftigt sich mit der Untersuchung und Beschreibung von Elementen, die miteinander zu einer geordneten Ganzheit, einem System, verknüpft sind, sowie mit deren jeweiligen Beziehungen zueinander [GIESECKE und RAPPE-GIESECKE 1997:397-407]. Die von der allgemeinen Systemtheorie zur Modellbildung bereitgestellten Konzepte zeichnen sich dabei vor allem dadurch aus, dass sie nicht auf eine bestimmte

wissenschaftliche Disziplin beschränkt, sondern fachübergreifend zur Beschreibung und Strukturierung von komplexen Sachverhalten anwendbar sind. Ihre Wurzeln reichen bis zum Systemdenken der antiken Philosophie zurück, sind jedoch insbesondere in der neuzeitlichen allgemeinen *Systemlehre* [BERTALANFFY 1976] sowie der *Kybernetik* [WIENER 1965] zu sehen. Diese neuere Systemtheorie [GIESECKE und RAPPE-GIESECKE 1997:397] ist vor allem aus der Einsicht entstanden, dass es zur Lösung moderner wissenschaftlicher Fragestellungen umfassender und allgemein gültiger theoretischer Konzepte zur Beschreibung komplexer Zusammenhänge in Form von Modellen bedarf. Sie wird heute in einer Vielzahl von Disziplinen als theoretische Grundlage verwendet und besitzt als *Systemtheorie der Technik* [ROPOHL 1999], auf die sich die weiteren Ausführungen beziehen, auch für die Ingenieurwissenschaften eine grundlegende Bedeutung [PAHL und BEITZ 2003:17-19].

Im Mittelpunkt der allgemeinen Systemtheorie steht das *System* als Modell eines Gebildes, das aus miteinander verknüpften Elementen besteht. Typischerweise stellt ein System dabei nicht nur eine Menge von Elementen dar, sondern besitzt eine zusätzliche Komplexität, die in seiner Beschreibung ebenfalls abzubilden ist. Als Ganzheit im aristotelischen Sinn [ARISTOTELES 300 v. Chr.] stellt es deshalb mehr als die Summe seiner Elemente dar (Prinzip der *Übersummation* [GIESECKE und RAPPE-GIESECKE 1997:397]). In Anlehnung an [ROPOHL 1978:31; ROPOHL 1999:77] lässt sich der Systembegriff wie folgt konkretisieren:

> Definition 3.1: System
>
> Ein System ist das Modell einer Ganzheit, die (a) aus miteinander verknüpften Teilen bzw. Subsystemen besteht, die (b) eine Menge äußerlich sichtbarer Attribute (wie bspw. Inputs, Outputs und Zustände) sowie Beziehungen zwischen diesen aufweist und (c) von ihrer Umwelt (bzw. einem Supersystem) abgegrenzt werden kann.

Wie aus der Definition ersichtlich wird, zeichnet sich die neuere Systemtheorie durch eine Modellbildung aus, die auf der Zusammenschau unterschiedlicher *Perspektiven* des zu beschreibenden Systems basiert und dabei mehrere Systemkonzepte zu einem einheitlichen Gesamtbild integriert. Als generische Systemkonzepte werden jeweils ein *strukturales*, *funktionales* sowie ein *hierarchisches Systemkonzept* eingeführt. Aus diesen Systemkonzepten lassen sich mehrere Modellperspektiven ableiten, für die die allgemeine Systemtheorie generische Modellierungskonzepte bereitstellt. Für die Konzeption des UNSCOM Spezifikationsrahmens sind dabei insbesondere die *statische*, *operationale*, *dynamische* sowie die *hierarchische Modellperspektive* [GIESECKE und RAPPE-GIESECKE 1997:397-407; ROPOHL 1999:75-81] von Interesse (vgl. Abb. 3.2).

Das *strukturale Systemkonzept* betrachtet Systeme hinsichtlich ihres inneren *Aufbaus* (als Struktursysteme) und ist dabei von dem Grundsatz bestimmt, dass die einzelnen Teile nicht

isoliert, sondern im Zusammenhang mit den anderen Teilen zu betrachten sind [ROPOHL 1999:75, 80]. Die Systemstruktur lässt sich durch die *statische Modellperspektive* (Struktursicht) im Systemmodell darstellen, die die konstituierenden Elemente des Systems und die zwischen den Elementen existierenden Verknüpfungen beschreibt. Als Systemelemente werden sowohl zusammengesetzte als auch einfache Teile, die im Modell nicht weiter untergliedert werden (können), betrachtet. Dabei ist die innere Struktur der einfachen Teile für den Modellierungszweck nicht relevant. Als Verknüpfungen werden alle konstanten Verbindungen, die *zwischen* den einzelnen Elementen existieren, betrachtet. Sie werden im Modell durch eine Menge von *Relationen* über den Elementen dargestellt.

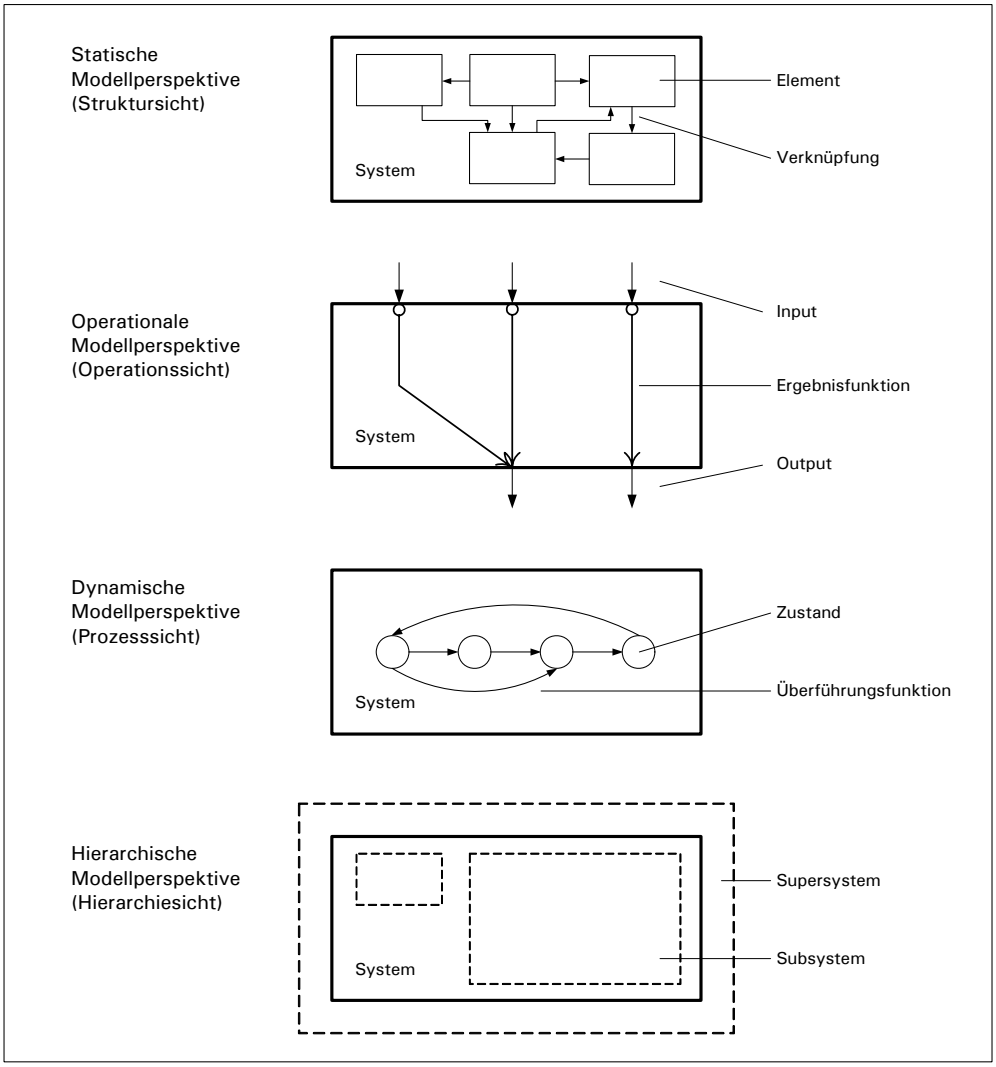


Abb. 3.2: Modellperspektiven und Modellierungskonzepte der allgemeinen Systemtheorie (in Anlehnung an [ROPOHL 1999:76]).

Von besonderem Interesse ist im Rahmen der statischen Modellperspektive zum einen die Beschreibung der *Anordnung* der Elemente, die die spezifische Komplexität des Systems

ausdrückt und durch die Gesamtheit der Relationen über den Elementen dargestellt wird. Die potenzielle Vielfalt dieser Relationen liefert den Grund dafür, dass dieselben Elemente mehrere unterschiedliche Systeme bilden können [ROPOHL 1999:80]. Zum anderen steht bei der Modellierung der Systemstruktur die von außen wahrnehmbare *Beschaffenheit* der einzelnen Elemente im Mittelpunkt. Dabei sind vor allem die Elementmerkmale im Modell zu repräsentieren, die die Integrierbarkeit des Elements in das System beeinflussen.

Die Identifizierung von Systemen als abgeschlossene Modellierungseinheiten setzt ferner voraus, dass diese sich von ihrer Umwelt, die sämtliche außerhalb der Systemgrenzen liegenden Elemente beinhaltet, auf signifikante Weise unterscheiden. Die konstituierenden Elemente eines Systems müssen demnach auf eine andere Art untereinander verbunden sein als mit den Elementen der Umwelt [GIESECKE und RAPPE-GIESECKE 1997:398f.], etwa durch unterschiedliche Verknüpfungen oder eine stärkere Intensität von Verknüpfungen. Diese sog. Innen-Außen-Differenz wird im Rahmen des *hierarchischen Systemkonzepts* [ROPOHL 1999:77, 80f.] betrachtet, wobei die Umwelt eines Systems üblicherweise als Summe derjenigen Weltausschnitte aufgefasst wird, zu denen das System eine *abgrenzende Beziehung* aufbaut. Durch die *hierarchische Modellperspektive*, die entsprechende Modellierungskonzepte bereitstellt, lässt sich die Umwelt als übergreifendes System, als sog. *Supersystem*, in das Systemmodell integrieren. Im Umkehrschluss erlaubt sie es zudem, die konstituierenden Teile eines Systems ggf. wiederum als eigenständige Systeme, als sog. *Subsysteme*, anzusehen [ROPOHL 1999:77]. Dank dieses Zusammenhangs ist es möglich, die Modellierungskonzepte der Systemtheorie auch zur Beschreibung einzelner Systemelemente und damit zur Darstellung komplexer Systeme auf unterschiedlichen Abstraktionsniveaus einzusetzen [ROPOHL 1999:77].

Neben den bereits genannten abgrenzenden Beziehungen besitzen Systeme vor allem interaktive, *funktionale Beziehungen* zu ihrer Umwelt, die sich mit dem *funktionalen Systemkonzept* als beobachtbares *Verhalten* des Systems in seiner Umwelt darstellen lassen [ROPOHL 1999:76]. Im funktionalen Konzept wird das System als Blackbox (als Funktionssystem) betrachtet und alleine aufgrund der funktionalen Zusammenhänge (*Beziehungen*) charakterisiert, die zwischen seinen nach außen sichtbaren Eigenschaften (den sog. *Systemattributen*) existieren. Diese Zusammenhänge werden als *Relationen* über den Systemattributen dargestellt. Zu den Eigenschaften, die für die Modellierung des Systemverhaltens relevant sind, gehören dabei vor allem die Ein- und Ausgaben (In- und Outputs) eines Systems sowie die verschiedenen Systemzustände, die die zu einem bestimmten Zeitpunkt jeweils möglichen Interaktionen eines Systems mit seiner Umwelt festlegen [ROPOHL 1999:75f.]. Sie lassen sich durch die operationale und die dynamische Modellperspektive in das Systemmodell integrieren.

Die *operationale Modellperspektive* (Operationssicht) fokussiert zunächst auf die Betrachtung der *In-* bzw. *Outputs* eines Systems und der zwischen diesen Attributen existierenden operationalen Beziehungen, die in der Systemtheorie als *Ergebnisfunktionen* bezeichnet werden [ROPOHL 1999:313f.]. Mit diesen Modellierungskonzepten lassen sich die möglichen *Reize*, die auf ein System von außen durch Eingaben ausgeübt werden können, sowie die darauf jeweils als Reaktion auftretenden *Wirkungen* beschreiben. Die Wirkungen werden nach außen durch die Ausgaben des Systems offenbar [ROPOHL 1999:76]. Die Ergebnisfunktionen legen so das für ein System charakteristische Reiz-Reaktions-Schema fest, das alle zwischen den einzelnen Ein- und Ausgaben existierenden Verknüpfungen umfasst.

Die *dynamische Modellperspektive* (Prozesssicht) [GIESECKE und RAPPE-GIESECKE 1997:399f.] ergänzt das Systemmodell schließlich um eine Repräsentation der verschiedenen nach außen sichtbaren Systemverfassungen, die als *Zustände* bezeichnet werden, sowie der zwischen diesen existierenden Beziehungen, den sog. *Übergangsfunktionen* [ROPOHL 1999:76]. Die Übergangsfunktionen beschreiben dabei die Zustandswechsel eines Systems, die in Abhängigkeit von den jeweils ausgeübten Reizen erfolgen. Durch eine Verknüpfung von Zuständen und Übergangsfunktionen wird es möglich, nicht nur die zu einem bestimmten Zeitpunkt jeweils möglichen Interaktionen mit einem System zu beschreiben, sondern auch bei der Interaktion auftretende Reihenfolgen im Modell darzustellen. Mit ihren Interaktionsprozessen trägt die dynamische Perspektive wesentlich zur vollständigen Beschreibung des Systemverhaltens bei [GIESECKE und RAPPE-GIESECKE 1997:399].

Neben den genannten grundlegenden Modellperspektiven werden in der allgemeinen Systemtheorie durch neuere Systemkonzepte zusätzliche Modellperspektiven definiert. So lassen sich durch eine *evolutionäre Modellperspektive* insbesondere auch strukturelle Systemveränderungen in Abhängigkeit von der Zeit im Modell darstellen. Da diese Modellperspektiven für die Konzeption des UNSCOM Spezifikationsrahmens jedoch keine Relevanz besitzen, ist zur vertiefenden Betrachtung dieser zusätzlichen Perspektiven auf die Literatur zu verweisen [ROPOHL 1999:81]. Nachfolgend wird der Systembegriff vielmehr unter Berücksichtigung der bereits eingeführten Modellperspektiven formalisiert. Danach lässt sich ein System in Anlehnung an [ROPOHL 1999:312-316] wie folgt definieren (im Folgenden bezeichnen L, M, N, P, Q, U, V, W $\in \mathbb{N}$ jeweils beliebige Indexmengen):

Definition 3.2: System (formal)

Ein System $\Sigma = (\varepsilon, \varpi, \alpha, \beta)$ ist ein Modell, das aus einem Quadrupel von Mengen besteht, wobei ε die Menge der Elemente, ϖ die Menge der Relationen zwischen den Elementen, α die Menge der von außen sichtbaren Systemattribute und β die zwischen den Attributen existierenden Funktionen darstellt.

Das erste Mengenpaar, (ε, ϖ), stellt ein *Struktursystem* dar und entspricht der *statischen Modellperspektive* (Struktursicht) eines Systems $\Sigma_{statisch}$. Diese Perspektive beschreibt die konstituierenden Elemente e_l sowie die zwischen ihnen existierenden konstanten Relationen v_m.

$$\Sigma_{statisch} = (\varepsilon, \varpi) \text{ mit } \varepsilon = \{e_l \mid l \in L\},\ \varpi = \{v_m \mid m \in M\},\ \varpi \subseteq \varepsilon \times \varepsilon \tag{3.1}$$

Das zweite Mengenpaar, (α, β), beschreibt ein *Funktionssystem* mit der Menge der nach außen sichtbaren Systemattribute α und der zwischen diesen existierenden Funktionen β. Es beinhaltet sowohl die operationale Modellperspektive (Operationssicht) $\Sigma_{operational}$ als auch die dynamische Modellperspektive (Prozesssicht) $\Sigma_{dynamisch}$, die gemeinsam das Systemverhalten beschreiben. Diese Zusammenschau ergibt sich aus der Definition 3.1, in der Eingaben und Ausgaben sowie Zustände (mit möglichen weiteren, hier jedoch nicht näher definierten Systemeigenschaften) als nach außen sichtbare Systemattribute zusammengefasst werden. Unter Hinzuziehung der beiden folgenden Definitionen lassen sich die Menge der Systemattribute α und die Menge der zwischen ihnen existierenden Funktionen β jedoch so unterteilen, dass eine getrennte Formalisierung der operationalen und dynamischen Modellperspektiven möglich wird (vgl. auch Abb. 3.3). Dabei gilt für die Systemattribute zunächst folgende Einteilung [ROPOHL 1999:313f.]:

> Definition 3.3: Eingaben, Ausgaben und Zustände
>
> Es seien $x \in \alpha$, $y \in \alpha$ und $z \in \alpha$ Attribute des Systems $\Sigma = (\varepsilon, \varpi, \alpha, \beta)$ und $a \in \alpha'$ ein beliebiges Attribut eines Systems $\Sigma' = (\varepsilon', \varpi', \alpha', \beta')$ aus der Umwelt von Σ. Dann ist x ein *Input*, wenn es in Relationen zwischen einem System aus der Umwelt und dem System (a, x) als Nachglied steht. Entsprechend ist y ein *Output*, wenn es in Relationen zwischen dem System und einem System aus der Umwelt (y, a) als Vorglied steht. Schließlich ist z ein *Zustand* des Systems, wenn es in keiner Relation mit a steht.

Den Ausführungen dieser Definition folgend können Eingaben (Inputs) $E \subseteq \alpha$, Ausgaben (Outputs) $A \subseteq \alpha$ und Zustände $Z \subseteq \alpha$ jeweils als Teilmengen der Systemattribute α dargestellt werden. In analoger Weise lassen sich auch die zwischen den verschiedenen Attributen eines Systems existierenden Beziehungen (Relationen) β in verschiedene Funktionen unterteilen [ROPOHL 1999:313f.]. Dabei sind vor allem folgende Funktionen bedeutsam:

> Definition 3.4: Ergebnis-, Überführungs- und Markierungsfunktionen
>
> Eine Funktion $f \in \beta$, $f \subseteq E \times A$ zwischen In- und Outputs heißt Ergebnisfunktion, eine Funktion $t \in \beta$, $t \subseteq E \times Z$ zwischen Inputs und Zuständen heißt Überführungsfunktion, eine Funktion $m \in \beta$, $m \subseteq Z \times A$ zwischen Zuständen und Outputs heißt Markierungsfunktion.

Abweichend von den strengeren Definitionen der Mengenalgebra werden die Funktionen dabei wie Relationen beschrieben (zur Begründung vgl. [ROPOHL 1999:312]). Die voneinander abgegrenzten Ergebnisfunktionen $F \subseteq \beta$, Überführungsfunktionen $T \subseteq \beta$ und Markierungsfunktionen $M \subseteq \beta$ können gemäß der Definition als Teilmengen der Funktionen β aufgefasst werden, die zwischen den Systemattributen existieren. Damit lässt sich zum einen die *operationale Modellperspektive* $\Sigma_{operational}$, die aus den Eingaben e_n, Ausgaben a_p sowie den Ergebnisfunktionen f_q besteht, als Mengenpaar $(E \cup A, F)$ darstellen.

$$\Sigma_{operational} = (E \cup A, F) \text{ mit } E = \{e_n \mid n \in N\}, A = \{a_p \mid p \in P\}, F = \{f_q \mid q \in Q\}, F \subseteq E \times A \tag{3.2}$$

Zum anderen lässt sich die *dynamische Modellperspektive* $\Sigma_{dynamisch}$, die aus den Ein- und Ausgaben, den Systemzuständen z_u, den Überführungsfunktionen t_v und Markierungsfunktionen m_w besteht, formalisieren und als Mengenpaar $(E \cup A \cup Z, T \cup M)$ darstellen. Diese Darstellung wird auch als *diskreter Automat* bezeichnet [ROPOHL 1999:314].

$$\Sigma_{dynamisch} = (E \cup A \cup Z, T \cup M) \text{ mit } E = \{e_n \mid n \in N\}, A = \{a_p \mid p \in P\}, Z = \{z_u \mid u \in U\},$$
$$T = \{t_v \mid v \in V\}, M = \{m_w \mid w \in W\}, T \subseteq E \times Z, M \subseteq Z \times A \tag{3.3}$$

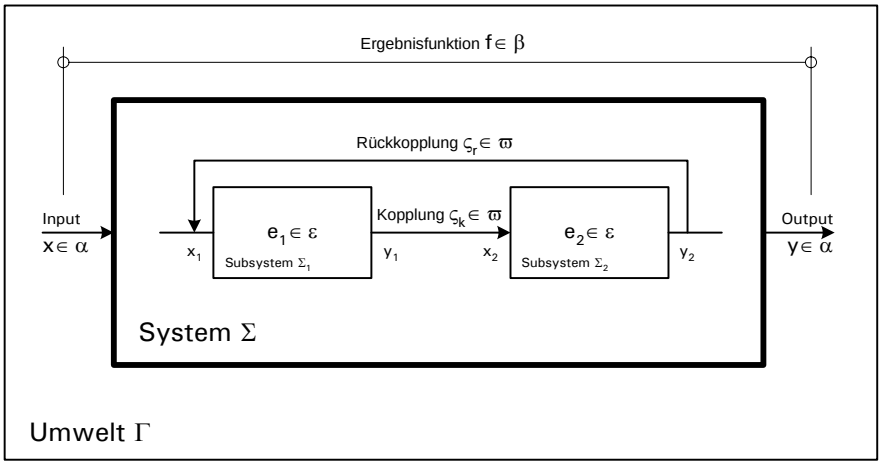


Abb. 3.3: Modellierungskonzepte der Systemtheorie (in Anlehnung an [ROPOHL 1999:314])

Ferner lässt sich unter Verwendung der *hierarchischen Modellperspektive* grundsätzlich jedes System Σ als Element e_Σ eines Supersystems $\Sigma_{Super} = (\varepsilon_{Super}, \varpi_{Super}, \alpha_{Super}, \beta_{Super})$ und jedes Element e_i eines Systems Σ als Subsystem Σ_{Sub} betrachten [ROPOHL 1999:313]. Für ein System $\Sigma = (\varepsilon, \varpi, \alpha, \beta)$ und seine Elemente $e_l \in \varepsilon$ gilt folglich

$$\Sigma = (\varepsilon, \varpi, \alpha, \beta) = e_\Sigma \in \varepsilon_{Super} \text{ und } e_l = \Sigma_{Sub} = (\varepsilon_{Sub}, \varpi_{Sub}, \alpha_{Sub}, \beta_{Sub}) \tag{3.4}$$

Dieser Zusammenhang ermöglicht es schließlich, verschiedene Arten von Relationen zwischen den Elementen eines Systems zu unterscheiden. Zu den wichtigsten Verknüpfungs-

typen gehören dabei vor allem die *Kopplung* und *Rückkopplung* (vgl. Abb. 3.3), die bestimmte Zusammenhänge zwischen den Ein- und Ausgaben der einzelnen Elemente festlegen [ROPOHL 1999:80, 315]:

> Definition 3.5: Kopplung und Rückkopplung
>
> Eine Relation zwischen zwei Subsystemen $\Sigma_1 \in \varepsilon$ und $\Sigma_2 \in \varepsilon$ heißt Kopplung $\varsigma_k = (\Sigma_1, \Sigma_2) \in \varpi$, wenn ein Output y_1 von Σ_1 zum Input x_2 von Σ_2 wird. Eine Relation zwischen zwei gekoppelten Subsystemen Σ_1 und Σ_2 heißt Rückkopplung $\varsigma_r = (\Sigma_2, \Sigma_1) \in \varpi$, wenn ein Output y_2 von Σ_2 zum Input x_1 von Σ_1 wird.

3.1.2 Kompositionen

Mit den Konzepten der allgemeinen Systemtheorie lassen sich komponentenorientierte Anwendungssysteme, deren Software-Architektur im Komponenten-Konnektor-Modell als *Komposition* dargestellt wird, nunmehr auch formal als Komponentensysteme definieren. Dabei werden Anwendungssysteme unter einem speziellen *software-technischen* Blickwinkel betrachtet, der sich vom weiter gefassten Begriffsverständnis der wirtschaftsinformatischen Systemplanung und -entwicklung [HEINRICH 1996:25-28] unterscheidet. Dort werden (betriebliche) Anwendungssysteme in der Regel ganzheitlich als *Mensch-Aufgabe-Technik-Systeme* [HEINRICH 1993:13f.] betrachtet und dementsprechend umfassender, nämlich als sog. *soziotechnische Systeme* [ROPOHL 1999:135-150] definiert.

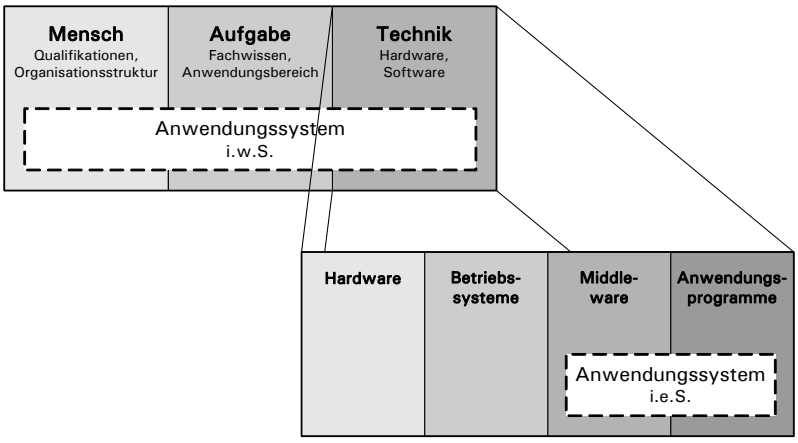


Abb. 3.4: Elemente von Anwendungssystemen (in Anlehnung an [TUROWSKI 2003:22]).

Für dieses weiter gefasste Begriffsverständnis (vgl. Abb. 3.4) ist vor allem die Auffassung charakteristisch, dass neben *technischen* Elementen, d.h. der zur Unterstützung einer (betrieblichen) Aufgabe eingesetzten *Soft- und Hardware* [FERSTL und SINZ 1998:4], auch *fachliche* und *organisatorische* Elemente zum Anwendungssystem hinzuzuzählen sind. So

werden im Rahmen der Systemplanung und -entwicklung auch die mit der Software zu unterstützende (betriebliche) Aufgabe und das hierzu benötigte Wissen aus dem Anwendungsbereich sowie die für die Nutzung der Software erforderlichen Organisationsstrukturen und Personalqualifikationen als Elemente eines (betrieblichen) Anwendungssystems behandelt und in den Entwicklungsprozess einbezogen [HEINRICH 1993:13f.]. Eine systemtheoretische Definition, die den ganzheitlichen Charakter von Anwendungssystemen berücksichtigt, findet sich in [TUROWSKI 2003:22f.].

Mit dem Komponenten-Konnektor-Modell werden jedoch ausschließlich die softwaretechnischen Bestandteile von Anwendungssystemen, also Anwendungssysteme im engeren Sinn (vgl. Abb. 3.4), betrachtet. Diesem spezielleren Begriffsverständnis entsprechend lassen sich (betriebliche) Anwendungssysteme in Anlehnung an [STAHLKNECHT und HASENKAMP 1997:242] wie folgt definieren:

> Definition 3.6: (Betriebliches) Anwendungssystem
>
> Ein Anwendungssystem umfasst die Gesamtheit aller Programme, die die Bearbeitung einer Menge von Aufgaben aus einem konkreten Anwendungsbereich unterstützen. Zu diesen zählen sowohl anwendungsspezifische als auch anwendungsunabhängige, generische Programme (sog. Basissysteme bzw. Middleware). Ein betriebliches Anwendungssystem unterstützt die Bearbeitung von Aufgaben aus einem betrieblichen Anwendungsbereich.

Komponentenorientierte Anwendungssysteme besitzen außerdem eine modulare Struktur, die aus miteinander kooperierenden (Programm-) Komponenten und Konnektoren besteht. Diese Struktur wird als *Komposition* bezeichnet [SHAW und GARLAN 1996:197]:

> Definition 3.7: Komposition
>
> Eine Komposition $K = (\chi, \upsilon)$ beschreibt die Struktur eines Software-Systems. Sie besteht aus einer Menge von konstituierenden (Programm-) Komponenten χ sowie einer Menge von Konnektoren υ, durch die die Komponenten zu einer Ganzheit verbunden sind.

Die Komponenten bilden die konstituierenden Elemente des jeweiligen Anwendungssystems und repräsentieren gemeinsam, als Menge der Komponenten χ, dessen *Informationsverarbeitungsaspekt*. Sie sind durch Konnektoren miteinander verknüpft, wobei als Verknüpfung jede Interaktion betrachtet wird, die sich zwischen den einzelnen Komponenten zur Laufzeit ereignet. Der Ablauf der Interaktionen wird von den Konnektoren beschrieben, die als Menge der Konnektoren υ somit nicht nur die Anordnung der Komponenten, sondern gleichzeitig den *Interaktionsaspekt* eines Anwendungssystems repräsentieren. Unter Berücksichtigung ihrer speziellen inneren Struktur lassen sich komponentenorientierte Anwendungssysteme zusammenfassend wie folgt formal definieren:

Definition 3.8: Komponentenorientiertes Anwendungssystem

Ein komponentenorientiertes Anwendungssystem A = (χ, υ, α, β) besteht aus einem Quadrupel von Mengen, wobei χ die Menge der konstituierenden Komponenten, υ die Menge der verknüpfenden Konnektoren, α die Menge der von außen sichtbaren Systemattribute und β die zwischen den Attributen existierenden Beziehungen darstellt.

Gewöhnlich ist die innere Struktur von komponentenorientierten Anwendungssystemen entsprechend dem *Geheimnisprinzip* (vgl. Abschnitt 2.1.2) gekapselt, weswegen nur die angebotene Funktionalität, jedoch nicht deren jeweilige Realisierung nach außen bekannt ist. Damit besitzen komponentenorientierte Anwendungssysteme typischerweise eine abgrenzende Beziehung zu ihrer Umwelt und lassen sich gemäß dem hierarchischen Systemkonzept ggf. als Elemente von umfassenderen Supersystemen darstellen. Von einem *sozio-technischen* Blickwinkel aus betrachtet lässt sich ein komponentenorientiertes Anwendungssystem bspw. als Element eines Mensch-Aufgabe-Technik-Systems [HEINRICH 1993:13f.] betrachten, was dem eingangs erwähnten ganzheitlichen Verständnis von Anwendungssystemen entspricht.

Jedoch treten komponentenorientierte Anwendungssysteme auch bei Betrachtung aus einem rein *software-technischen* Blickwinkel häufig als Bestandteile umfassenderer Supersysteme auf. So führt nicht zuletzt die aktuelle wirtschaftliche Entwicklung, mit der unter anderem die Prozessorientierung und die Schaffung flexibler virtueller Kooperationen einhergehen [BROWN 2000:208f.; HERZUM und SIMS 2000:47], dazu, dass einzelne Anwendungssysteme in der Lage sein müssen, sowohl innerhalb eines Unternehmens als auch unternehmensübergreifend im Rahmen einer kooperativen Informationsverarbeitung miteinander zu interagieren. Moderne Anwendungssysteme werden daher üblicherweise nicht mehr völlig isoliert voneinander entwickelt, sondern so konzipiert, dass sie möglichst effizient mit einer Vielzahl anderer Anwendungssysteme zu *Föderationen* gekoppelt werden können [HERZUM und SIMS 2000:46-49]. Ein derart auf die flexible Kopplung ausgelegtes Anwendungssystem lässt sich jedoch wiederum als Komponente betrachten, die sich lediglich durch ihre komplexe innere Struktur auszeichnet [HERZUM und SIMS 2000:46]. Idealerweise stellt ein komponentenorientiertes Anwendungssystem somit selbst eine spezielle zusammengesetzte Komponente dar, die sich im Vergleich zur einfachen Komponente in Anlehnung an [SZYPERSKI, et al. 2002:420] allgemein wie folgt definieren lässt:

Definition 3.9: Zusammengesetzte und einfache Komponenten

Eine Komponente X = (ε, ϖ, α, β) besteht aus einem Quadrupel von Mengen, wobei ε die Menge der Elemente, ϖ die Menge der zwischen den Elementen existierenden Verknüpfungen, α die Menge der von außen sichtbaren Systemattribute und β die Menge der zwischen den Att-

ributen existierenden Beziehungen darstellt. Die innere Struktur einer zusammengesetzten Komponente $X_z = (\chi, \upsilon, \alpha_z, \beta_z)$ besteht aus einer Komposition $K = (\chi, \upsilon) = (\varepsilon_z, \varpi_z)$ aus Komponenten χ und Konnektoren υ. Die innere, bei der Strukturierung des Anwendungssystems zunächst nicht weiter untergliederte Struktur einer einfachen Komponente X_e besteht je nach dem zur Realisierung eingesetzten Paradigma aus Klassen, Modulen oder Funktionen.

Anzumerken bleibt, dass der vorgenommenen rekursiven Anwendung des Komponentenkonzepts in der Literatur nicht ausnahmslos zugestimmt wird. So wird in [ORFALI, et al. 1996:34] der gegenteilige Standpunkt vertreten, dass ein Anwendungssystem als Ganzes keinesfalls eine Komponente darstellt. Darüber hinaus ist zu erwähnen, dass sich die Kopplung von Anwendungssystemen im Rahmen der inner- bzw. überbetrieblichen Anwendungsintegration, dem sog. *Enterprise Application Integration* (EAI), in der Praxis durchaus von der Kopplung von Komponenten während der Anwendungsentwicklung unterscheidet. Insbesondere werden Anwendungssysteme im Allgemeinen nur lose miteinander verbunden, weshalb ihre Verbindung auch als *Föderation* und nicht als *Komposition* bezeichnet wird [HERZUM und SIMS 2000:47, 234]. Da die verschiedenen Kopplungstechniken prinzipiell jedoch unabhängig von den zu verbindenden Elementen eingesetzt werden können und zudem zahlreiche Gemeinsamkeiten aufweisen, überwiegen die Vorteile der hier vorgenommenen gemeinsamen Betrachtung unter Verwendung eines einheitlichen Komponentenkonzepts.

3.1.3 Komponenten

Die im Rahmen des Entwicklungs- bzw. Integrationsprozesses miteinander zu einem Ganzen verbundenen Komponenten bilden die konstituierenden *Elemente* eines komponentenorientierten Anwendungssystems bzw. einer Föderation von Anwendungssystemen. Sie fungieren als *Strukturierungseinheiten* und repräsentieren im Modell diejenigen Orte, an denen eine kooperative Informationsverarbeitung erfolgt. Gemeinsam stellen sie somit den *Informationsverarbeitungsaspekt* des jeweils zu modellierenden Systems dar [SHAW und GARLAN 1996:197]. Als eigenständige Elemente kapseln die einzelnen Komponenten ihre innere Struktur und bieten nach außen eine unter fachlichen Gesichtspunkten abgeschlossene, selbstständige Funktionalität an. Außerdem werden sie üblicherweise in ausführbarer, kompilierter Form bereitgestellt, so dass sie während der Konfigurierung bzw. Wartung unabhängig voneinander eingesetzt und ausgewechselt werden können (zur ausführlichen Diskussion dieser Eigenschaften von Komponenten vgl. Abschnitt 2.1.4).

Wegen seiner grundlegenden Bedeutung für die (betriebliche) Anwendungsentwicklung existieren in der Literatur zahlreiche Definitionen für den Komponentenbegriff, die zum

Teil erheblich voneinander abweichen. Eine ausführliche Gegenüberstellung der unterschiedlichen Begriffsdefinitionen findet sich bspw. in [SZYPERSKI, et al. 2002:195-204]. Mit der zunehmenden Akzeptanz des modernen komponentenorientierten Entwicklungsparadigmas hat sich inzwischen jedoch ein weitgehend einheitliches Begriffsverständnis durchgesetzt. Es basiert auf der in [SZYPERSKI 1998:34; SZYPERSKI, et al. 2002:41] gegebenen Definition, wobei sich durch eine Betrachtung von Komponenten auf verschiedenen Abstraktionsebenen ergänzende Definitionsteile ergeben (vgl. hierzu Abb. 2.4). Unter Berücksichtigung und Integration dieser verschiedenen Blickwinkel lässt sich der Komponentenbegriff in Anlehnung an [BROY 1997:137; D'SOUZA und WILLS 1999:387; HERZUM und SIMS 2000:40; BASTER, et al. 2001:92; ACKERMANN, et al. 2002:1; SZYPERSKI, et al. 2002:41; OMG 2003b:136f.] allgemein wie folgt definieren:

> Definition 3.10: Komponente (allgemein)
>
> Eine Komponente ist eine Software-Einheit, die eine fachlich eigenständige Funktionalität realisiert, ihre Implementierung verbirgt, lediglich explizite Kontextabhängigkeiten besitzt und Dienste über wohldefinierte Schnittstellen zur Verfügung stellt bzw. von anderen Komponenten in Anspruch nimmt. Sie kann ohne Modifikation ihrer Implementierung mit anderen Komponenten zu größeren Einheiten verbunden werden und ist unabhängig von anderen Komponenten austauschbar.

Diese Definition beschränkt sich auf die Festlegung der zentralen Merkmale von Komponenten, die zahlreichen verschiedenen Komponentenarten gemeinsam sind. Hinsichtlich des allgemeinen Aufbaus, d.h. der Software-Architektur komponentenorientierter Anwendungssysteme lassen sich dabei vor allem *Fachkomponenten*, *Systemkomponenten*, *Komponenten-Anwendungs-Frameworks* sowie *Komponenten-System-Frameworks* als Komponentenarten unterscheiden (vgl. Abb. 3.5).

Fach- und Systemkomponenten bieten spezialisierte *Dienste* an, die zur Erfüllung der Aufgaben des jeweils zu entwickelnden Anwendungssystems benötigt werden. Die bei der Entwicklung eingesetzten Fachkomponenten (*Anwendungskomponenten* [SIEDERSLEBEN 2004:165f.] bzw. *Application Domain-Specific Components* [BROWN 2000:118f.; GRISS 2001:151f.]) realisieren (fach-) spezifische Dienste aus dem Anwendungsbereich des jeweils zu entwickelnden Systems[9], während von den Systemkomponenten (*Infrastructure* bzw. *Middleware Components* [BROWN 2000:119f.; GRISS 2001:151f.]) generische Dienste erbracht werden, die unabhängig vom jeweiligen Anwendungsbereich einsetzbar sind:

[9] Bisweilen wird der Begriff „Fachkomponente“ mit einem engeren Verständnis auch synonym zum Begriff „Business Component“ verwendet [ACKERMANN, et al. 2002:1; TUROWSKI 2003:19].

Definition 3.11: Fach- und Systemkomponenten

Eine Fachkomponente ist eine Komponente, die eine Menge von Diensten einer anwendungsspezifischen (vertikalen) Domäne anbietet. Eine Systemkomponente ist eine Komponente, die eine Menge von Diensten einer anwendungsunabhängigen, generischen (horizontalen) Domäne anbietet.

Zu den bei der Anwendungsentwicklung häufig eingesetzten Systemkomponenten zählen bspw. Expertensysteme, Datenbank- und Workflow-Managementsysteme. Unter den Fachkomponenten sind vor allem Komponenten hervorzuheben, die Dienste aus *betrieblichen* Anwendungsbereichen anbieten und dementsprechend als betriebliche Fachkomponenten bzw. *Business Components* [HERZUM und SIMS 2000:40; BASTER, et al. 2001:92] bezeichnet werden. Diese Komponenten, zu denen unter anderem Kundenverwaltungs-, Lagerverwaltungs- oder Finanzbuchhaltungssysteme gehören, besitzen im Rahmen der betrieblichen Anwendungsentwicklung eine wesentliche Bedeutung.

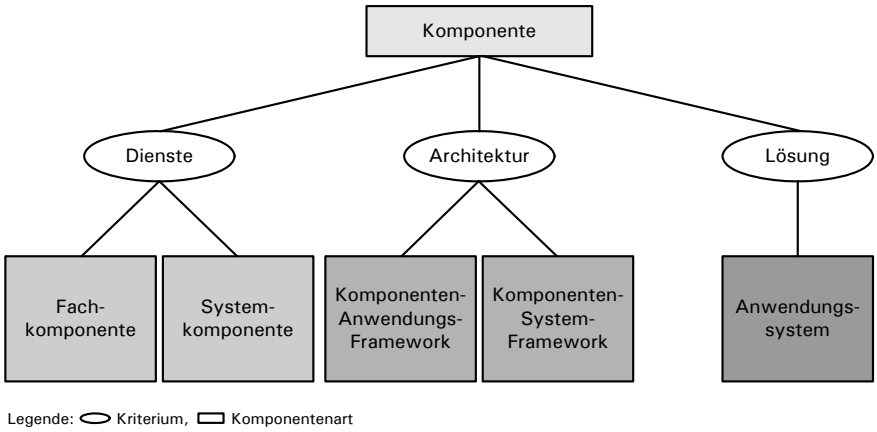


Abb. 3.5: Architekturspezifische Einteilung von Komponentenarten.

Neben Komponenten, die spezialisierte Dienste für die Entwicklung von Anwendungssystemen anbieten, gibt es auch solche, die Teile seiner *Architektur* in Form eines (Code-) Rahmens festlegen und implementieren. Durch diesen (Code-) Rahmen, der durch weitere Programme (Komponenten) zu einem Anwendungssystem zu ergänzen ist, wird das Zusammenwirken der einzelnen Komponenten schematisch geregelt [SZYPERSKI, et al. 2002:419f.]. Dies geschieht bspw. durch die Festlegung bestimmter Abläufe oder die verbindliche Anordnung der Komponenten gemäß einem Strukturgerüst [GRIFFEL 1998:115; SZYPERSKI, et al. 2002:419f.].

Entsprechend ihrer Funktion werden diese Komponenten auch als *Komponenten-Frameworks* bezeichnet [SZYPERSKI, et al. 2002:419]. Sie lassen sich als Blackbox-Frameworks [PREE 1997:46, 63-84] ausschließlich durch Ankopplung von Komponenten an den spe-

ziell dafür vorgesehenen Stellen (Hot Spots) erweitern [D'SOUZA und WILLS 1999:27f.]. Damit unterscheiden sie sich vor allem von objektorientierten Whitebox-Frameworks, bei denen die Erweiterung durch eine feste Integration von Programmen unter Etablierung von Vererbungsbeziehungen erfolgt. Die mittels Vererbung und Spezialisierung durchgeführte Komposition verursacht jedoch eine enge Kopplung der verschiedenen Teile des entstehenden Anwendungssystems, wodurch vor allem deren unabhängige Wartung erschwert wird [D'SOUZA und WILLS 1999:28]. Zudem können mit diesem Konzept ausschließlich Teile verbunden werden, die in der gleichen Programmiersprache realisiert wurden. Da beide Eigenschaften der geforderten Flexibilität eines Komponentenansatzes widersprechen, sind Whitebox-Frameworks für eine Verwendung als Komponenten-Frameworks im Allgemeinen nicht geeignet [D'SOUZA und WILLS 1999:28].

Komponenten-Frameworks lassen sich hinsichtlich des von ihnen jeweils festgelegten Architekturteils eines Anwendungssystems in Komponenten-Anwendungs-Frameworks und Komponenten-System-Frameworks unterteilen [GRIFFEL 1998:114; TUROWSKI 2003:39f.]. Durch die Komponenten-Anwendungs-Frameworks wird zunächst die Architektur der anwendungsspezifischen Teile eines Anwendungssystems, d.h. vor allem die Anordnung der einzusetzenden Fachkomponenten, festgelegt. Sie ist mit der Architektur der generischen Teile eines Anwendungssystems abzustimmen, die durch die eingesetzten Komponenten-System-Frameworks bestimmt wird. Die eingesetzten Komponenten-System-Frameworks legen nicht nur die Anordnung der Systemkomponenten fest, sondern stellen gleichzeitig eine Integrationsplattform für das gesamte Anwendungssystem dar:

> Definition 3.12: Komponenten-Anwendungs- und Komponenten-System-Frameworks
>
> Komponenten-Anwendungs-Frameworks sind Komponenten, die die Architektur des anwendungsspezifischen Teils eines Anwendungssystems festlegen. Komponenten-System-Frameworks sind Komponenten, die die Architektur des generischen Teils eines Anwendungssystems festlegen. Komponenten-System-Frameworks bilden darüber hinaus die Integrationsplattform des Anwendungssystems.

Gemeinsam repräsentieren die ggf. in verschiedenen Hierarchiestufen angeordneten Komponenten-Frameworks somit die Architektur eines Anwendungssystems [SZYPERSKI, et al. 2002:419], die in Abb. 3.6 modellhaft dargestellt ist. Beispiele für Komponenten-Anwendungs-Frameworks finden sich am Markt in Form des Microsoft .NET Business Frameworks bzw. der Common Business Objects des IBM San Francisco Frameworks [WESKE 1999:9f.]. Als Komponenten-System-Frameworks sind vor allem die verschiedenen Applikationsserver, die teilweise als Bestandteil des Betriebssystems ausgeliefert werden, hervorzuheben. Abschließend ist noch darauf hinzuweisen, dass neben den genannten Komponenten auch komponentenorientierte *Anwendungssysteme* als eigenständige Komponen-

tenart anzusehen sind (vgl. Abb. 3.5). Diese Komponenten decken ihren Aufgabenbereich jeweils in Form einer vollständigen, lauffähigen *Lösung* ab. Sie lassen sich mit anderen Anwendungssystemen zu Föderationen verbinden (zur ausführlichen Diskussion vgl. Abschnitt 3.1.2).

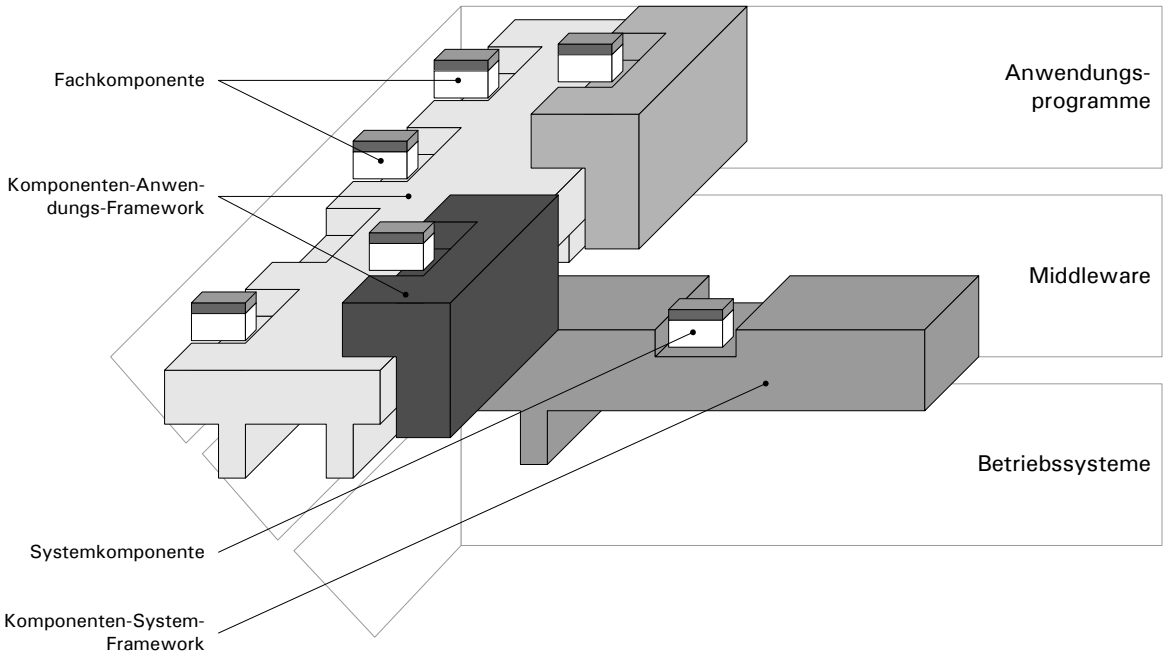


Abb. 3.6: Allgemeines Modell der Software-Architektur eines komponentenorientierten Anwendungssystems (in Anlehnung an [SZYPERSKI, et al. 2002:422; TUROWSKI 2003:38, 40]).

Bezogen auf die eingangs dargestellte, allgemeine Komponentendefinition ist ferner anzumerken, dass sie zwar die durchgängige *Modularität* auf den verschiedenen Abstraktionsebenen betont, jedoch weder die *Wiederverwendbarkeit* noch die *Vermarktbarkeit* von Komponenten explizit fordert. Damit steht sie scheinbar im Widerspruch zu einigen Komponentendefinitionen in der Literatur, die gerade diese Merkmale als grundlegende Eigenschaften ansehen [HERZUM und SIMS 2000:40; BASTER, et al. 2001:92; ACKERMANN, et al. 2002:1]. Auf eine Übernahme dieser Merkmale in die allgemeine Komponentendefinition wurde jedoch verzichtet, da sich durch die Einführung der Komponentenorientierung auch dann viele Fortschritte erzielen lassen, wenn eine Wiederverwendbarkeit von Komponenten nicht angestrebt wird. So trägt die Komponentenorientierung bereits mit ihrem durchgängig modularen Konzept entscheidend zur Beherrschung der Anwendungskomplexität, zur Steigerung der Anpassungsfähigkeit von Anwendungssystemen sowie zu einer flexibleren Anwendungsentwicklung bei [BROWN 2000:74f.; CHEESMAN und DANIELS 2001:2]. Diese zählen zu den zentralen Herausforderungen in der betrieblichen Anwendungsentwicklung bzw. im Software Engineering (vgl. hierzu auch Kapitel 1).

Die Einführung eines Konzepts zur *Wiederverwendung* von Komponenten dient demgegenüber vor allem der Verkürzung der Entwicklungszeit und der Steigerung der Qualität durch das Einbeziehen von bereits existierenden, ausgereiften Bauteilen [LIM 1994; BROWN 2000:74]. Die zuvor dargestellte allgemeine Komponentendefinition begünstigt die Wiederverwendbarkeit von Komponenten dadurch, dass sie die *Dokumentation* der Schnittstellen und Kontextabhängigkeiten sowie die Realisierung einer auch unter fachlichen Gesichtspunkten *eigenständigen Funktionalität* verlangt. Im Allgemeinen sind jedoch weitere Maßnahmen erforderlich, um die effiziente Wiederverwendbarkeit von Komponenten zu garantieren. So ist während der Entwicklung bspw. durch eine generalisierende Vorgehensweise (*Development for Reuse*) sicherzustellen, dass die einzelnen Komponenten von einem konkreten Problemkontext unabhängig und dadurch in verschiedenen Anwendungsszenarien einsetzbar werden [SOMMERVILLE 1992:309-320; SAMETINGER 1997:173f.]. Da gerade durch diese Maßnahmen ein erheblicher Mehraufwand entsteht, der zur Realisierung der übrigen genannten Vorteile der Komponentenorientierung nicht zwangsläufig betrieben werden muss, wird auf eine explizite Forderung nach Wiederverwendbarkeit in der allgemeinen Komponentendefinition verzichtet.

Dessen ungeachtet werden durch die Komponentendefinition allerdings die einzusetzenden Wiederverwendungstechniken eingeschränkt. So ist aufgrund der Forderung, dass die Verbindung von Komponenten ohne eine Modifikation ihrer Implementierungen zu erfolgen hat, jede Art der Wiederverwendung ausgeschlossen, die auf einer Veränderung des Programmcodes beruht. Eine *Anpassung* von Komponenten kann daher allenfalls durch das Setzen von Parametern erfolgen, die vom Entwickler vorgesehen wurden [D'SOUZA und WILLS 1999:387]. Bedingt durch die Tatsache, dass Komponenten ihre Implementierung verbergen, ist bei der Wiederverwendung im Allgemeinen auch keine *Einsichtnahme* in den Programmcode möglich. Damit schließt die Komponentendefinition jede Form der *Whitebox-Wiederverwendung*, die eine Einsichtnahme oder Veränderung des Programmcodes gestattet, zugunsten einer *Blackbox-Wiederverwendung* aus, bei der lediglich die Schnittstellen und die zugehörige Dokumentation der Komponenten eingesehen werden dürfen [SAMETINGER 1997:29; SZYPERSKI, et al. 2002:40f.]. Eine Wiederverwendung gemäß dem Blackbox-Prinzip verhindert dabei vor allem, dass bei der Konfigurierung Abhängigkeiten auf Teile der Implementierung von Komponenten entstehen, die etwa im Falle eines Versionswechsels Inkompatibilitäten verursachen können [SAMETINGER 1997:29f.; SZYPERSKI, et al. 2002:41]. Zudem unterstützt das Konzept der Blackbox-Wiederverwendung den Komponentenhersteller beim Schutz seines Fachwissens, das sich vor allem in der Implementierung seiner Komponenten offenbart. Eine vertiefende Diskussion der verschiedenen Nachteile einer Whitebox-Wiederverwendung findet sich bei [SAMETINGER 1997:28-30; SZYPERSKI, et al. 2002:40-42, 109-138].

Allerdings macht die allgemeine Komponentendefinition keine Einschränkungen hinsichtlich verschiedener Wiederverwendungsmöglichkeiten und ihren jeweils unterschiedlichen Vor- und Nachteilen. So kann zur Inanspruchnahme der von einer Komponente angebotenen Dienste entweder eine Kopie der Komponente installiert, in das Anwendungssystem eingebunden und anschließend aufgerufen werden. Diese derzeit vorherrschende Vorgehensweise entspricht einer *logischen Wiederverwendung*, da von verschiedenen Anwendungen jeweils *das gleiche* Artefakt in Form einer Kopie verwendet wird. Eine *physische Wiederverwendung* liegt dagegen dann vor, wenn eine Komponente von einem Anbieter einmal installiert wurde und ihre Dienste von den Anwendungssystemen jeweils über einen Prozedurfernaufruf in Anspruch genommen werden. Bei dieser Vorgehensweise, die vor allem bei der Verwendung von XML Web-Services bzw. CORBA Komponenten zum Einsatz kommt, wird *dasselbe* Artefakt von verschiedenen Anwendungen verwendet.

3.1.4 Ports

Die von den Komponenten angebotenen bzw. nachgefragten *Dienste* repräsentieren die verschiedenen Aktionen im Systemmodell, die zur Laufzeit ausführbar sind. Bei der Ausführung werden ggf. verschiedene Ein- und Ausgaben im Rahmen einer Interaktion zwischen dem jeweiligen Dienstnutzer und dem Erbringer des Dienstes ausgetauscht:

Definition 3.13: Dienst

Ein Dienst ist eine zur Laufzeit ausführbare Aktion, die die Bewältigung einer oder mehrerer Aufgaben unterstützt. Die Ausführung kann davon abhängig sein, dass geeignete Größen als Parameter übergeben werden und mit der Rückgabe einer Ergebnisgröße enden.

Mit diesem Konzept lassen sich die verschiedenen *Reize*, die auf eine Komponente in Form von Aufrufen ausgeübt werden können, als nach außen *angebotene Dienste* im Systemmodell beschreiben und gleichzeitig in einen funktionalen Zusammenhang mit der ggf. erfolgenden *Wirkung* setzen. Um sowohl die Reize als auch die jeweils korrespondierenden Wirkungen näher zu charakterisieren, sind bei der Beschreibung eines Dienstes die als Parameter übergebenen Eingaben sowie die als Reaktion gelieferte Ausgabe anzugeben. Demgemäß lassen sich Dienste mit den Konzepten der Systemtheorie als *Ergebnisfunktionen* [ROPOHL 1999:313f.] formal definieren:

Definition 3.14: Dienst (formal)

Ein Dienst $\Delta \in \delta \subseteq E_\Delta \times A_\Delta \subseteq E \times A \subseteq \beta$ ist eine Ergebnisfunktion f_Δ zwischen den Eingaben $E \subseteq \alpha$ und den Ausgaben $A \subseteq \alpha$ einer Komponente $X = (\chi, \upsilon, \alpha, \beta)$, die diesen im Rahmen ihrer Implementierung realisiert.

Als Ganzheit trägt die Menge der Dienste einer Komponente δ zur Darstellung ihres Reiz-Reaktions-Schemas bei, das den Zusammenhang zwischen allen Ein- und Ausgaben beschreibt [ROPOHL 1999:75f.]. Dabei sind neben den nach außen *angebotenen* auch die durch die Komponente *nachgefragten* Dienste einzubeziehen, um zu einer vollständigen Darstellung zu gelangen [DEREMER und KRON 1976:118]. Die nachgefragten Dienste beschreiben diejenigen Reize, die eine Komponente während der Laufzeit, also während der Diensterbringung, auf andere Komponenten in Form von Aufrufen ausübt. Aus Sicht der aufrufenden Komponente zählen die hierbei übergebenen Parameter zu den Ausgaben und das zurück gelieferte Ergebnis zu den Eingaben. Sie sind schon deswegen bei der Darstellung des Reiz-Reaktions-Schemas einer Komponente zu berücksichtigen.

Durch die nachgefragten Dienste werden nicht nur zusätzliche Ein- und Ausgaben einer Komponente, sondern zugleich auch die *Abhängigkeiten* beschrieben, die sie in Bezug auf ihre jeweilige Umwelt besitzt. Mit der Festlegung eines nachgefragten Dienstes wird gefordert, dass dieser in exakt der beschriebenen Weise durch die Umwelt, d.h. durch andere Komponenten erbracht wird. Werden diese Abhängigkeiten nicht erfüllt, ist die nachfragende Komponente unter Umständen nicht mehr in der Lage, ihre Dienste anzubieten bzw. zur Laufzeit zu erbringen. Die Beschreibung der nachgefragten Dienste mit dem Ziel, die Kontextabhängigkeiten von Komponenten zu beschreiben, ist damit zugleich eine wesentliche Voraussetzung, um die Forderung nach expliziten Kontextabhängigkeiten von Komponenten (vgl. Definition 3.10) zu erfüllen.

Bei der Beschreibung werden zusammengehörige Dienste, die während der Laufzeit typischerweise gemeinsam in Anspruch genommen werden, zu größeren Einheiten, den *Komponentenschnittstellen*, zusammengefasst. Hierdurch lässt sich einerseits eine rollenspezifische Gruppierung der Dienste erreichen [BROWN 2000:106-109; SZYPERSKI, et al. 2002:42]. Andererseits führt die Zusammenfassung von Diensten zu größeren Einheiten vor allem auch zu einer besseren Überschaubarkeit der zwischen den Komponenten existierenden Abhängigkeiten [CHEESMAN und DANIELS 2001:122]. Allgemein gilt „jede gedachte oder tatsächliche Verbindung zweier interagierender Systeme“ als Schnittstelle [HEINRICH 1993:333], weshalb neben Schnittstellen zwischen Programmbausteinen zunächst auch Mensch-Maschine-Schnittstellen (z.B. Benutzungsoberflächen) sowie Software-Hardware-Schnittstellen unter diesen Begriff fallen.

Im Folgenden wird jedoch ausschließlich die erstgenannte Schnittstellenform betrachtet und der Schnittstellenbegriff zudem in einem engeren Sinne benutzt. Da dem zugrunde liegenden Strukturierungsparadigma entsprechend nämlich die Verbindung von Kompo-

nenten zu größeren Einheiten im Mittelpunkt steht, lassen sich Schnittstellen prinzipiell als Komponentenschnittstellen definieren:[10]

Definition 3.15: Schnittstelle (als Komponentenschnittstelle)

Eine Schnittstelle ist eine Verbindung zweier interagierender Komponenten, die Regeln für deren Interaktion festlegt.

Mit systemtheoretischen Modellierungskonzepten lassen sich Schnittstellen unter Rückgriff auf das zuvor eingeführte Dienstkonzept beschreiben. Formal gilt somit:

Definition 3.16: Schnittstelle (formal)

Eine Schnittstelle $I = \{\Delta_l \mid l \in L\} \in \iota$ besteht aus einer Menge von Diensten $\Delta_l \in \delta$.

Wie bei Diensten lassen sich auch bei Schnittstellen sog. Angebots- und Nachfrageschnittstellen unterscheiden [D'SOUZA und WILLS 1999:387f.; SZYPERSKI, et al. 2002:42-44; OMG 2003b:137]. *Angebotsschnittstellen* fassen jeweils eine Menge von angebotenen Diensten zusammen, während *Nachfrageschnittstellen* jeweils Dienste beinhalten, die zur Laufzeit durch die Komponente aufgerufen werden. Dementsprechend werden die Abhängigkeiten einer Komponente hinsichtlich ihrer jeweiligen Umwelt durch ihre Nachfrageschnittstellen beschrieben [SZYPERSKI, et al. 2002:44]. Als zentrales Konzept werden Angebots- und Nachfrageschnittstellen nicht nur bei der *Beschreibung* der angebotenen und nachgefragten Dienste im Rahmen eines Systemmodells, sondern auch während der *Implementierung* genutzt. So findet das Schnittstellenkonzept in den meisten gängigen Programmiersprachen Anwendung, wo es zum Aufrufen von Methoden anderer Komponenten verwendet wird. Im Komponenten-Konnektor-Modell werden die Angebots- und Nachfrageschnittstellen sog. *Ports* zugeordnet, die die Ausgangs- bzw. Endpunkte für komplexe Interaktionen zwischen einer Komponente und ihrer Umwelt bilden [SHAW und GARLAN 1996:197; D'SOUZA und WILLS 1999:410; OMG 2003b:138]:

Definition 3.17: Port (formal)

Ein Port $\Pi = \{I_q \mid q \in Q\} \in \pi$ besteht aus einer Menge von Schnittstellen $I_q \in \iota$, die gemeinsam komplexe Regeln für die Interaktion einer Komponente und ihrer Umwelt festlegen.

---

[10] Dies entspricht der Vorgehensweise gängiger Modellierungstechniken (vgl. [CHEESMAN und DANIELS 2001; OMG 2003b]). In der Literatur werden jedoch bisweilen auch *Konnektoren* Schnittstellen zugeordnet [SHAW und GARLAN 1996:197]. Diese werden im nächsten Abschnitt implizit als Rollen eingeführt.

Umgekehrt lassen sich die Ports einer Komponente durch Hinzunahme des Schnittstellenkonzepts näher charakterisieren und detaillierter im Systemmodell beschreiben [OMG 2003b:136]. Üblicherweise wird zur Beschreibung der logischen Architektur eines komponentenorientierten Anwendungssystems deshalb ein um Schnittstellen erweitertes Komponenten-Konnektor-Modell verwendet (vgl. Abb. 3.7). Unter Rückgriff auf die zuvor definierten Ports lassen sich Komponenten nunmehr auch gemäß dem funktionalen Systemkonzept formal definieren, wobei zunächst die operationale Modellperspektive, also die Beschreibung der Ergebnisfunktionen, im Mittelpunkt steht:

Definition 3.18: Komponente (operational)

Eine Komponente ist ein Funktionssystem $X = (\pi)$, das aus einer Menge von Ports π besteht. Diese beschreiben die Ergebnisfunktionen $f \subseteq E \times A \subseteq \beta$ einer Komponente und die Abhängigkeiten hinsichtlich ihrer Umwelt jeweils als Menge von Diensten δ.

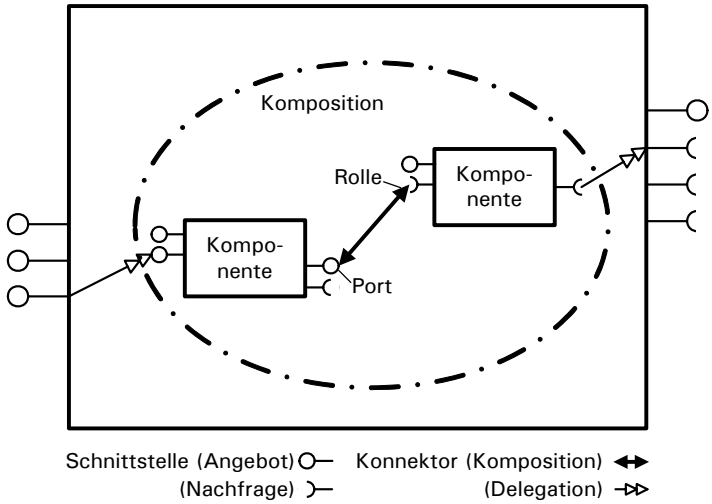


Abb. 3.7: Erweitertes Komponenten-Konnektor-Modell mit Schnittstellenkonzept (in Anlehnung an [SHAW und GARLAN 1996:197]).

Mit der Definition von Komponenten als Funktionssysteme wird eine Blackbox-Sicht in das Komponentenmodell eingeführt, die ausschließlich die nach außen sichtbaren Schnittstellen und die dort jeweils genannten Eigenschaften umfasst. Diese Sicht passt nicht nur zur allgemeinen Komponentendefinition, sondern berücksichtigt zugleich die Funktion von Komponenten als *Abstraktionseinheiten* bei der Strukturierung von Anwendungssystemen (vgl. Abschnitt 2.1.1). Dies gilt insbesondere, da es nach dem systemtheoretischen Gesetz der *Äquifunktionalität* grundsätzlich unmöglich ist, aus den beschriebenen Funktionen, also insbesondere den als Ergebnisfunktionen beschriebenen Diensten, auf die Struktur eines Systems zu schließen. Die nach außen sichtbaren Funktionen eines Systems lassen sich vielmehr stets von verschiedenen Strukturen erzeugen [ROPOHL 1999:316].

3.1.5 Konnektoren

Die Interaktionen, die sich während der Laufzeit in Form von Dienstaufrufen zwischen den Komponenten ereignen, werden im Systemmodell durch *Konnektoren* beschrieben. Sie repräsentieren den *Interaktionsaspekt* eines Anwendungssystems und stellen aufgrund ihrer Bedeutung eigenständige Bestandteile, wenn auch keine Strukturierungseinheiten, von Anwendungssystemarchitekturen dar [SHAW und GARLAN 1996:197]. Konnektoren bestimmen den Ablauf einer Interaktion zwischen Komponenten und legen dabei die Beteiligten sowie den Initiator der Interaktionen fest. Je nach Detaillierungsgrad können Konnektoren jedoch auch komplexere Abläufe in Form von Interaktionsprotokollen definieren:

> Definition 3.19: Konnektor
>
> Ein Konnektor verbindet zwei oder mehr Komponenten und legt den Ablauf der Interaktion zwischen diesen fest.

Im Systemmodell bilden die Konnektoren feste Verknüpfungen zwischen den einzelnen Elementen, den Komponenten. Die Menge υ der Konnektoren legt die *Anordnung* der Elemente eines komponentenorientierten Anwendungssystems bzw. einer Komposition fest, die ein wesentlicher Bestandteil der Systemstruktur ist. Mit den Konzepten der Systemtheorie lässt sich ein Konnektor wie folgt formal definieren:

> Definition 3.20: Konnektor (formal)
>
> Ein Konnektor $Y \in \upsilon \subseteq \times\chi \,.\, \times\chi = \ldots X_{l-1} \times X_l \times X_{l+1} \ldots$ ist eine Verknüpfung zwischen zwei oder mehreren Komponenten einer Komposition K.

Hinsichtlich des Charakters der Verbindung, die zwischen den Komponenten geschaffen wird, lassen sich verschiedene Arten von Konnektoren unterscheiden [OMG 2003b:143]. Durch einen *Delegationskonnektor* wird ein Dienstaufruf von einer Komponente auf eine andere Komponente weitergeleitet [OMG 2003b:143]. Dabei verbinden Delegationskonnektoren entweder Angebotsschnittstellen oder Nachfrageschnittstellen untereinander, bspw. um die Dienste einer Komponente auf deren innere Struktur abzubilden (vgl. Abb. 3.8). *Kompositionskonnektoren* werden dagegen eingesetzt, um Nachfrageschnittstellen mit passenden Angebotsschnittstellen zu verbinden. Dabei werden unterschiedliche Komponenten mit dem Ziel zu größeren Einheiten (Kompositionen) zusammengesetzt, eine Koordination zwischen Schnittstellennachfrage und -angebot zu erreichen [OMG 2003b:143]. Durch die Verbindung mit einem Kompositionskonnektor wird eine *Kopplung* zwischen den Komponenten hergestellt, bei der die Ausgaben einer Komponente zu Eingaben von anderen Komponenten werden. Diese lässt sich wie folgt formal beschreiben:

Definition 3.21: Kopplung

Eine Relation zwischen zwei Komponenten χ_1 und χ_2 heißt Kopplung ς_k, wenn der Output y_1 von χ_1 zum Input x_2 von χ_2 wird. Eine Rückkopplung ς_r zwischen zwei gekoppelten Komponenten liegt vor, wenn der Output y_2 von χ_2 zum Input x_1 von χ_1 wird.

Da eine Kopplung von Komponenten grundsätzlich ihre jeweiligen Schnittstellen und damit nicht ihre Implementierungen betrifft, wird die geschaffene Verbindung auch als *lose Kopplung* bezeichnet. Bei einer losen Kopplung ist die Austauschbarkeit von Komponenten gewährleistet, solange die verbundenen Schnittstellen unverändert bleiben.

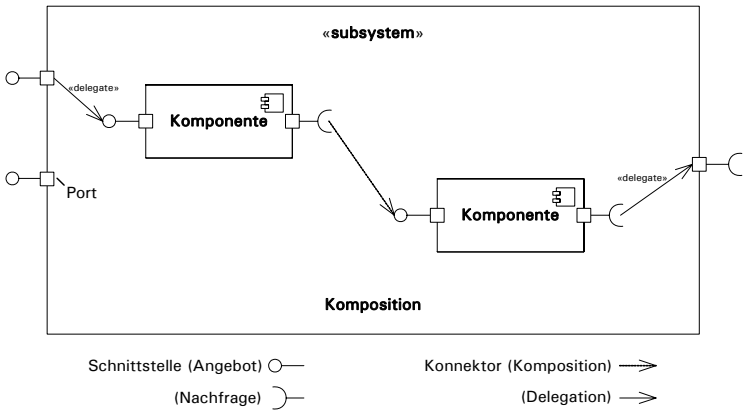


Abb. 3.8: Komponenten und Konnektoren im UML 2.0 Komponentendiagramm.

Hinsichtlich der Komplexität von Konnektoren existieren in der Literatur unterschiedliche Auffassungen. So sieht das Komponenten-Konnektor-Modell, das den Komponentendiagrammen der UML 2.0 zugrunde liegt, nur einfache Konnektoren vor (vgl. Abb. 3.8), mit denen sich die an einer Interaktion beteiligten Komponenten sowie der Initiator kennzeichnen lassen [OMG 2003b:143]. Viele Architekturbeschreibungssprachen erlauben darüber hinaus die Spezifikation eines komplexen Protokolls, das den Verlauf einer Interaktion zwischen den Beteiligten festlegt [SHAW und GARLAN 1996:197; D'SOUZA und WILLS 1999:410f.]. Aus diesem Protokoll lässt sich der Ablauf der Interaktion aus Sicht der jeweils beteiligten Komponenten ableiten. Daraus ergeben sich die *Rollen* der beteiligten Komponenten [SHAW und GARLAN 1996:197; D'SOUZA und WILLS 1999:410f.]:

Definition 3.22: Rolle

Eine Rolle legt den Ablauf einer Interaktion für einen Beteiligten fest.

Eine Komponente kann also nur dann an einer Interaktion teilnehmen, wenn sie in der Lage ist, die ihr durch den Konnektor zugewiesene Rolle einzunehmen. Dazu muss sie in der Lage sein, ihre an den Ports aufgeführten Dienste in der Reihenfolge bereitzustellen bzw.

nachzufragen, die sich aus dem Interaktionsverlauf ergibt [D'SOUZA und WILLS 1999:410]. Bei der Beschreibung von Komponenten als Funktionssysteme reicht es deshalb nicht aus, nur die Ergebnisfunktionen als Dienste zu beschreiben. Vielmehr sind zumindest auch diejenigen *Systemverfassungen* der Komponenten als relevante Systemattribute zu berücksichtigen, die die Ausführung der Dienste beeinflussen. Sie lassen sich unter Verwendung der dynamischen Perspektive im Modell darstellen, die die Beschreibung (der Ports) von Komponenten vervollständigt:

Definition 3.23: Komponente (dynamisch)

Eine Komponente ist ein Funktionssystem $X = (E \cup A \cup Z, T \cup M)$ mit $Z = \{z_u \mid u \in U\}$, $T = \{t_v \mid v \in V\}$, $M = \{m_w \mid w \in W\}$, $T \subseteq E \times Z$, $M \subseteq Z \times A$, das aus einer Menge von Ein- und Ausgaben, Zuständen Z, Überführungsfunktionen T und Markierungsfunktionen M besteht.

3.1.6 Metamodell

Die zentralen Konzepte des Komponentenmodells, die in diesem Abschnitt erläutert und systemtheoretisch begründet wurden, lassen sich zu dem in Abb. 3.9 grafisch dargestellten Metamodell zusammenfassen und gleichzeitig in einen Gesamtzusammenhang einordnen. Inhaltlich entspricht das dargestellte Metamodell dabei im Wesentlichen dem Metamodell des UML 2.0 Standards [OMG 2003b:135], das (unter anderem) den Aufbau von Komponentendiagrammen festlegt. Um die erläuterten Konzepte besser hervorzuheben, wurden jedoch die unterschiedenen Komponentenarten als Aufzählung sowie die beiden Rollen „Schnittstellenangebot“ und „Schnittstellennachfrage“ als Elemente in das Metamodell mit aufgenommen. Darüber hinaus unterscheidet das in Abb. 3.9 dargestellte Metamodell zwischen der (abstrakten) *Beschreibung* einer Komponente und den (konkreten) *Realisierungen*, die dieser Beschreibung jeweils entsprechen. Prinzipiell umfasst die Beschreibung dabei sowohl die Innen- als auch die Außensicht einer Komponente. Die Innensicht umfasst als *Strukturs*ystem die statische Modellperspektive, die den inneren Aufbau der Komponente beschreibt (vgl. Abb. 3.9 unten). Falls es sich bei der beschriebenen Komponente um eine zusammengesetzte Komponente handelt, besteht die innere Struktur wiederum aus Komponenten, die durch Konnektoren miteinander verbunden sind. Handelt es sich dagegen um eine einfache Komponente, besteht die innere Struktur in Abhängigkeit von dem zur Realisierung eingesetzten Entwicklungsparadigma aus Klassen, Funktionen oder Modulen. Die Zerlegung von einfachen Komponenten in Implementierungseinheiten wird mit dem Komponentenentwurf festgelegt und im Rahmen des konzeptionellen Komponentenmodells, dessen Darstellungen von der Realisierung unabhängig sind, zunächst nicht in die Beschreibung aufgenommen.

Die mit dem UNSCOM Spezifikationsrahmen darzustellende Außensicht umfasst – neben einigen nach außen sichtbaren, statischen Komponenteneigenschaften – als *Funktionssystem* vor allem die operationale und die dynamische Modellperspektive, die die von einer Komponente angebotenen bzw. nachgefragten Dienste beschreiben (vgl. Abb. 3.9 oben). Die operationale Perspektive beschreibt den funktionalen Zusammenhang zwischen den Ein- und Ausgaben der jeweiligen Dienste und fasst zusammengehörige Dienste zu (Angebots- und Nachfrage-) *Schnittstellen* zusammen. Die Zusammenfassung zu Schnittstellen führt dabei zu einer rollenspezifischen Gruppierung von Diensten. Diese ist unter Verwendung der dynamischen Modellperspektive durch eine Beschreibung der Interaktionen zu vervollständigen, an denen eine Komponente in ihren verschiedenen Rollen partizipiert.

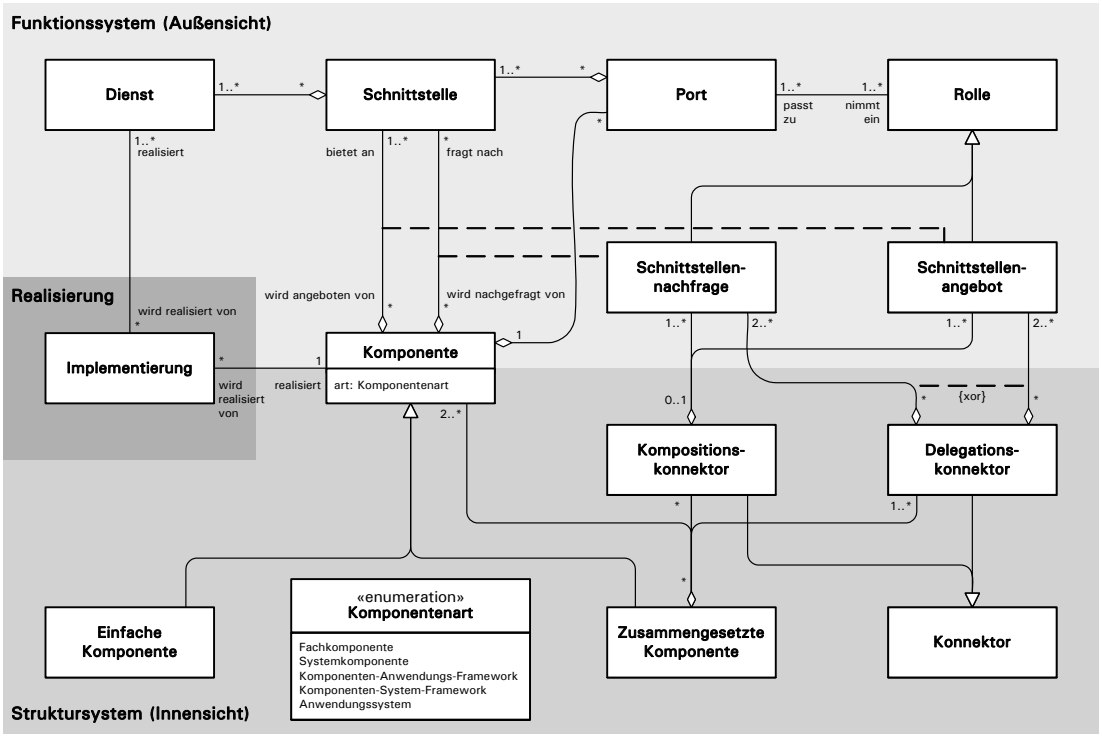


Abb. 3.9: Zentrale Elemente und Zusammenhänge im konzeptionellen Komponentenmodell.

3.1.7 Anwendung zur Architekturbeschreibung

Mit dem konzeptionellen Komponentenmodell lässt sich auch die Architektur des Verkaufssystems beschreiben, dessen Entwicklung der Unterstützung des Distributionsprozesses eines Internetversandhändlers dient (vgl. Abschnitt 2.4). Die hierzu festgelegte Architektur des Anwendungssystems lässt sich, unter Vorwegnahme der später ausführlicher darzustellenden Ergebnisse des Systementwurfs, durch ein UML 2.0 Komponentendiagramm [OMG 2003b:133-149] modellhaft darstellen (vgl. Abb. 3.10). Hauptsächlich dient

das dargestellte Modell der Architektur dabei der Beschreibung der *inneren Struktur* des zu realisierenden Anwendungssystems, dessen fachspezifischer Programmteil durch das UML Subsystem Verkaufssystem repräsentiert wird. Im Mittelpunkt steht somit die Darstellung der Komponenten, aus denen das System aufgebaut ist, sowie der zwischen ihnen existierenden Verbindungen. Allerdings sind die von den Komponenten jeweils angebotenen bzw. nachgefragten Schnittstellen zunächst nur überblicksartig und unter Verwendung einer verkürzten Notation [OMG 2003b:139] dargestellt.

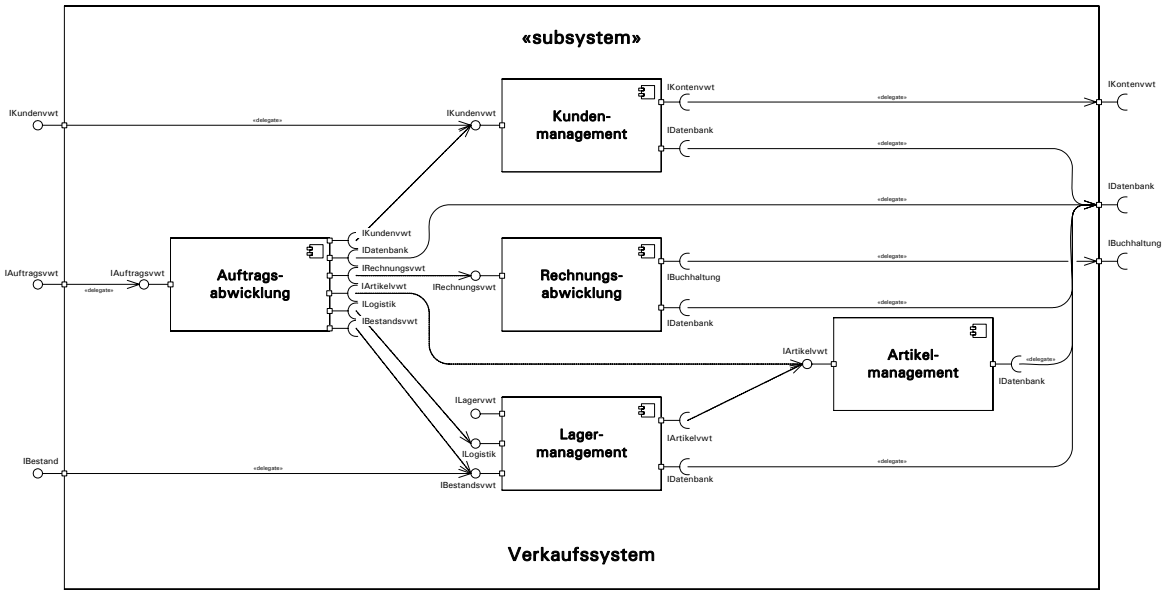


Abb. 3.10: Architektur des zur Unterstützung des Distributionsprozesses konzipierten Verkaufssystems.

Der *Verkauf* mit seinem für das Anwendungssystem zentralen Geschäftsprozess zur Annahme und *Bearbeitung von Kundenaufträgen* [BECKER und SCHÜTTE 2004:431] wird durch die Komponente Auftragsabwicklung repräsentiert. Diese Komponente koordiniert als Komponenten-Anwendungs-Framework die übrigen Komponenten des Anwendungssystems, die die anderen an der Bearbeitung von Kundenaufträgen beteiligten Funktionsbereiche repräsentieren [BECKER und SCHÜTTE 2004:396f.]. Für die Benutzung durch andere Komponenten stellt die Komponente Auftragsabwicklung Dienste zur Verwaltung von Kundenaufträgen bereit. Mit diesen Diensten lassen sich Kundenaufträge erstellen (und damit in die Bearbeitung aufnehmen), modifizieren bzw. stornieren. Im Gegenzug werden bei der Bearbeitung von Kundenaufträgen Dienste von anderen Komponenten in Anspruch genommen, mit denen sich die Daten des bestellenden Kunden abfragen, die bestellten Artikel aus dem Lager kommissionieren, die Veränderungen im Lagerbestand dokumentieren, Rechnungen erstellen sowie die Daten der Kundenaufträge in einer (unternehmensweiten) Datenbank ablegen lassen.

Die Komponente Kundenmanagement stellt als spezialisierte Fachkomponente vor allem Dienste zur *Verwaltung von Kundenstammdaten* für die Verwendung durch andere Komponenten bereit. Dabei werden Dienste anderer Komponenten verwendet, um die zu verwaltenden Daten in einer Datenbank abzulegen bzw. in die (Debitoren-) Buchhaltung einzubinden. Neben den angesprochenen Diensten sind von der Komponente Kundenmanagement üblicherweise weitere Dienste, etwa zur Auswertung von Kundenbeziehungen im Rahmen eines Customer Relationship Management (CRM), anzubieten. Solche Dienste besitzen jedoch für das im Rahmen der Fallstudie zu realisierende Anwendungssystem keine Bedeutung und sind deshalb nicht näher zu betrachten.

Der Funktionsbereich des *Artikelmanagements* wird im Verkaufssystem durch die gleichnamige Fachkomponente realisiert. Diese stellt für die Benutzung durch andere Komponenten Dienste zur *Verwaltung von Artikelstammdaten* bereit. Im Gegenzug greift sie auf die Dienste einer fremden Komponente zurück, um die zu verwaltenden Artikelstammdaten in einer Datenbank abzulegen. Auch von dieser Komponente sind im Allgemeinen zusätzliche Dienste bereitzustellen, mit denen bspw. die Sortiment- und Preispolitik im Rahmen des Marketings zu unterstützen ist. Da diese Funktionsbereiche jedoch nicht Gegenstand der Anforderungsdefinition der zu betrachtenden Fallstudie sind, wird auf eine vertiefende Betrachtung an dieser Stelle verzichtet und statt dessen auf die einschlägige Literatur verwiesen (vgl. etwa [BECKER und SCHÜTTE 2004:397-430]).

Die Fachkomponente Lagermanagement repräsentiert die Funktionsbereiche des *Wareneingangs*, des *Warenausgangs* sowie der hierzu in Beziehung stehenden *Bestandsführung*. Zur Realisierung des Warenausgangs bietet die Komponente Lagermanagement Dienste zur physischen Kommissionierung und Auslieferung von Artikeln für die Benutzung durch andere Komponenten an. Diese werden zusammen mit den (hier nicht näher betrachteten) Diensten des Wareneingangs zur Funktionsgruppe Logistik zusammengefasst und über eine gemeinsame Schnittstelle bereitgestellt. Darüber hinaus bietet die Komponente weitere Dienste an, mit denen der jeweilige Lagerbestand eines Artikels abgefragt und Veränderungen ggf. mengenmäßig erfasst werden können. Diese Dienste werden zur Funktionsgruppe Bestandsverwaltung zusammengefasst und ebenfalls über eine gemeinsame Schnittstelle bereitgestellt. Schließlich besitzt die Komponente Lagermanagement noch eine dritte Angebotsschnittstelle zur *Verwaltung der Lagerstammdaten*. Mit den dort angebotenen Diensten lassen sich die verschiedenen Lagerplätze und ihre speziellen Eigenschaften dokumentieren [BECKER und SCHÜTTE 2004:504-507]. Zur Durchführung des Lagermanagements werden Dienste einer anderen Komponente verwendet, um die Lagerdaten in einer Datenbank abzulegen. Zudem wird auf die Verwaltung der Artikelstammdaten zugegriffen, um die lagerspezifischen Daten von Artikeln abzufragen.

Die *Fakturierung* ist Bestandteil der Fachkomponente Rechnungsabwicklung, die nach außen Dienste zur *Verwaltung von Rechnungen* anbietet. Mit diesen Diensten lassen sich Rechnungen erstellen, bearbeiten und ggf. stornieren. Bei der Durchführung der Rechnungsabwicklung werden die entstehenden offenen Forderungen in die Debitorenbuchhaltung aufgenommen. Neben den Diensten zur Speicherung von Rechnungsdaten in einer Datenbank wird deshalb eine Schnittstelle zur Debitorenbuchhaltung verwendet, über die die wertmäßige Buchung eingegangener (Lastschrift-) Zahlungen und offener Forderungen für die ausgelieferten Artikel erfolgt.

Die zuvor beschriebene innere Struktur des Verkaufssystems bleibt anderen Komponenten, die mit diesem System interagieren, gemäß den Vorgaben des Geheimnisprinzips zumindest logisch verborgen. Für die Interaktion werden deshalb spezialisierte Dienste angeboten, mit denen sich Kundenstammdaten sowie Kundenaufträge verwalten und Lagerbestände von Artikeln abfragen lassen. Die zur Laufzeit erfolgenden Aufrufe dieser Dienste werden durch Delegationskonnektoren an die einzelnen, spezialisierten Komponenten des Anwendungssystems weitergeleitet. Auf die gleiche Weise werden sämtliche Abhängigkeiten von Komponenten der inneren Struktur, die nicht durch diese befriedigt werden konnten, nach außen weitergegeben. Hieraus ergeben sich die im Diagramm dargestellten Nachfrageschnittstellen des Verkaufssystems. Anders als seine Bestandteile stellt das dargestellte Verkaufssystem selbst jedoch *keine* Komponente dar, da seine inneren Bestandteile zumindest insofern sichtbar bleiben, dass sie auch in andere Anwendungssysteme (etwa zur Unterstützung des Beschaffungsprozesses) eingebunden werden können.

Das Verkaufssystem ist als primär logisches Konstrukt somit nicht unabhängig von anderen Komponenten austauschbar und erfüllt deshalb nicht die Anforderungen der Komponentendefinition (vgl. Definition 3.10). Die in der Darstellung vorgenommene Klassierung des Verkaufssystems als „Subsystem“ deutet ferner an, dass nicht alle Komponenten, sondern nur die fachspezifischen Teile des Anwendungssystems modelliert wurden. Nicht in die Darstellung einbezogen sind folglich Systemkomponenten wie etwa die Datenbank sowie diejenigen Komponenten, die die grafische Benutzungsoberfläche realisieren.

3.2 Komponententypen

Als wesentliche Bestandteile umfasst die Beschreibung der Außensicht einer Komponente gemäß dem zuvor beschriebenen Komponentenmodell die Komponentenschnittstellen einschließlich der dort aufgeführten Dienste und der verschiedenen Rollen, die eine Komponente bei Interaktionen mit ihrer Umwelt während der Laufzeit einnimmt (vgl. Abb. 3.9).

Dabei führt erst die Beschreibung der Angebots- und Nachfrageschnittstellen, die sowohl angebotene als auch nachgefragte Dienste in die Spezifikation einbezieht, zu einer vollständigen Darstellung der Eigenschaften von Komponenten [DEREMER und KRON 1976:118]. Eine solche vollständige Darstellung ist zugleich eine wesentliche Voraussetzung dafür, dass Komponenten als selbstständige Analyseeinheiten betrachtet und mit Blick auf ihre (äußeren) Eigenschaften hin untersucht werden können. Die Einführung des Schnittstellenkonzepts und die damit einhergehende Unterscheidung zwischen Angebots- und Nachfrageschnittstellen im Komponentenmodell berücksichtigt deshalb nicht nur die Funktion von Komponenten als *Abstraktionseinheiten*, sondern auch deren Funktion als *Analyseeinheiten* (vgl. Abschnitt 2.1.1).

Gleichzeitig legen die als Bestandteil einer Spezifikation dokumentierten Komponentenschnittstellen als Funktionssystem lediglich solche Eigenschaften fest, die sich prinzipiell durch verschiedene Implementierungen realisieren lassen (Gesetz der Äquifunktionalität [ROPOHL 1999:316]). Dieser Teil der Komponentenspezifikation beschreibt deshalb nicht eine einzelne Komponente, sondern charakterisiert eine Menge von Komponenten, die jeweils gleiche nach außen sichtbare Eigenschaften besitzen. Die Angebots- und Nachfrageschnittstellen definieren einen *Komponententyp*, durch den Komponenten mit nach außen identischen Eigenschaften zu einer (Äquivalenz-) Klasse zusammengefasst werden [CARDELLI und WEGNER 1985:473; D'SOUZA und WILLS 1999:33; SECO und CAIRES 2000:115f.; OMG 2003b:136]:

> Definition 3.24: Komponententyp
>
> Ein Komponententyp $\Omega \rightarrow P$, $\Omega \subseteq \iota_I$ und $P \subseteq \iota_I$, beschreibt eine Klasse von Komponenten X_I, die jeweils die durch Ω repräsentierten Dienste nachfragen und die durch P repräsentierten Dienste für die Verwendung nach außen anbieten.

Dieser besondere Umstand ermöglicht es, die Funktion von Komponenten als Analyseeinheiten nicht nur im Komponentenmodell zu berücksichtigen, sondern mit einem speziellen *Komponententypsystem* [SECO und CAIRES 2000:119-121] explizit zu unterstützen. Ein solches Typsystem stellt effiziente Algorithmen in Form eines Kalküls bereit, mit denen sich verschiedene Untersuchungen zur *Entwicklungszeit* (statisch) durchführen lassen. Im Rahmen der komponentenorientierten Anwendungsentwicklung steht dabei vor allem die Beantwortung der Frage nach der *Kombinierbarkeit* (Kompositionalität) und *Ersetzbarkeit* (Substitutionalität) von Komponenten im Mittelpunkt [SECO und CAIRES 2000:109; SZYPERSKI, et al. 2002:89]. Die zentrale Bedeutung dieser beiden Fragen ergibt sich unmittelbar aus den Konstruktionsprinzipien [DAVIS 1990:122; SCHIENMANN 1997:55], die der Entwicklung im Großen zugrunde liegen und die Entwicklungsarbeit maßgeblich prägen.

3.2.1 Konstruktionsprinzipien

Die *Komposition* von Strukturierungseinheiten, d.h. die Verbindung von Komponenten zu einem Ganzen bildet das wesentliche Konstruktionsprinzip bei der komponentenorientierten Anwendungsentwicklung (vgl. Abb. 3.11 unten). Sie basiert auf der *Abhängigkeit* (Dependenz) der einzelnen Teile, die durch die Verbindung mit passenden Teilen zu befriedigen ist. Dabei ergeben sich im Allgemeinen neue Eigenschaften, die dem entstehenden Ganzen zuzuordnen sind [SCHIENMANN 1997:59] und sich aus den Eigenschaften der einzelnen Teile ggf. durch *kompositionales Schließen* (Compositional Reasoning [CLEMENTS, et al. 2003:261f.]) ableiten lassen. Zur formalen Beschreibung der Komposition wurden verschiedene Logiken entwickelt, aus denen sich folgende allgemeine Merkmale von Teil-Ganzheits-Beziehungen ($<_p$) zwischen Objekten ableiten lassen [SIMONS 1987:37]:

Asymmetrie: $\forall\, x, y \,.\, x <_p y \Rightarrow \neg\, (y <_p x)$ (3.6)

Transitivität: $\forall\, x, y, z \,.\, (x <_p y) \wedge (y <_p z) \Rightarrow (x <_p z)$ (3.7)

Ganzheit: $\forall\, x, y \,.\, x <_p y \Rightarrow \exists\, z \,.\, (z <_p y) \wedge \neg\, (z \equiv x)$ (3.8)

Danach gilt zunächst, dass kein Objekt ein Teil von sich selbst ist (3.6). Wenn ein Objekt x ein Teil von Objekt y ist, und y wiederum ein Teil von einem Objekt z ist, so ist x ebenfalls ein Teil von z (3.7). Schließlich ist festgelegt, dass ein Ganzes y nicht nur aus einem einzigen Teil bestehen darf, sondern zumindest aus zwei verschiedenen Teilen zusammengesetzt sein muss (3.8). Auffallend ist jedoch, dass die meisten Logiken zur Beschreibung von Teil-Ganzheits-Beziehungen nicht in der Lage sind, auch die einer Komposition zugrunde liegende Abhängigkeit der einzelnen Teile in ihre formale Beschreibung einzubeziehen [SCHIENMANN 1997:62]. Diesbezüglich offenbart sich eine grundlegende Schwäche derzeit existierender Ansätze zur Beschreibung von Teil-Ganzheits-Beziehungen, die eine gesonderte Betrachtung der Abhängigkeitsbeziehung ($\leftrightarrow$) zwischen zwei Objekten notwendig macht. Für diese gelten die folgenden charakteristischen Merkmale:

Reflexivität: $\forall\, x \,.\, x \leftrightarrow x$ (3.9)

Transitivität: $\forall\, x, y, z \,.\, (x \leftrightarrow y) \wedge (y \leftrightarrow z) \Rightarrow (x \leftrightarrow z)$ (3.10)

Jedes Objekt besitzt danach zunächst eine Abhängigkeit auf sich selbst (3.9). Wenn ein Objekt x eine Abhängigkeit von einem Objekt y aufweist, und y wiederum eine Abhängigkeit von einem Objekt z aufweist, so weist auch x eine Abhängigkeit von z auf (3.10).

Neben der Komposition, bei der durch das Zusammensetzen von Teilen neue Objekte entstehen, spielt bei der komponentenorientierten Anwendungsentwicklung vor allem die

Substitution, d.h. der Austausch von Komponenten eine wichtige Rolle. Ein solcher Austausch erfolgt immer dann, wenn eine Komponente im Rahmen von Wartungsarbeiten durch eine neue Komponente zu ersetzen ist (vgl. Abb. 3.11 oben). Grundsätzlich beinhaltet auch die Phase der ersten Konfigurierung einen zumindest impliziten Substitutionsprozess, bei dem die während des Systementwurfs spezifizierten (Modell-) Komponenten durch ihre jeweiligen Realisierungen ersetzt werden [SIEDERSLEBEN 2004:59]. Der Austausch von Komponenten basiert auf der *Gleichheit* (Äquivalenz) der Komponenteneigenschaften, die über die Ersetzbarkeit entscheidet.

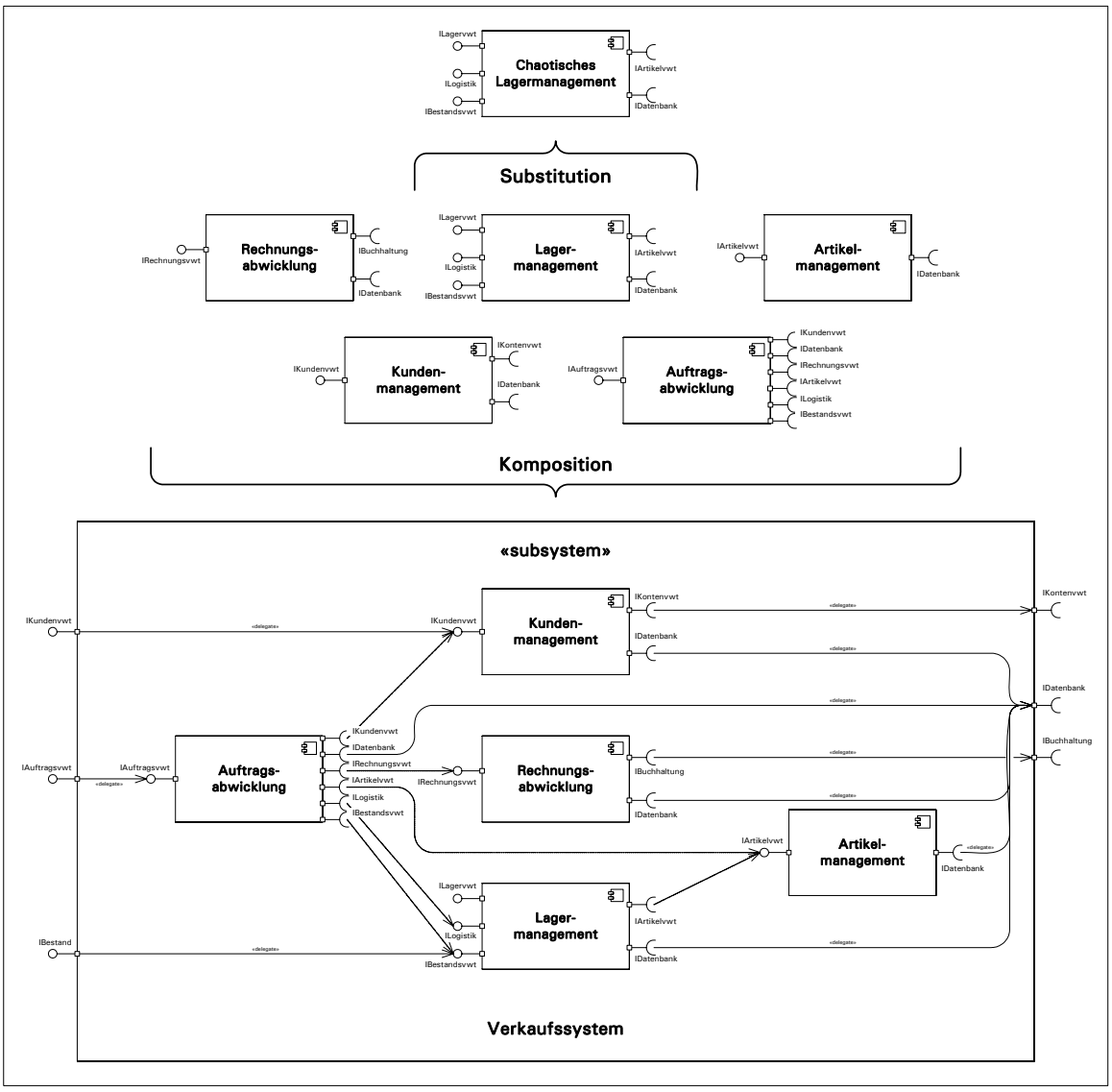


Abb. 3.11: Konstruktionsprinzipien der komponentenorientierten Anwendungsentwicklung.

Damit ein Austausch zwischen zwei Komponenten vorgenommen werden kann, müssen diese jedoch nicht zwingend gleich, also vom selben Typ sein. Vielmehr ist es im Allgemeinen bereits ausreichend, dass die Eigenschaften der neuen Komponente *konform* zu den

Eigenschaften der auszutauschenden Komponente sind und die einzufügende Komponente dementsprechend eine *Spezialisierung* der auszutauschenden Komponente darstellt [SECO und CAIRES 2000:113f.; SZYPERSKI, et al. 2002:89f.; SIEDERSLEBEN 2004:127]. Für eine Konformitätsbeziehung ($\leq_c$) zwischen zwei Objekten sind folgende allgemeine Merkmale charakteristisch:

Reflexivität: $\forall x \,.\, x \leq_c x$ (3.11)

Transitivität: $\forall x, y, z \,.\, (x \leq_c y) \wedge (y \leq_c z) \Rightarrow (x \leq_c z)$ (3.12)

Identität: $\forall x, y \,.\, (x \leq_c y) \wedge (y \leq_c x) \Rightarrow (x \equiv y)$ (3.13)

Danach gilt zunächst, dass jedes Objekt konform zu sich selbst ist (3.11). Wenn ein Objekt x konform zu einem Objekt y ist, und y wiederum konform zu einem Objekt z ist, so ist auch x konform zu z (3.12). Schließlich bleibt festzuhalten, dass zwei wechselseitig in einer Konformitätsbeziehung zueinander stehende Objekte x und y grundsätzlich gleich ($\equiv$) sind (3.13).

Komposition und Substitution von Komponenten sind *orthogonale Konstruktionsprinzipien* der komponentenorientierten Anwendungsentwicklung, da sie prinzipiell auf unterschiedlichen und voneinander unabhängigen Fundierungsrelationen basieren [SCHIENMANN 1997:63f.]. Aus diesem Grund sind auch die aus den Konstruktionsprinzipien resultierenden Fragen nach der Kompositionalität und Substitutionalität von Komponenten zunächst getrennt voneinander zu betrachten. Beide lassen sich jedoch unter Rückgriff auf das Konzept der *Konformität* von Komponentenschnittstellen beantworten, das somit das wesentliche Kriterium für den Vergleich zweier Komponentenspezifikationen durch ein Komponententypsystem bildet.

3.2.2 Kompositionalität von Komponenten

Zwei Komponenten X_1 und X_2 lassen sich miteinander durch einen Kompositionskonnektor zu einem Ganzen verbinden, falls Komponente X_2 die von der Komponente X_1 nachgefragten Dienste zumindest teilweise anbietet. Sind die von einer Komponente X_1 nachgefragten Dienste explizit spezifiziert, lässt sich die Frage nach der Kompositionalität durch einen Vergleich der dort festgelegten Diensteigenschaften mit den Eigenschaften der von X_2 angebotenen Dienste beantworten [SECO und CAIRES 2000:113]. Bezogen auf die Schnittstellen der miteinander zu verbindenden Komponenten X_1 und X_2 bedeutet dies, dass Komponente X_2 mindestens eine Angebotsschnittstelle besitzen muss, die mit einer

der Nachfrageschnittstellen von Komponente X_1 übereinstimmt, damit die Kompositionalität besteht [SECO und CAIRES 2000:120f.].

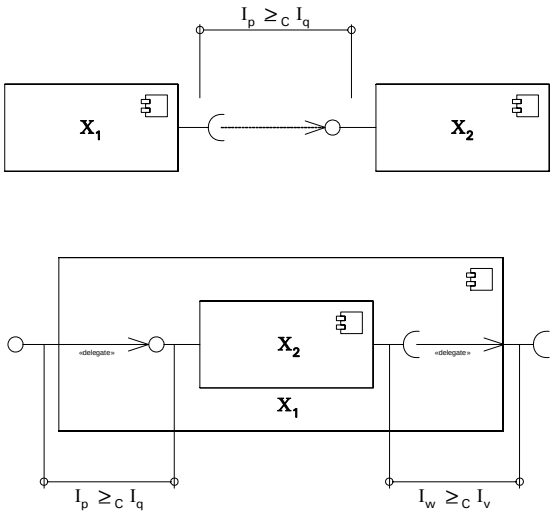


Abb. 3.12: Variabilität der Schnittstellen unter Wahrung der Kompositionalität von Komponenten.

Diese Übereinstimmung ist bspw. gegeben, falls zwei miteinander zu vergleichende Angebots- und Nachfrageschnittstellen *identisch* sind [D'SOUZA und WILLS 1999:410]. In diesem Fall beschreibt die Anforderungsdefinition der Komponente X_1 eine Menge von Diensten, die von Komponente X_2 exakt mit den geforderten Eigenschaften realisiert und angeboten wird. Die auf der Identität von Angebots- und Nachfrageschnittstellen basierende Komposition von Komponenten ist, unter Rückgriff auf die als Anwendungsbeispiel genutzte Fallstudie, in Abb. 3.11 unten dargestellt. Im Allgemeinen reicht es jedoch aus, dass die von einer Komponente X_2 angebotenen Dienste *konform* zu den Diensten sind, die als Bestandteil der Anforderungsdefinition von Komponente X_1 genannt werden [SECO und CAIRES 2000:121]. Folglich genügt es wie in Abb. 3.12 oben graphisch dargestellt, dass eine Angebotsschnittstelle $I_q \in P_{X2}$ der Komponente X_2 konform zu einer Nachfrageschnittstelle $I_p \in \Omega_{X1}$ der Komponente X_1 ist, damit die Kompositionalität zwischen beiden Komponenten gegeben ist [D'SOUZA und WILLS 1999:410, 449; SECO und CAIRES 2000:121; OMG 2003b:144]:

$$\text{Komposition:} \quad \forall X_1, X_2 \,.\; X_1 \text{ compose } X_2 \Leftrightarrow (\exists\, I_p \in \Omega_{X1} \wedge I_q \in P_{X2} \,.\, I_q \leq_c I_p) \tag{3.14}$$

Die in (3.14) geforderte Konformität zwischen den Schnittstellen besteht, falls die Angebotsschnittstelle der Komponente X_2 eine *Spezialisierung* der zu ihr korrespondierenden Nachfrageschnittstelle von Komponente X_1 darstellt [SECO und CAIRES 2000:110f.]. Das Vorhandensein einer Konformitätsbeziehung zwischen den zu verbindenden Schnittstellen

gewährleistet somit, dass diese miteinander (steck- bzw. plug-) *kompatibel* sind und durch einen Kompositionskonnektor verbunden werden können [D'SOUZA und WILLS 1999:449]. Ein vergleichbarer Zusammenhang gilt auch bei der Verbindung von Schnittstellen durch einen Delegationskonnektor. Das Ziel einer solchen Verbindung ist jedoch eine Weiterleitung von Dienstaufrufen, die entweder zwischen Angebots- oder zwischen Nachfrageschnittstellen erfolgen kann. Dafür ist es notwendig, dass die Schnittstelle, die das Ziel der Weiterleitung ist, in einer Konformitätsbeziehung zu derjenigen Schnittstelle steht, von der die Weiterleitung ausgeht (vgl. Abb. 3.12 unten):

Delegation: $\forall X_1, X_2 . \; X_1 \text{ delegate } X_2 \Leftrightarrow (\exists I_p \in P_{X1} \wedge I_q \in P_{X2} . \; I_q \leq_c I_p)$ (3.15)

$\forall X_1, X_2 . \; X_2 \text{ delegate } X_1 \Leftrightarrow (\exists I_v \in \Omega_{X1} \wedge I_w \in \Omega_{X2} . \; I_v \leq_c I_w)$ (3.16)

Der Zusammenhang, der der erfolgreichen Weiterleitung von Dienstaufrufen zwischen den Schnittstellen zugrunde liegt, lässt sich (wie bereits in Abb. 3.12 unten durch die Verschachtelung der Komponenten angedeutet) zu einem Kriterium für die Substitutionalität von Komponenten verallgemeinern.

3.2.3 Substitutionalität von Komponenten

Eine Komponente X_1 lässt sich durch eine Komponente X_2 ersetzen, falls diese ein äquivalentes Angebot an Diensten realisiert und zugleich auch hinsichtlich der nachgefragten Dienste mit X_1 übereinstimmt. Sind die von den beiden Komponenten jeweils angebotenen und die nachgefragten Dienste explizit spezifiziert, lässt sich die Frage der Substitutionalität wiederum durch einen Vergleich der jeweils festgelegten Eigenschaften von angebotenen und nachgefragten Diensten beantworten [SECO und CAIRES 2000:119]. Bezogen auf die Schnittstellen der beiden Komponenten X_1 und X_2 bedeutet dies zunächst, dass X_2 nicht weniger Angebots- und nicht mehr Nachfrageschnittstellen als X_1 besitzen darf. Ferner müssen die zueinander korrespondierenden Schnittstellen von X_1 und X_2 übereinstimmen, damit die Substitutionalität besteht [SECO und CAIRES 2000:119].

Die geforderte Übereinstimmung ist bspw. gegeben, wenn die zu vergleichenden Angebots- bzw. Nachfrageschnittstellen der beiden Komponenten jeweils paarweise *identisch* sind. In diesem Fall stimmen beide Komponenten sowohl hinsichtlich der angebotenen als auch der nachgefragten Dienste vollständig miteinander überein und sind vom selben Typ [OMG 2003b:136]. Im Allgemeinen reicht es jedoch aus, dass die Komponente X_2 *konform* zur Komponente X_1 ist, die sie ersetzen soll. So kann die Komponente Lagerverwaltung in der als Anwendungsbeispiel gewählten Fallstudie durch eine Komponente ersetzt

werden, die ein spezielles Verfahren zur Lagerverwaltung nutzt (vgl. Abb. 3.11 oben). Die Substitutionalität ist deshalb auch dann gewährleistet, wenn der Typ der Komponente X_2 lediglich ein *Subtyp* des Typs der auszutauschenden Komponente X_1 ist [SECO und CAIRES 2000:119; OMG 2003b:136]. Das bedeutet wie in Abb. 3.13 graphisch dargestellt, dass die Angebotsschnittstellen P_{X2} der Komponente X_2 lediglich paarweise konform zu den korrespondierenden Angebotsschnittstellen P_{X1} der Komponente X_1 zu sein brauchen [SECO und CAIRES 2000:119]. Gleichzeitig brauchen auch die Nachfrageschnittstellen Ω_{X1} der Komponente X_1 lediglich paarweise konform zu den korrespondierenden Nachfrageschnittstellen Ω_{X2} der Komponente X_2 zu sein [SECO und CAIRES 2000:119]:

$$\begin{aligned} \text{Substitution:} \quad \forall\, X_1, X_2 .\ X_2 \text{ substitute } X_1 \ &\Leftrightarrow\ X_2 \leq_c X_1 \\ &\Leftrightarrow\ (\forall\, I_p \in P_{X1} .\ \exists\, I_q \in P_{X2} .\ I_q \leq_c I_p) \wedge \\ &\quad\ (\forall\, I_w \in \Omega_{X2} .\ \exists\, I_v \in \Omega_{X1} .\ I_v \leq_c I_w) \end{aligned} \qquad (3.17)$$

Die in (3.17) geforderte Konformität zwischen den Schnittstellen besteht, falls die Angebotsschnittstellen der Komponente X_2 *Spezialisierungen* der zu ihnen korrespondierenden Angebotsschnittstellen von Komponente X_1 und die Nachfrageschnittstellen der Komponente X_2 *Generalisierungen* der zu ihnen korrespondierenden Nachfrageschnittstellen von Komponente X_1 darstellen [SECO und CAIRES 2000:119]. Gemäß (3.17) darf die Komponente X_2 verglichen mit Komponente X_1 zudem über eine höhere Zahl an Angebotsschnittstellen [SECO und CAIRES 2000:119; SZYPERSKI, et al. 2002:89] sowie eine geringere Zahl an Nachfrageschnittstellen verfügen [SECO und CAIRES 2000:119], ohne dass die Substitutionalität hierdurch beeinträchtigt wird.

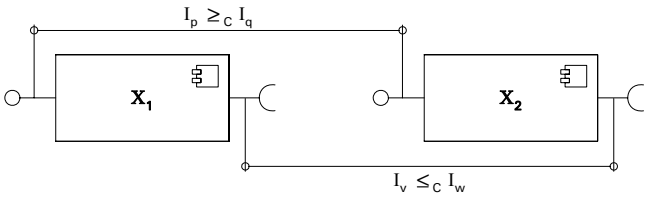


Abb. 3.13: Variabilität der Schnittstellen unter Wahrung der Substitutionalität von Komponenten.

Während das Vorhandensein weiterer Angebotsschnittstellen beim Austausch von Komponenten für das Anwendungssystem ohne Bedeutung und deshalb unkritisch ist, verändert sich durch den Wegfall von Nachfrageschnittstellen jedoch seine Ablaufstruktur. So führt das Wegfallen von Nachfrageschnittstellen dazu, dass zuvor aufgerufene Dienste von anderen Komponenten nach dem Austausch nicht mehr genutzt werden. Im günstigsten Fall können dabei weitere Komponenten überflüssig und aus dem Anwendungssystem entfernt werden. Wurden die an den Nachfrageschnittstellen aufgeführten Dienste dagegen durch

eine Architekturentscheidung bewusst einer anderen (aufzurufenden) Komponente zugewiesen, würde die Architektur des Anwendungssystems durch den Wegfall dieser Nachfrageschnittstellen beim Austausch von Komponenten ggf. in unzulässiger Weise kompromittiert. Deshalb ist neben der Substitutionalität, die als notwendige Bedingung zunächst angibt, ob eine Komponente X_2 in der gleichen Umgebung wie eine Komponente X_1 funktionieren und sie demnach prinzipiell ersetzen *kann*, stets auch die Verträglichkeit des geplanten Austauschs mit der Anwendungssystemarchitektur zu untersuchen.

3.2.4 Konformität von Schnittstellen

Sowohl bei der Beantwortung der Frage nach der Kompositionalität als auch der Frage nach der Substitutionalität von Komponenten steht vor allem die Untersuchung der *Konformität von Komponentenschnittstellen* im Mittelpunkt. Sie bildet deshalb den zentralen Bestandteil von Komponententypsystemen und wird durch ihre jeweiligen Typkalküle algorithmisch unterstützt [SECO und CAIRES 2000:119-121]. Im Allgemeinen gilt dabei, dass eine Schnittstelle I_2 konform zu einer Schnittstelle I_1 ist, wenn sie entweder identisch mit I_1 ist oder eine *Spezialisierung* von I_1 darstellt [SZYPERSKI, et al. 2002:89]. Bei der Untersuchung ist deshalb zu prüfen, ob die von I_2 zusammengefassten Dienste entweder mit denen von I_1 identisch sind oder Spezialfälle der Dienste von I_1 darstellen:

Schnittstellenkonformität: $\forall\, I_1, I_2 \,.\, I_2 \leq_c I_1 \Leftrightarrow \forall\, \Delta_p \in I_1 \,.\, \exists\, \Delta_q \in I_2 \,.\, \Delta_q \leq_c \Delta_p$ (3.18)

Darüber hinaus darf I_2 nach (3.18) im Allgemeinen zusätzliche Dienste beinhalten und die Menge der Dienste von I_1 somit erweitern, ohne dass hierdurch die Konformität beeinträchtigt wird [MEYER 1997:499; SZYPERSKI, et al. 2002:89]. Bei der Untersuchung ist ferner zu berücksichtigen, dass sich die Spezialisierung eines Dienstes nicht nur auf seine *Funktionalität*, sondern auch die *Eigenschaften seiner Ein- und Ausgaben* auswirken kann, falls dies durch die Komponententechnologie unterstützt wird [MEYER 1997:595-604; SZYPERSKI, et al. 2002:90-92]. So kann ein spezialisierter Dienst der Schnittstelle I_2 grundsätzlich einen Subtyp der Ausgabe erzeugen, die vom korrespondieren Dienst der Schnittstelle I_1 zurückgegeben wird (Prinzip der *Kovarianz*) [MEYER 1997:598; SZYPERSKI, et al. 2002:90]. Gleichzeitig kann ein spezialisierter Dienst der Schnittstelle I_2 den Typ der Eingabewerte verglichen mit dem korrespondierenden Dienst von Schnittstelle I_1 verallgemeinern (Prinzip der *Kontravarianz*) [SZYPERSKI, et al. 2002:90f.]. Hieraus ergibt sich, dass der Typ von Übergabewerten, die als Input-Output-Parameter gleichzeitig zur Ein- und Ausgabe genutzt werden, grundsätzlich nicht verändert werden darf (Prinzip der *Invarianz*) [SZYPERSKI, et al. 2002:91]:

$$\text{Dienstkonformität: } \forall\, \Delta_1, \Delta_2 \,.\, \Delta_2 \leq_c \Delta_1 \Leftrightarrow \begin{array}{l} f_{\Delta 2} \leq_c f_{\Delta 1} \wedge (|E_{\Delta 2}| = |E_{\Delta 1}|) \wedge (|A_{\Delta 2}| = |A_{\Delta 1}|) \wedge \\ (\forall\, e_{w\Delta 2} \in E_{\Delta 2} \,.\, \exists\, e_{u\Delta 1} \in E_{\Delta 1} \,.\, e_{u\Delta 1} \leq_c e_{w\Delta 2}) \wedge \\ (\forall\, a_{w\Delta 2} \in E_{\Delta 2} \,.\, \exists\, a_{u\Delta 1} \in E_{\Delta 1} \,.\, a_{w\Delta 2} \leq_c a_{u\Delta 1}) \end{array} \quad (3.19)$$

Während der Implementierung lässt sich die Konformität von Schnittstellen auf zwei verschiedene Weisen erreichen. So kann bei der Definition einer Schnittstelle I_2 entweder eine Basisschnittstelle I_1 angegeben werden, die von I_2 erweitert und ggf. in den zuvor genannten Grenzen redefiniert wird [SZYPERSKI, et al. 2002:89f.]. Dabei wird die Schnittstellenvererbung (vgl. Abschnitt 2.1.3) zur Erzeugung eines Subtyps im Rahmen einer *expliziten Definition* genutzt, deren Einhaltung sich bei der Typprüfung effizient auswerten lässt [SZYPERSKI, et al. 2002:94]. Alternativ können während der Definition einer Schnittstelle I_2 die Dienstvereinbarungen der Schnittstelle I_1 jedoch auch wiederholt und dabei ggf. in den zuvor genannten Grenzen verändert werden. In diesem Fall ist bei der Typprüfung festzustellen, ob eine Teilmenge der Dienste von I_2 den Diensten von I_1 entspricht und I_2 somit überhaupt einen Subtyp von I_1 darstellt. Dabei wird in der Regel nur die Kompatibilität der jeweiligen *Strukturen* untersucht, während die *Namen* von Diensten und Parametern ignoriert werden [SZYPERSKI, et al. 2002:94].

Prinzipiell ist jedoch zu beachten, dass eine Konformität von Schnittstellen $I_2 \leq_c I_1$ tatsächlich nur dann gegeben ist, wenn die als Bestandteil von I_2 genannten Dienste auch hinsichtlich ihrer *Funktionalität* (also der Ergebnisfunktion $f_{\Delta i}$) Spezialisierungen der Dienste von I_1 darstellen. Eine Typprüfung, die schon aus Gründen der Effizienz üblicherweise auf eine Überprüfung der implementierten Funktionalität verzichten muss, kann deshalb immer nur eine eingeschränkte Aussage treffen [SZYPERSKI, et al. 2002:90]. Die Güte der Konformitätsaussage hängt dabei wesentlich von den Informationen ab, die bei der Untersuchung über die Dienste zur Verfügung stehen [SZYPERSKI, et al. 2002:88]. Um die Funktion von Komponenten als Analyseeinheiten zu unterstützen, sollte ein Spezifikationsrahmen deshalb eine möglichst vollständige Beschreibung von Diensten anstreben, die auch deren Funktionalität einschließt. Gleichzeitig ist zu berücksichtigen, dass nur solche Informationen bei der Untersuchung verwendet werden können, die sich im Rahmen eines Kalküls effizient miteinander vergleichen lassen. Weil diese Vergleichbarkeit im Allgemeinen umso eingeschränkter möglich ist, je ausdrucksstärker die Informationen sind, sollte der Spezifikationsrahmen grundsätzlich ein Gleichgewicht zwischen Ausdrucksmächtigkeit und statischer Auswertbarkeit anstreben.

Anzumerken bleibt zum einen, dass aufgrund der zuvor gemachten Äußerungen vor allem rein strukturbasierte Verfahren, die selbst die Namen von Diensten und Übergabewerten ignorieren, nur bedingt zur Typprüfung geeignet sind. Häufig stellen nämlich gerade die

von ihnen nicht überprüften Namen die einzig auswertbaren Verweise auf die Funktionalität des Dienstes dar [SZYPERSKI, et al. 2002:94]. Zum anderen hat sich gezeigt, dass bei der Untersuchung der Kompositionalität und Substitutionalität neben der Konformität der Schnittstellen vor allem auch das Verhalten der Komponenten bei Interaktionen zu berücksichtigen ist (Prinzip der *verhaltensmäßigen Konformität*) [LISKOV und WING 1994]. Einige Typsysteme streben deshalb an, auch die Konformität der durch die dynamische Modellperspektive beschriebenen Interaktionsprotokolle in die Untersuchung der Kompositionalität bzw. Substitutionalität einzubeziehen [NIERSTRASZ 1993; LEE und XIONG 2001].

3.3 Software-Verträge

Um die Eigenschaften der von den Komponenten angebotenen bzw. nachgefragten Dienste zu beschreiben, lässt sich vor allem das Konzept des *Software-Vertrags* verwenden. Software-Verträge wurden zunächst mit dem Prinzip des „Programming by Contract“ [MEYER 1988:115-123] zur Unterstützung der Programmierung eingeführt, das in der Folgezeit unter dem Stichwort „Design by Contract“ [MEYER 1992; MEYER 1997:331-406] zu einem umfassenden Entwurfsprinzip der Anwendungsentwicklung ausgebaut wurde. Allgemein lässt sich mit Software-Verträgen das Verhältnis von miteinander interagierenden Anwendungssystemteilen beschreiben, indem ihre jeweiligen Rechte und Verpflichtungen bei der Interaktion festgeschrieben werden [MEYER 1997:342]. Als Basis für die Festschreibung dient dabei ausschließlich die Außensicht der jeweils beteiligten Anwendungsteile, so dass das *Geheimnisprinzip* durch die Formulierung eines Software-Vertrags nicht verletzt wird.

Anders als etwa Verträge über die Entwicklung von Software (vgl. [HEINRICH 1999:267-274]) stellen Software-Verträge zwar keine Verträge im juristischen Sinne dar, wie sie etwa zwischen natürlichen oder juristischen Personen geschlossen werden. Nichtsdestotrotz entfalten die vereinbarten Software-Verträge eine bedeutende normative Wirkung, die sich sowohl auf die Komponenten- als auch die Anwendungsentwicklung erstreckt. Darüber hinaus lassen sich durch die Nutzung von Software-Verträgen vor allem die bislang noch nicht berücksichtigten allgemeinen Funktionen unterstützen, die Komponenten als Strukturierungseinheiten während der Anwendungsentwicklung besitzen. Dabei unterstützen Software-Verträge gleichermaßen die Funktionen von Komponenten als *Entwurfs- und Kompositionseinheiten* (vgl. Abschnitt 2.1.1).

Für die Komponentenentwicklung stellen die durch Software-Verträge getroffenen Vereinbarungen verbindliche *Entwicklungsvorgaben* dar, die beim Entwurf der Innensicht und der anschließenden Realisierung von Komponenten einzuhalten sind. Bei der Realisierung

ist deshalb zum einen dafür zu sorgen, dass die zu entwickelnde Komponente ihren vertraglich zugesicherten Verpflichtungen nachkommt [MEYER 1997:342]. Zum anderen darf bei der Realisierung der Komponente jedoch davon ausgegangen werden, dass die im Vertrag genannten Voraussetzungen für eine erfolgreiche Interaktion durch potenzielle Interaktionspartner in derselben Weise eingehalten werden [MEYER 1997:343]. Die explizite Festlegung von Rechten und Verpflichtungen im Rahmen eines Software-Vertrags unterstützt somit die unabhängige Entwicklung von Komponenten und trägt dazu bei, dass sie während des Entwicklungsprozesses wie in Abschnitt 2.1.1 gefordert als eigenständige *Entwurfseinheiten* behandelt werden können.

Bei der Anwendungsentwicklung geben Software-Verträge vor allem Aufschluss über die jeweiligen Anforderungen und Leistungen der einzelnen Komponenten. Sie dienen somit als eine wichtige Informationsquelle für die Überprüfung der Kompositionalität bzw. Substitutionalität von Komponenten [MEYER 1997:389]. Da sich die jeweils aufgeführten Voraussetzungen für eine erfolgreiche Interaktion dabei stets an einen abstrakten Interaktionspartner richten, unterstützt das Konzept des Software-Vertrags die Kombinierbarkeit mit einer möglichst hohen Zahl an verschiedenen Komponenten. So kann insbesondere darauf verzichtet werden, bestimmte Komponenten als Interaktionspartner zu nennen und damit die Kombinierbarkeit unnötig einzuschränken. Die damit geförderte Wiederverwendbarkeit von Komponenten in Kontexten, die zur Zeit ihrer Entwicklung nicht vorhergesehen wurden, trägt wesentlich dazu bei, dass Komponenten als flexible *Kompositionseinheiten* funktionieren können.

Bezogen auf die komponentenorientierte Entwicklung von Anwendungssystemen lassen sich grundsätzlich zwei Arten von Software-Verträgen voneinander unterscheiden, die auf unterschiedliche Weise zur Beschreibung des Verhältnisses von miteinander interagierenden Komponenten beitragen [REUSSNER und SCHMIDT 2002; OVERHAGE 2004:6f.]. Diese verschiedenen Software-Verträge werden in der Literatur auch als Dienst- bzw. Komponentenverträge bezeichnet, um sie inhaltlich voneinander abzugrenzen.

3.3.1 Dienstverträge

Dienstverträge wurden als Bestandteil des vertragsbasierten Entwurfsprinzips (Design by Contract [MEYER 1992; MEYER 1997:331-406]) in die Anwendungsentwicklung eingeführt und haben vor allem im Bereich der objektorientierten Entwicklung eine wichtige Bedeutung erlangt. Mit ihnen lassen sich *Bedingungen* beschreiben, die während der *Laufzeit* zur erfolgreichen Inanspruchnahme eines Dienstes durch den Dienstnachfrager und den

Dienstanbieter einzuhalten sind. Durch einen Dienstvertrag wird zunächst die Signatur eines Dienstes festgelegt [SZYPERSKI, et al. 2002:89]. Diese besteht aus dem Namen des Dienstes, der ggf. durch eine Liste von typisierten Übergabeparametern sowie eine typisierte Rückgabe ergänzt werden kann. Darüber hinaus werden durch einen Dienstvertrag meist weitere *Vor- und Nachbedingungen* festgelegt, die für den Dienstanbieter und den Dienstnachfrager bindend sind [MEYER 1997:334-337]:

> Definition 3.25 : Dienstvertrag
>
> Ein Dienstvertrag DV_{Δ} beschreibt eine Menge von Vorbedingungen V_{Δ}, die vor Inanspruchnahme eines Dienstes Δ vom Dienstnachfrager zu erfüllen sind, sowie eine Menge von Nachbedingungen N_{Δ}, die nach einer korrekten Inanspruchnahme durch den Dienstanbieter gewährleistet werden.

Die *Vorbedingungen* beschreiben die Voraussetzungen, die für eine erfolgreiche Inanspruchnahme eines Dienstes erfüllt sein müssen [MEYER 1997:340] und vor dem Aufruf eines Dienstes durch den Dienstnachfrager zu schaffen sind [MEYER 1997:342]. Während eine Vorbedingung für den Dienstnachfrager somit eine Verpflichtung darstellt, darf der Dienstanbieter auf deren Einhaltung vertrauen (vgl. Abb. 3.14). Im Gegensatz dazu beschreiben die *Nachbedingungen* die *Garantien*, die bei der korrekten Inanspruchnahme eines Dienstes gewährleistet werden [MEYER 1997:340]. Dabei ist eine korrekte Inanspruchnahme immer dann gegeben, wenn bei einem Dienstaufruf alle Vorbedingungen durch den Dienstnachfrager eingehalten wurden. Die festgelegten Garantien sind durch den Dienstanbieter zu erfüllen und stellen für ihn somit Verpflichtungen dar [MEYER 1997:342]. Der Dienstnachfrager darf hingegen auf die Einhaltung der Garantien durch den Dienstanbieter vertrauen (vgl. Abb. 3.14).

buche	**Verpflichtungen (Aufwand)**	**Garantien (Ertrag)**
Dienstnachfrager (Client)	**Vorbedingungen einhalten:** Der Betrag einer Bestandsveränderung muss größer als 0 und darf bei einer Minderung nicht größer als der aktuelle Bestand sein	**Nachbedingungen gelten:** Der Lagerbestand wird um den genannten Betrag verändert und ist danach größer oder gleich 0
Dienstanbieter (Server)	**Nachbedingungen einhalten:** Der Lagerbestand ist um den genannten Betrag zu verändern	**Vorbedingungen gelten:** Der Betrag der Bestandsveränderung braucht nicht auf Gültigkeit überprüft zu werden

Abb. 3.14: Verpflichtungen und Garantien beim Dienstvertrag (in Anlehnung an [MEYER 1997:342]).

Üblicherweise werden die Vor- und Nachbedingungen eines Dienstvertrags unter Nutzung einer formalen Spezifikationssprache als *prädikatenlogische Ausdrücke* formuliert und

legen dabei jeweils einen bestimmten Zustand fest, der vor der Nutzung eines Dienstes bzw. im Anschluss an seine Nutzung zu gelten hat [MEYER 1997:337]. Zur Beschreibung dieser Zustände lassen sich verschiedene Spezifikationssprachen einsetzen, deren Ausdrucksmächtigkeit von einfachen wahrheitslogischen Aussagen bis hin zu komplexen Aussagen der Prädikatenlogik erster Stufe reichen. Eine Gegenüberstellung verschiedener Spezifikationssprachen und ihrer jeweiligen Eigenschaften findet sich bei [MEYER 1997:399f.; SIEDERSLEBEN 2004:128-130].

```
context IBestandsvwt::buche(artikelnr:String, auftragsnr:String, v:Veraenderung, menge:Integer):Bestand
  pre : menge > 0
  pre : if v = Veraenderung::Warenausgang
          then
            menge <= aktuellerBestand(artikelnr)
          else
            true
        endif
  post: result = if v = Veraenderung::Warenausgang
                   then
                     aktuellerBestand@pre(artikelnr) - menge
                   else
                     aktuellerBestand@pre(artikelnr) + menge
                 endif
  post: result >= 0
```

Abb. 3.15: Dienstvertrag zur Dokumentation von Veränderungen im Lagerbestand.

Abb. 3.15 zeigt beispielhaft einen Dienstvertrag, der unter Verwendung der standardisierten Object Constraint Language (OCL [OMG 2003a]) beschrieben wurde und die Bedingungen zur erfolgreichen Inanspruchnahme eines Dienstes beschreibt, mit dem sich die mengenmäßige Veränderung des Lagerbestands von Artikeln dokumentieren lässt. Der entsprechende Dienst buche ist Bestandteil der Schnittstelle IBestandsvwt, einer Angebotsschnittstelle der in Abschnitt 3.1.7 beschriebenen Komponente Lagermanagement. Dabei besagen die in der Abbildung dargestellten Vorbedingungen zunächst, dass der Betrag der zu dokumentierenden mengenmäßigen Veränderung des Lagerbestands stets größer als Null sein muss und im Falle einer Minderung (also einem Warenausgang) nicht größer als der aktuelle Lagerbestand sein darf. Im Gegenzug wird garantiert, dass der Lagerbestand nach der Ausführung des Dienstes um den beim Aufruf genannten Betrag erniedrigt bzw. erhöht wurde und größer oder gleich Null ist.

In Abhängigkeit von der eingesetzten Spezifikationssprache und den formulierten Zusicherungen lässt sich zur Laufzeit zumindest prinzipiell automatisch überprüfen, ob die durch die Vor- und Nachbedingungen jeweils beschriebenen Zustände während der Interaktion tatsächlich auftreten oder ob stattdessen eine Vertragsverletzung vorliegt [MEYER 1997:392-394]. Grundsätzlich problematisch ist hingegen die automatisierte *statische Überprüfung* von Dienstverträgen während der Entwicklungszeit. So lässt sich während der Komponentenentwicklung typischerweise nicht bestätigen, dass eine Komponentenimple-

mentierung in der Rolle als Dienstnehmer stets die Vorbedingungen bzw. in der Rolle als Dienstgeber stets die Nachbedingungen der spezifizierten Dienstverträge erfüllt. Diese Forderungen ließen sich nur durch einen mathematischen Programmbeweis überprüfen, der mit den heute vorhandenen Methoden im Allgemeinen jedoch nicht durchführbar ist [MEYER 1997:398, 578].

Aus den gleichen Gründen erweist sich auch die Überprüfung der Einhaltung von Dienstverträgen bei der Komposition von Komponenten als unmöglich. Bei der Komposition von Komponenten brauchen die Dienstverträge, die als Teil der zu verbindenden Angebots- und Nachfrageschnittstellen spezifiziert wurden, nicht exakt überein zu stimmen, sondern müssen lediglich *konform* zueinander sein (vgl. Abschnitt 3.2.2). Bezogen auf die in den Schnittstellen enthaltenen Dienstverträge bedeutet dies, dass der Dienstanbieter weniger und schwächere Vorbedingungen in Kauf nehmen sowie gleichzeitig mehr und stärkere Nachbedingungen garantieren kann, als im Dienstvertrag des Dienstnachfragers festgelegt ist [MEYER 1997:573]. Vor der Komposition ist deshalb die Konformität der als logische Aussagen beschriebenen Zustände zu überprüfen. Diese ist gegeben, wenn die Vorbedingungen des Dienstanbieters aus den Vorbedingungen des Dienstnachfragers und die Nachbedingungen des Dienstnachfragers aus den Nachbedingungen des Dienstanbieters jeweils logisch folgen [MEYER 1997:573]:

$$\text{Vertragskonformität: } \forall\, DV_1, DV_2 \,.\, DV_2 \leq_c DV_1 \Leftrightarrow (\forall\, V_q \in DV_2 \,.\, \exists\, V_p \in DV_1 \,.\, V_p \Rightarrow V_q) \wedge (\forall\, N_u \in DV_1 \,.\, \exists\, N_w \in DV_2 \,.\, N_w \Rightarrow N_u) \quad (3.20)$$

Als aussagenlogische Theoreme lassen sich jedoch auch die in (3.20) formulierten Aussagen grundsätzlich nicht in effizienter Weise automatisch überprüfen [MEYER 1997:578; SZYPERSKI, et al. 2002:88]. Ungeachtet der Einschränkungen, die sich hinsichtlich einer automatisierten statischen Überprüfung zur Entwicklungszeit ergeben, haben sich Dienstverträge in der Praxis vor allem als ein adäquates Mittel erwiesen, um die Anforderungen und Garantien von Diensten beim *Entwurf* in *präziser Form* zu dokumentieren. Sie besitzen deshalb auch für die Spezifikation von Komponenten eine wesentliche Bedeutung [D'SOUZA und WILLS 1999:86-92, 600-602; CHEESMAN und DANIELS 2001:125-128; SIEDERSLEBEN 2004:116f.].

3.3.2 Komponentenverträge

Zur Unterstützung der komponentenorientierten Anwendungsentwicklung leistet die Spezifikation von Dienstverträgen einen wesentlichen Beitrag. Alleine ist sie jedoch nicht ausreichend, da durch sie ausschließlich diejenigen Bedingungen der Inanspruchnahme von

Diensten beschrieben werden, die sich auf die Laufzeit beziehen. Damit Komponenten als eigenständige Kompositionseinheiten [SZYPERSKI, et al. 2002:143] verwendet werden können, sind jedoch auch die zur *Kompositionszeit* zu erfüllenden Bedingungen zu beschreiben, unter denen eine Komponente überhaupt erst Dienste mit bestimmten Eigenschaften für eine spätere Benutzung durch andere Komponenten anbietet. Eine solche Beschreibung lässt sich durch die Formulierung eines Komponentenvertrags erreichen. Dieser beschreibt zunächst die unveränderlichen charakteristischen Merkmale einer Komponente, die durch ihre Realisierung zur Kompositionszeit festgelegt sind. Zu diesen Merkmalen zählen intrinsische Qualitätseigenschaften wie die Verwendbarkeit, die Wartbarkeit sowie die Portabilität. Neben den statischen Komponenteneigenschaften werden mit einem Komponentenvertrag vor allem die von der Komponente angebotenen und nachgefragten Dienste und ihre Eigenschaften festgelegt. Dies schließt die Definition der *Angebots- und Nachfrageschnittstellen* der Komponente als wesentlichen Bestandteil ein [REUSSNER und SCHMIDT 2002; OMG 2003b:137; OVERHAGE 2004:7]:

> Definition 3.26: Komponentenvertrag
>
> Ein Komponentenvertrag KV_X beschreibt die von einer Komponente X nachgefragte Menge von Diensten N_X, die bei der Komposition einer Komponente X durch die Umgebung bereitzustellen sind, sowie die Menge angebotener Dienste A_X, die nach einer korrekten Komposition durch die Komponente X bereitgestellt werden.

Die Nachfrageschnittstellen legen die von einer Komponente benutzten Dienste fest, die von anderen Komponenten bei der Komposition in der vom Vertrag beschriebenen Weise bereitzustellen sind [OMG 2003b:137]. Aus der Sicht der im Vertrag spezifizierten Komponente stellen die an den Nachfrageschnittstellen genannten Dienste und ihre jeweiligen Eigenschaften somit Voraussetzungen (Vorbedingungen) dar, die von der *Umgebung* der Komponente zu erfüllen sind, bevor sie ihrerseits Dienste mit vertraglich festgelegten Eigenschaften anbietet. Während die in einem Vertrag enthaltenen Nachfrageschnittstellen für die Umgebung folglich *Verpflichtungen* darstellen, darf die Komponente auf die Bereitstellung der geforderten Dienste vertrauen (vgl. Abb. 3.16). Im Gegenzug bietet die Komponente unter der Voraussetzung, dass alle von ihr geforderten Dienste in der festgelegten Weise bereitgestellt wurden, die in den Angebotsschnittstellen beschriebenen Dienste für die Benutzung durch die Umwelt an [OMG 2003b:137]. Aus Sicht der Umwelt stellen die bereitgestellten Dienste und ihre jeweils vertraglich festgelegten Eigenschaften *Garantien* (Nachbedingungen) dar, die bei Erfüllung der Voraussetzungen durch die Komponente gegeben werden. Die im Vertrag festgelegten Garantien sind also durch die Komponente zu erfüllen und stellen für sie folglich Verpflichtungen dar, während die Umwelt auf deren Einhaltung vertrauen darf (vgl. Abb. 3.16).

Lagermanagement	Verpflichtungen (Aufwand)	Garantien (Ertrag)
Umwelt	**Vorbedingungen einhalten:** Es sind Dienste zur Artikelverwaltung und Datenhaltung bereitzustellen	**Nachbedingungen gelten:** Es werden Dienste zur Lagerverwaltung, Logistik und Bestandsverwaltung bereitgestellt
Komponente	**Nachbedingungen einhalten:** Es sind Dienste zur Lagerverwaltung, Logistik und Bestandsverwaltung bereitzustellen	**Vorbedingungen gelten:** Es sind Dienste zur Artikelverwaltung und Datenhaltung verfügbar

Abb. 3.16: Verpflichtungen und Garantien beim Komponentenvertrag.

Die von einer Komponente nachgefragten und angebotenen Dienste lassen sich mit einem Komponentenvertrag auf mehreren *Abstraktionsebenen* beschreiben, die jeweils unterschiedliche Eigenschaften der zunächst als Schnittstellen aufgeführten Dienste betrachten und in die Beschreibung einbeziehen [HAN 1998; BEUGNARD, et al. 1999:38f.; SZYPERSKI, et al. 2002:55-57]. Prinzipiell lassen sich dabei auch die spezifizierten Dienstverträge als Bestandteile eines Komponentenvertrags auffassen [BEUGNARD, et al. 1999:39]. Ähnlich wie schon bei Dienstverträgen können zur Beschreibung der Eigenschaften von Diensten auf einer Abstraktionsebene verschiedene Spezifikationssprachen mit jeweils abweichender Ausdrucksmächtigkeit herangezogen werden. Im Gegensatz zur Beschreibung von Dienstverträgen sind bei der Spezifikation von Komponentenverträgen jedoch typischerweise *mehrere* Spezifikationssprachen zu nutzen, um die verschiedenen Abstraktionsebenen abzudecken [BEUGNARD, et al. 1999:38-40].

```
context Lagermanagement
  require: IDatenbank
  require: IArtikelvwt
  provide: ILagervwt
  provide: ILogistik
  provide: IBestandsvwt
```

Abb. 3.17: Komponentenvertrag für die Komponente Lagermanagement (vereinfacht).

Abb. 3.17 zeigt den Komponentenvertrag der Komponente Lagermanagement in stark vereinfachter Darstellung, die lediglich die Angebots- und Nachfrageschnittstellen benennt und auf eine nähere Beschreibung der Eigenschaften von Diensten gänzlich verzichtet. Dabei besagen die in der Abbildung dargestellten Vorbedingungen, dass zunächst Dienste zur Artikelverwaltung sowie zur Datenhaltung bereitzustellen sind, damit die Komponente ihrerseits Dienste zur Benutzung anbieten kann. Im Gegenzug wird garantiert, dass für die Benutzung durch andere Komponenten Dienste angeboten werden, mit denen sich die Lagerverwaltung, der Wareneingang sowie der Warenausgang und die Bestandsverwaltung durchführen lassen (zur näheren Beschreibung vgl. Abschnitt 2.4).

Zur Kompositionszeit ist dann zu überprüfen, ob die von einer Komponente angebotenen Dienste für das zu entwickelnde Anwendungssystem geeignet sind und die in Form von nachgefragten Diensten beschriebenen Voraussetzungen durch die Umgebung, d.h. die anderen Komponenten der Komposition erfüllt werden. Anderenfalls liegt eine Vertragsverletzung vor, die das Funktionieren der Komponente verhindert. Damit diese Untersuchung automatisiert werden kann, müssen die Beschreibungen der Komponentenverträge *statisch vergleichbar* sein. Bei der Auswahl von Spezifikationssprachen ist deshalb auf die statische Vergleichbarkeit zu achten und die Ausdrucksmächtigkeit von Komponentenverträgen ggf. einzuschränken. Eine vollständige statische Vergleichbarkeit von Komponentenverträgen stellt allerdings schon bei der Einbeziehung der nicht statisch überprüfbaren Dienstverträge ein Ideal dar, das sich in der Praxis nur näherungsweise erreichen lässt.

3.3.3 Parametrisierte Komponentenverträge

Vielfach hängt die Funktionsfähigkeit der von einer Komponente angebotenen Dienste lediglich davon ab, dass *ein Teil* der durch die Nachfrageschnittstellen beschriebenen Dienste in der vertraglich geforderten Weise von der Umwelt bereitgestellt wird. Damit ist eine Komponente ggf. auch dann noch in der Lage, eine Teilmenge ihrer Dienste anzubieten, wenn bei der Komposition nicht sämtliche der nachgefragten Dienste durch die anderen Komponenten einer Komposition bereitgestellt werden können. Um der Funktion von Komponenten als flexible Kompositionseinheiten, die sich in möglichst vielen Anwendungsentwicklungsprojekten verwenden lassen, gerecht zu werden, ist der Komponentenvertrag deshalb um eine Beschreibung der *Abhängigkeiten* zu ergänzen, die *zwischen einzelnen angebotenen und nachgefragten Diensten* und damit zwischen den einzelnen Angebots- und Nachfrageschnittstellen der Komponente existieren. Durch die Hinzufügung der Abhängigkeiten zwischen den Diensten in die vertragliche Spezifikation entsteht ein *parametrisierter Komponentenvertrag*, mit dem sich die Bereitstellung von Angebotsschnittstellen in Beziehung zu den jeweils durch die Umwelt bereitzustellenden Nachfrageschnittstellen setzen lässt [REUSSNER 2001:60f., 70]:

> Definition 3.27: Parametrisierter Komponentenvertrag
>
> Ein Komponentenvertrag PKV_X beschreibt die die von einer Komponente X nachgefragte Menge von Diensten N_X, die von einer Komponente X angebotene Menge von Diensten A_X, sowie die zwischen den angebotenen und nachgefragten Diensten bestehenden Abhängigkeiten, welche durch eine umkehrbare Funktion $\wp : D \rightarrow \wp(D)$ dargestellt werden. Bei einem nachbedingungsparametrisierten Komponentenvertrag gilt $D := A_X$ und $\wp(D) := N_X$, während bei einem vorbedingungsparametrierten Komponentenvertrag $D := N_X$ und $\wp(D) := A_X$ gilt.

Abb. 3.18 zeigt beispielhaft die Abhängigkeiten, die zwischen den Angebots- und Nachfrageschnittstellen einer Komponente bestehen und sich aus der Abbildung $\wp$ des Komponentenvertrags ableiten lassen. Die in der Abbildung graphisch dargestellten Abhängigkeiten lassen sich in zweierlei Hinsicht auswerten. Sind die in einem bestimmten Anwendungskontext von einer Komponente zu verwendenden Angebotsschnittstellen bekannt, lässt sich durch eine *nachbedingungsparametrisierte Auswertung* [REUSSNER 2001:61] ihres Komponentenvertrags ermitteln, welche Schnittstellen bei der Komposition durch die Umgebung bereitzustellen sind. Sind in einem Entwicklungsprojekt hingegen die von der Umgebung bereitgestellten Schnittstellen vorgegeben, so lässt sich durch eine *vorbedingungsparametrisierte Auswertung* [REUSSNER 2001:61, 71] des Komponentenvertrags ermitteln, welche Angebotsschnittstellen einer Komponente im Rahmen der Komposition nutzbar sind. Durch die beiden Auswertungsmöglichkeiten lässt sich bei der Strukturierung eines Anwendungssystems im Rahmen des Systementwurfs (vgl. Abschnitt 2.2.2) gleichermaßen die Vorgehensweise nach dem Top-Down Prinzip, bei dem die nachbedingungsparametrisierte Auswertung überwiegt, sowie die Vorgehensweise nach dem Bottom-Up Prinzip, bei der die vorbedingungsparametrisierte Auswertung dominiert, unterstützen.

Lagermanagement	**IArtikelvwt**	**IDatenbank**
ILagervwt		**x**
ILogistik	**x**	**x**
IBestandsvwt		**x**

Abb. 3.18: Abhängigkeiten zwischen einzelnen Angebots- und Nachfrageschnittstellen.

Bei der Spezifikation von parametrisierten Komponentenverträgen mit einer textbasierten Notation ist zunächst eine grundsätzliche Entscheidung darüber zu treffen, ob die Abhängigkeiten zwischen den von einer Komponente angebotenen und nachgefragten Diensten durch eine vorbedingungs- oder nachbedingungsparametrisierte Abbildung beschrieben werden. Eine vorbedingungsparametrisierte Abbildung erlaubt es gemäß Definition 3.27, für jede Nachfrageschnittstelle die durch sie jeweils unterstützten Angebotsschnittstellen zu ermitteln. Eine nachbedingungsparametrisierte Abbildung unterstützt hingegen für jede Angebotsschnittstelle die Ermittlung derjenigen Nachfrageschnittstellen, die jeweils von der Umgebung bereitzustellen sind. Beide Repräsentationsformen von parametrisierten Komponentenverträgen können dabei prinzipiell durch eine Invertierung der umkehrbaren Abbildung $\wp$ ineinander überführt werden [REUSSNER 2001:61].

Abb. 3.19 zeigt den parametrisierten Komponentenvertrag der Komponente Lagermanagement. Für die Darstellung wurde dabei, mit dem Vorgehen in [REUSSNER 2001] übereinstimmend, eine nachbedingungsparametrisierte Beschreibung der Abhängigkeiten zwischen angebotenen und nachgefragten Diensten gewählt. Statt der Abhängigkeiten zwischen den einzelnen Diensten sind der besseren Übersichtlichkeit halber jedoch lediglich für jede Angebotsschnittstelle die jeweils bereitzustellenden Nachfrageschnittstellen dargestellt. Zur Durchführung der Lager- sowie der Bestandsverwaltung werden demnach nur die Dienste einer Datenbank benötigt, während die Durchführung des Wareneingangs und des Warenausgangs zudem Dienste der Artikelverwaltung voraussetzt. Die Dienste der Artikelverwaltung werden dabei benötigt, um die lagerbezogenen Daten der Artikel aus den Stammdaten auslesen zu können.

```
context Lagermanagement
  ILagervwt requires IDatenbank;
  ILogistik requires IArtikelvwt and IDatenbank;
  IBestandsvwt requires IDatenbank;
```

Abb. 3.19: Parametrisierter Komponentenvertrag für die Komponente Lagermanagement (vereinfacht).

Entgegen der zuvor gezeigten Abbildungen, die zunächst hauptsächlich das Konzept des parametrisierten Komponentenvertrags verdeutlichen sollen, werden die Abhängigkeiten im Folgenden nicht als Abhängigkeiten zwischen den Angebots- und Nachfrageschnittstellen, sondern wie ursprünglich gefordert als Abhängigkeiten zwischen den dort jeweils aufgeführten *Diensten* beschrieben [REUSSNER 2001:71]. Aus einer solch feingranularen Beschreibung lassen sich die Abhängigkeiten zwischen den Schnittstellen zwar nur noch indirekt berechnen. Dennoch erweist sich diese Vorgehensweise vor allem deshalb als überlegen, da durch sie auch solche Abhängigkeiten beschrieben werden können, die sich auf die *Eigenschaften* der angebotenen und nachgefragten Dienste beziehen. So lässt sich bspw. darstellen, auf welche Weise die Qualität (Zuverlässigkeit, Performanz) eines angebotenen Dienstes von der Qualität derjenigen externen Dienste abhängt, die von der Komponente während der Ausführung aufgerufen werden [REUSSNER 2001:72].

3.4 Komponentenspezifikationen

Ausgehend von dem zu Beginn des Kapitels entwickelten Komponentenmodell, mit dem die einzelnen Elemente der Außensicht von Komponenten beschrieben wurden, und den im Anschluss daran erläuterten Konzepten zur Analyse bzw. Beschreibung von Komponenteneigenschaften bleibt zur Vervollständigung der konzeptionellen Grundlegung noch festzulegen, welche Eigenschaften der Komponentenaußensicht in die Beschreibung einzube-

ziehen und demzufolge bei der Spezifikation von Komponenten zu berücksichtigen sind. Damit der zu entwickelnde UNSCOM Spezifikationsrahmen dabei dem Kriterium der *Vollständigkeit* genügt, ist einerseits zu fordern, dass die Beschreibung der Außensicht sowohl die äußere *Struktur* (Struktursystem) als auch das *Verhalten* (Funktionssystem) der Komponente möglichst lückenlos dokumentiert (vgl. Abschnitt 3.1.6). Andererseits ergibt sich durch die ebenso zu erfüllenden Kriterien der (automatisierten) Auswertbarkeit und der Angemessenheit die Forderung nach einer möglichst präzisen, statisch auswertbaren Beschreibung von Komponenteneigenschaften, wodurch die Ausdruckskraft im Allgemeinen eingeschränkt wird [SZYPERSKI, et al. 2002:88]. Hinsichtlich des Spezifikationsumfangs resultieren aus den vom UNSCOM Spezifikationsrahmen zu erfüllenden Kriterien somit zwei grundsätzlich konträre Ziele, zwischen denen zunächst ein angemessener Ausgleich zu schaffen ist.

3.4.1 Spezifikationsumfang

Grundsätzlich ließen sich sämtliche Merkmale der Außensicht von Komponenten, die für die Komponenten- und Anwendungssystementwicklung von Bedeutung sind, im Rahmen einer umgangssprachlichen Dokumentation beschreiben. Als Spezifikationskonzept sind umgangssprachliche *Dokumentationen* folglich insbesondere in der Lage, eine möglichst *vollständige* Beschreibung der Komponentenaußensicht zu gewährleisten. Wegen der inhärent vorhandenen Vagheiten, Inkonsistenzen und Mehrdeutigkeiten ist die Umgangssprache jedoch nur sehr bedingt für eine unmissverständliche und konsistente Beschreibung der Komponentenaußensicht geeignet, die den Anforderungen des Entwicklungsprozesses gerecht wird [RAMAMOORTHY, et al. 1984:193f.]. Mit dem zuvor beschriebenen Konzept des Komponentenvertrags wird deshalb vor allem versucht, die für die Entwicklung relevanten Merkmale der Komponentenaußensicht möglichst *präzise* zu beschreiben [SZYPERSKI, et al. 2002:88]. Daraus folgt, dass formale Sprachen aufgrund ihrer eindeutig festgelegten Semantik bei der Vereinbarung von Komponentenverträgen gegenüber der Umgangssprache bevorzugt werden [SZYPERSKI, et al. 2002:88].

Durch einen Komponentenvertrag lassen sich allerdings in der Praxis nicht alle relevanten Merkmale der Komponentenaußensicht angemessen beschreiben, da eine vollständige Formalisierung häufig als zu aufwändig und schwierig angesehen wird [SZYPERSKI, et al. 2002:88]. Mit der Verwendung von (Komponenten-) *Verträgen* als Spezifikationskonzept geht daher in der Regel zunächst eine Reduktion der Ausdruckskraft einher (vgl. Abb. 3.20). Diesem Umstand kann durch die Einführung verschiedener Vertragsebenen, mit denen sich verschiedene Merkmale der Komponentenaußensicht auch formal in möglichst

effizienter Weise beschreiben lassen, entgegengewirkt werden. Zusätzliche Einschränkungen in der Ausdruckskraft ergeben sich darüber hinaus jedoch im Hinblick auf die statische Auswertbarkeit von formalen Beschreibungen. So lassen sich mit den derzeit verfügbaren typtheoretischen Kalkülen, die auf dem Spezifikationskonzept des statisch *analysierbaren* (Komponenten-) *Typs* basieren, nicht alle Merkmale der formal beschriebenen Komponentenverträge einer automatisierten Auswertung unterziehen (vgl. Abb. 3.20). Typsysteme sind deshalb zumindest heutzutage ausschließlich in der Lage, Aussagen bezüglich eines vereinfachten Komponentenvertrags zu treffen [SZYPERSKI, et al. 2002:89, 94].

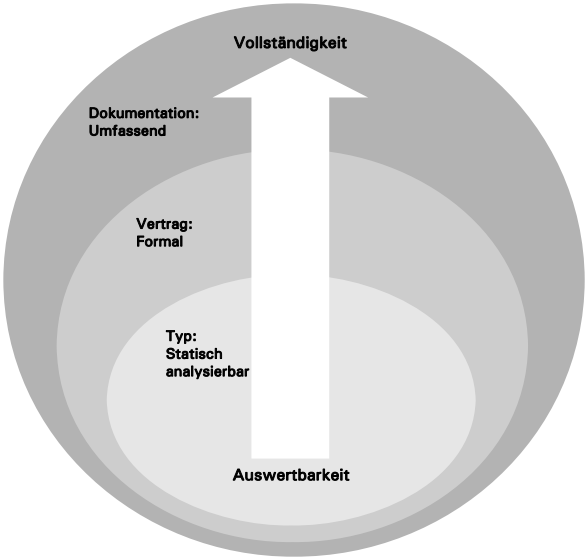


Abb. 3.20: Unterschiedliche Ausdruckskraft verschiedener Spezifikationskonzepte.

Ungeachtet des Verlusts an Ausdruckskraft gegenüber umgangssprachlichen Dokumentationen und der nur unvollständigen statischen Auswertbarkeit stellen Komponentenverträge für den UNSCOM Spezifikationsrahmen dennoch das am besten geeignete Spezifikationskonzept dar. Für die Nutzung von Komponentenverträgen spricht zunächst die methodische Unterstützung bei der Erstellung möglichst präziser, unmissverständlicher Beschreibungen, die auf dem bevorzugten Einsatz formaler Sprachen basiert. Zudem existieren bereits ausgereifte Ansätze, mit denen sich verschiedene Merkmale der Komponentenaußensicht in die formale Beschreibung von Komponentenverträgen einbeziehen lassen [HAN 1998; BEUGNARD, et al. 1999:38f.; SZYPERSKI, et al. 2002:55-57]. Durch die Einbeziehung dieser Merkmale lässt sich die Ausdruckskraft eines Komponentenvertrags dem Umfang einer umgangssprachlichen Dokumentation annähern.

Abschließend bleibt anzumerken, dass für die in den Komponentenverträgen beschriebenen Sachverhalte nach und nach auch typtheoretische Kalküle entstehen, so dass sich die statisch auswertbaren Bestandteile sukzessive erhöhen [FROLUND und KOISTINEN 1998:38-

55]. Da bei der Hinzunahme neuer Merkmale in die formale Beschreibung eines Komponentenvertrags grundsätzlich auch auf deren potenzielle statische Überprüfbarkeit zu achten ist (vgl. Abschnitt 3.3.2), bilden Komponentenverträge im Gegensatz zu umgangssprachlich formulierten Dokumentationen das besser geeignete Spezifikationskonzept für die zurzeit in der Entstehung befindlichen (Komponenten-) Typsysteme, die um zusätzliche Aspekte erweitert sind [WALLNAU 2003:65].

3.4.2 Komponenteneigenschaften

Mit der Entscheidung für den Komponentenvertrag als geeignetes Spezifikationskonzept sind nunmehr die einzelnen Merkmale der Komponentenaußensicht genauer zu benennen, die für die Komponenten- bzw. Anwendungsentwicklung von Bedeutung sind. Aus den als entwicklungsrelevant identifizierten Merkmalen ergeben sich die verschiedenen Vertragsebenen, die bei der Festlegung von Komponentenverträgen zu beschreiben sind. Um das Kriterium der *Vollständigkeit* zu erfüllen, werden zur Identifikation neben existierenden Ansätzen zur Spezifikation von Komponenten (vgl. Abschnitt 2.3.2) auch etablierte *Rahmenwerke* für den Entwurf und die Realisierung von Anwendungssystemen herangezogen, die in den vergangenen Jahrzehnten entstanden sind. Letztere sind schon wegen ihrer Ordnungsfunktion in der Lage, wichtige Hinweise auf die während der Entwicklung üblicherweise verwendeten Informationen zu geben [SCHIENMANN 1997:50f.]. Obgleich es wegen der fehlenden Möglichkeit einer analytischen Beweisführung nicht möglich ist, die Vollständigkeit einer Spezifikation in absoluter Weise zu beurteilen, lässt sich durch die Anlehnung an etablierte Rahmenwerken wie bspw. *Syntropy* [COOK und DANIELS 1994], *Catalysis* [D'SOUZA und WILLS 1999] oder *ARIS* [SCHEER 1998] sowie durch den Vergleich mit existierenden Ansätzen und zumindest eine relative, praktische Bewertung erzielen.

Als Ausgangspunkt für die Identifikation der zu spezifizierenden Merkmale der Komponentenaußensicht dienen zunächst die *Modellperspektiven* des zuvor dargestellten konzeptionellen Komponentenmodells (vgl. Abschnitte 3.1.1 und 3.1.6). Danach ist die Außensicht einer Komponente prinzipiell aus drei Sichten heraus zu beschreiben, aus deren Zusammenschau sich die vollständige Darstellung ergibt. Auffallend ist, dass die zuvor systemtheoretisch begründeten Modellperspektiven auch von zahlreichen der bislang entstandenen Rahmenwerke beim Entwurf und der Realisierung von Anwendungssystemteilen als wesentlich für die Reduktion der Komplexität angesehen werden [OLLE, et al. 1991:12f.; DAVIS 1993:33; COOK und DANIELS 1994:21; GRAHAM 1994:224; SCHIENMANN 1997:51f.; SCHEER 1998:37; D'SOUZA und WILLS 1999:43]. Unabhängig vom eingesetzten Entwick-

lungsparadigma besteht die vollständige Beschreibung eines Anwendungssystemteils somit aus folgenden Perspektiven [WILKIE 1993:349]:

- **Statische Sicht** (auch als Datensicht, Type View bzw. Informational Aspect bezeichnet). Die statische Sicht beschreibt die nach außen sichtbare *Struktur* der Komponente, d.h. die zeitlich unveränderliche Gestaltung der Komponentenaußensicht.

- **Operationale Sicht** (auch als Funktionssicht, Operational View bzw. Functional Aspect bezeichnet). Die operationale Sicht beschreibt die *Wirkungen* der Operationen, die auf der Komponente ausgeführt werden können oder von der Komponente auf ihrer jeweiligen Umgebung ausgeführt werden.

- **Dynamische Sicht** (auch als Prozesssicht, State View bzw. Interactive Aspect bezeichnet). Die dynamische Sicht beschreibt die zeitlichen Veränderungen des nach außen sichtbaren Zustands einer Komponente, die bei der Ausführung von Operationen auftretenden zeitlichen Abhängigkeiten und die daraus resultierenden *Interaktionsabläufe*.

Gelegentlich werden die operationale und dynamische Sicht auch als *Verhaltenssicht* zusammengefasst, was der Zusammenschau der operationalen und dynamischen Modellperspektive im funktionalen Systemkonzept der Systemtheorie entspricht [ROPOHL 1999:76]. Orthogonal zu den genannten Modellperspektiven werden die Anwendungssystemteile während der einzelnen Phasen des Entwicklungsprozesses (vgl. Abschnitt 2.2.2) auf unterschiedlichen *Abstraktionsebenen* betrachtet, die sich in ihrer Nähe zur informationstechnischen Umsetzung unterscheiden [SCHEER 1998:37; D'SOUZA und WILLS 1999:35f.; BECKER und SCHÜTTE 2004:69]. Diese Abstraktionsebenen geben ebenfalls wichtige Hinweise für die Identifikation der zu spezifizierenden Merkmale einer Komponentenaußensicht:

- **Fachliche Abstraktionsebene** (auch als Fachkonzept, Domain Model bzw. Business Type Model bezeichnet). Auf der fachlichen Abstraktionsebene wird das der Informationsverarbeitung zugrunde liegende Begriffsverständnis des Anwendungsbereichs geklärt. Das dabei in Form einer Ontologie rekonstruierte Begriffssystem beschreibt die *Funktionalität* der von einer Komponente angebotenen bzw. nachgefragten Dienste unabhängig von der zur Realisierung eingesetzten Technologie.

- **Logische Abstraktionsebene** (auch als Systemkonzept, Architectural Model bzw. Component External Model bezeichnet). Auf der logischen Abstraktionsebene wird die systemtechnische Beschaffenheit einer Komponente festgelegt. Die dabei festgelegten Signaturen beschreiben zusammen mit den für die Methoden vereinbarten Zusicherungen und den Interaktionsprotokollen die *Architektur* der Komponentenaußensicht.

- **Physische Abstraktionsebene** (auch als Implementierung, Implementation Model bzw. Component Internal Model bezeichnet). Die physische Abstraktionsebene beschreibt die Qualitätsmerkmale der Komponente und der von ihr angebotenen bzw. nachgefragten Dienste. Die *Qualität* ist durch die Komponentenrealisierung festgelegt, die bei der Spezifikation der Außensicht verborgen bleibt (Blackbox-Prinzip).

Durch die Zusammenschau der zueinander orthogonalen Modellperspektiven und Abstraktionsebenen lassen sich die verschiedenen Merkmale der Komponentenaußensicht identifizieren, die als Bestandteile einer Komponentenspezifikation im Rahmen eines Komponentenvertrags zu beschreiben sind (vgl. Abb. 3.21). Nach dieser Zusammenschau ist die *Funktionalität* der von der Komponente angebotenen bzw. nachgefragten Dienste aus fachlicher (technologieunabhängiger) Sicht anhand derjenigen Begriffe aus dem Anwendungsbereich zu beschreiben, die die während der Informationsverarbeitung bearbeiteten *Informationsobjekte* (Dinge), die unterstützten *Funktionen* (Handlungen) und die jeweils durchlaufenen *Prozesse* (Abläufe) repräsentieren [ORTNER 1997:84f.; SCHIENMANN 1997:111-113; SCHEER 1998:37].

	Funktionalität / Fachbegriffe (fachlich)	**Architektur / Beschaffenheit (logisch)**	**Implementierung / Qualität (physisch)**
Statische Sicht (Struktur)	Informationsobjekte (Dinge)	Signaturen (Komponente, Schnittstellen, Methoden)	Verwendbarkeit, Wartbarkeit, Portabilität
Operationale Sicht (Wirkungen)	Funktionen (Handlungen)	Zusicherungen	Funktionalität
Dynamische Sicht (Interaktionen)	Prozesse (Abläufe)	Interaktionsprotokolle	Zuverlässigkeit, Effizienz

Abb. 3.21: Klassifikation von Eigenschaften der Komponentenaußensicht.

Die Beschreibung der systemtechnischen *Architektur* der Komponentenaußensicht erfolgt zunächst durch eine Reihe von *Signaturen*. Hierunter fällt zum einen die Festlegung der Angebots- und Nachfrageschnittstellen einer Komponente als *Komponentensignatur* [SECO und CAIRES 2000:113] und zum anderen die Definition von *Datentypen* bzw. Eigenschaften (*Attributen*), die Bestandteile der *Schnittstellensignaturen* sind. Mit der Festlegung der Schnittstellensignaturen sind auch die *Signaturen der Methoden* (sowie der Ausnahmen und Ereignisse) zu definieren, die bereits einen Teil der Wirkungen beschreiben. Die Beschreibung der Wirkungen ist durch die Vor- und Nachbedingungen (*Zusicherungen*), die bei der Inanspruchnahme der definierten Methoden jeweils auftreten, zu ergänzen (vgl. Abschnitt 3.3.1). Vervollständigt wird die Beschreibung der Architektur schließlich durch die Angabe der Interaktionsprotokolle, die das technische Zusammenwirken der Komponente mit anderen Anwendungssystemteilen festlegen.

Die nach außen sichtbare *Qualität* der Komponente lässt sich durch Qualitätsmerkmale beschreiben, die vom Qualitätsmodell des ISO 9126 Standards zu insgesamt sechs Merkmalsklassen zusammengefasst werden [ISO/IEC 2001:7]. Bei der Spezifikation der Qualität sind zunächst intrinsische, kontextunabhängige (statische) Qualitätsmerkmalsklassen der Komponente zu beschreiben, zu denen die *Verwendbarkeit*, die *Wartbarkeit* und die *Portabilität* gehören [ISO/IEC 2001:7]. Ferner sind Wirkungen der Operationen zu beschreiben, die sich bei der Ausführung zusätzlich zur fachlichen Funktionalität ergeben und vom ISO Standard zur *Funktionalität* zusammengefasst werden. Zu diesen Wirkungen gehören bspw. die Sicherheit, die Persistenz und weitere Merkmale, mit denen sich auch die aspektorientierte Programmierung befasst [ISO/IEC 2001:7f.]. Abgeschlossen wird die Beschreibung der Qualität schließlich durch die Merkmalsklassen *Zuverlässigkeit* und *Effizienz*, die während der Laufzeit im Vordergrund stehen [ISO/IEC 2001:7].

Anhand des in Abb. 3.21 dargestellten Klassifikationsschemas, das die verschiedenen Merkmale der Komponentenaußensicht unter Rückgriff auf die Vorgaben etablierter Rahmenwerke unterscheidet, lassen sich somit insgesamt *neun Vertragsebenen* identifizieren, die in die Vereinbarung eines Komponentenvertrags einzubeziehen sind. Diese Vertragsebenen bilden den Kern des UNSCOM Spezifikationsrahmens und werden im vierten Kapitel im Detail betrachtet. Verglichen mit den in Abschnitt 2.3.2 vorgestellten Ansätzen zur Spezifikation von Komponenten zeigt sich zunächst, dass die durch das Klassifikationsschema identifizierten Vertragsebenen von keinem der bislang existierenden Ansätze in ausreichendem Maße unterstützt werden.

Sowohl *Catalysis* als auch *UML Components* konzentrieren sich bei der Spezifikation der Komponentenaußensicht ausschließlich auf die Beschreibung der Architektur und vernachlässigen dabei die Spezifikation der Komponentenfunktionalität und -qualität. Zwar wird das Begriffsverständnis des Anwendungsbereichs bei beiden Ansätzen im Rahmen eines Fachkonzepts (als Domain Model [D'SOUZA und WILLS 1999:35] bzw. Business Concept Model [CHEESMAN und DANIELS 2001:43f.]) näher betrachtet. Dieses wird jedoch ausschließlich zur Entwicklung der Komponentenarchitektur genutzt und nicht etwa verwendet, um die fachliche Funktionalität der Komponente zu beschreiben. Vergleichbare Defizite weisen auch die Ansätze auf, die zur Spezifikation der Komponentenaußensicht auf *Architekturbeschreibungssprachen* zurückgreifen (zur Diskussion vgl. Abschnitt 2.3.2.3). Am ehesten ließen sich die durch das Klassifikationsschema identifizierten Merkmale deshalb mit den anderen vertragsebenenorientierten Ansätzen beschreiben, die ebenfalls eine möglichst vollständige Beschreibung der Komponentenaußensicht anstreben. Da für sich genommen jedoch auch keiner dieser Ansätze in der Lage ist, sämtliche der als relevant identifizierten Merkmale bei der Spezifikation zu berücksichtigen, zeigen die vertragsebe-

nenorientierten Ansätze ebenfalls Mängel hinsichtlich des von ihnen unterstützten Spezifikationsumfangs.

Hervorzuheben ist indes die Tatsache, dass die Klassifikation sämtliche Merkmale der Komponentenaußensicht abdeckt, die von den existierenden Ansätzen insgesamt als relevant angesehen werden. Dementsprechend umfasst der UNSCOM Spezifikationsrahmen die in Abschnitt 2.3.2 betrachteten existierenden Ansätze jeweils inhaltlich und erfüllt so bereits eine für einen vereinheitlichten Ansatz wesentliche Voraussetzung.

3.4.3 Komponenten und Schnittstellen

Werden die durch das Klassifikationsschema als Vertragsebenen identifizierten Merkmale einer Komponentenaußensicht in Zusammenhang mit dem zuvor entwickelten konzeptionellen Komponentenmodell betrachtet, fällt auf, dass sich die einzelnen Merkmale auf verschiedene Modellelemente beziehen. So besteht die Komponentenaußensicht gemäß dem Metaschema des Komponentenmodells aus den verschiedenen *Schnittstellen*, den nach außen sichtbaren Attributen der *Komponente*, sowie einer *Beziehung* zwischen der Komponente und ihren Schnittstellen, die die Rolle der Schnittstelle als Angebots- bzw. Nachfrageschnittstelle festlegt (vgl. Abb. 3.22). Abschließend ist daher noch zu klären, auf welche Weise die verschiedenen Vertragsebenen einer Komponentenaußensicht den einzelnen *Modellelementen* zuzuordnen sind.

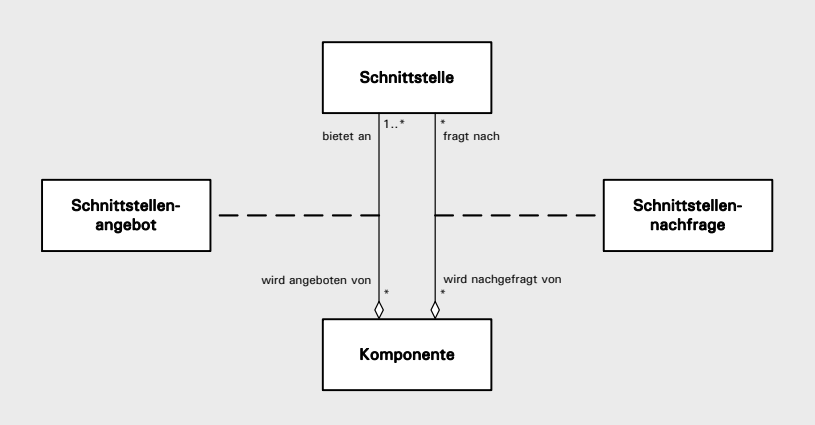


Abb. 3.22: Zusammenhang zwischen Komponenten- und Schnittstellenmerkmalen.

Zu den Merkmalen, die der Schnittstelle als Attribute zuzuordnen sind, gehören zunächst die schnittstellenspezifischen Vereinbarungen, also die *Signaturen* der einzelnen Methoden, Ereignisse und Ausnahmen. Weiterhin können der Schnittstelle die Dienstverträge (die die bei der Inanspruchnahme der einzelnen Methoden auftretenden funktionalen Wir-

kungen in Form von *Zusicherungen* beschreiben), die *qualitativen Wirkungen* von Methodenaufrufen sowie die *Interaktionsprotokolle* (die die beim Aufruf von Methoden zu berücksichtigenden Reihenfolgebedingungen beschreiben) zugeordnet werden.[11] Schließlich sind auch die Begriffe, mit denen die Funktionalität der an den Schnittstellen vereinbarten Dienste aus fachlicher Sicht beschrieben wird, der Schnittstelle zuzuordnen. In der Literatur werden die Vertragsebenen, die der Schnittstelle zuzuordnen sind, deshalb bisweilen auch als *Schnittstellenmodelle* bezeichnet [HAN 1998; REUSSNER 2001:41f.].

Zu den Merkmalen, die nach außen sichtbare Attribute der Komponente darstellen, gehören auf der anderen Seite die intrinsischen Qualitätsmerkmale, also die *Verwendbarkeit*, die *Wartbarkeit* sowie die *Portabilität*. Darüber hinaus zählen die *Abhängigkeiten*, die zwischen angebotenen und nachgefragten Diensten bestehen, zu den Merkmalen, die durch die Komponente bzw. deren jeweilige Realisierung festgelegt werden. Diese sind bei der Vereinbarung eines parametrisierten Komponentenvertrags ebenfalls in die Beschreibung einzubeziehen. Gleichermaßen ist die Einordnung der verschiedenen Schnittstellen als Angebots- bzw. Nachfrageschnittstellen, also die Definition der *Komponentensignatur*, der Komponente zuzuordnen [SZYPERSKI, et al. 2002:174]. Diese Einordnung wird im Komponentenmodell durch die zwischen der Komponente und ihren Schnittstellen existierenden Beziehungen ausgedrückt (vgl. Abb. 3.22).

Den Beziehungen (und nicht etwa den Schnittstellen) sind neben der jeweiligen Schnittstellenrolle auch die Aussagen über die Dienstgüte, d.h. die *Zuverlässigkeit* und die *Effizienz* der an der Schnittstelle jeweils aufgeführten Dienste, als Attribute zuzuordnen. Die Zuordnung der Dienstgüte zu den Beziehungen erfolgt dabei aufgrund der Tatsache, dass die Dienstgüte maßgeblich durch die Realisierung der Komponente bestimmt wird. Zudem hängt die Güte eines angebotenen Dienstes von der Güte derjenigen Dienste ab, die bei seiner Ausführung ggf. von der Komponente nachgefragt werden. Die Dienstgüte lässt sich deshalb prinzipiell nur unter Berücksichtigung der Abhängigkeiten zwischen angebotenen und nachgefragten Diensten beschreiben, die erst durch die Komponente bzw. deren Realisierung festgelegt werden. Die entsprechende Vertragsebene, die die Dienstgüte beschreibt, kann deshalb *nicht* (wie es in der Literatur gelegentlich geschieht [HAN 1998; REUSSNER 2001:41f.]) bloß als ein weiteres Schnittstellenmodell betrachtet werden.

[11] Interaktionsprotokolle, die Reihenfolgebeziehungen zwischen den Diensten mehrerer Schnittstellen beschreiben, sind dabei ggf. Ports zuzuordnen, die mehrere Schnittstellen umfassen.

4 UNSCOM Spezifikationsrahmen

Nachdem in den vorangegangenen Kapiteln die notwendigen Grundlagen gelegt wurden, ist dieses Kapitel der detaillierten Darstellung des UNSCOM Spezifikationsrahmens und der Erläuterung seiner Vorgaben für die Spezifikation der Komponentenaußensicht gewidmet. Hierzu wird in Abschnitt 4.1 zunächst der grundsätzliche *Aufbau* des UNSCOM Spezifikationsrahmens beschrieben und dabei insbesondere auf den Inhalt einer Komponentenspezifikation, die zur Repräsentation von Spezifikationen gewählten Notationen sowie auf den herzustellenden Zusammenhang der einzelnen Spezifikationsteile mit dem Entwicklungsprozess eingegangen. Aufbauend auf diesem allgemeinen Überblick werden in den restlichen Abschnitten die Vorgaben für die einzelnen Bestandteile des Spezifikationsrahmens im Detail vorgestellt.

In Abschnitt 4.2 wird die Spezifikation der *allgemeinen* und für den *kommerziellen* Gebrauch von Komponenten notwendigen *Informationen* dargestellt und in Abschnitt 4.3 um weitere Vorgaben zur *Klassierung* von Komponenten ergänzt. Im Anschluss daran wird in Abschnitt 4.4 dargestellt, auf welche Weise die *Funktionalität* von Komponenten aus fachlich-konzeptioneller Sicht zu spezifizieren ist. Abschnitt 4.5 ist dann der Beschreibung der systemtechnischen *Architektur* der Komponentenaußensicht gewidmet, die – wie schon bei den bereits existierenden Ansätzen zur Spezifikation von Komponenten – einen weiteren Schwerpunkt des UNSCOM Spezifikationsrahmens bildet. Vervollständigt wird die Spezifikation der Komponentenaußensicht schließlich durch die Beschreibung der nach außen sichtbaren *Qualitätseigenschaften*, deren Spezifikation in Abschnitt 4.6 thematisiert wird. Die Darstellung des UNSCOM Spezifikationsrahmens endet mit einer Betrachtung des Entwicklungsprozesses in Abschnitt 4.7, in deren Rahmen die schrittweise Entstehung und die Verwendung einer Komponentenspezifikation dargestellt werden. Dabei wird auf die beiden verschiedenen Vorgehensweisen bei der komponentenorientierten Anwendungsentwicklung näher eingegangen, die in Abschnitt 2.2 bereits idealtypisch unterschieden und als Top-Down bzw. Bottom-Up bezeichnet wurden.

4.1 Aufbau

Der in der Gegenstandsbestimmung aufgestellten Definition 2.3 folgend lassen sich grundsätzlich drei verschiedene Aspekte einer Spezifikation unterscheiden, die von einem Spezi-

fikationsrahmen und seinen methodischen Vorgaben gleichermaßen zu adressieren sind (vgl. Abschnitt 2.3.1). So ist zuerst der *Inhalt* einer Spezifikation durch Vorgaben darüber zu konkretisieren, *was* es an Spezifikationsteilen im Einzelnen zu beschreiben gilt. Darüber hinaus sind von einem Spezifikationsrahmen Festlegungen hinsichtlich der *Repräsentation* der Spezifikationen zu treffen. Auf den inhaltlichen Vorgaben aufbauend bestimmen diese Festlegungen, *womit* einzelne Spezifikationsteile darzustellen sind. Schließlich ist auch die *Geltung* der Spezifikationsteile festzulegen. Dabei sind vor allem Vorgaben im Hinblick auf die Erarbeitung und Begründung von Spezifikationen zu treffen, die festsetzen, *wie* deren einzelne Bestandteile zu erstellen und in den Gesamtzusammenhang einzuordnen sind. Eine solche normative Festlegung der Geltung ist eine wichtige Voraussetzung für das angestrebte gemeinsame Verständnis der Spezifikation seitens der am Entwicklungsprozess beteiligten Personen [POHL 1994:246; SCHIENMANN 1997:49f.].

Der UNSCOM Spezifikationsrahmen adressiert mit seinen Vorgaben, die die Spezifikation der Komponentenaußensicht festlegen, jeden der zuvor genannten drei grundlegenden Aspekte. Seine Vorgaben für die Spezifikation von Komponenten lassen sich deshalb grundsätzlich den nachfolgend näher betrachteten drei Bereichen *Inhalt*, *Repräsentation* bzw. *Geltung* zuordnen. Der besseren Übersicht halber werden diese Bereiche auch bei der detaillierten Betrachtung in den späteren Abschnitten dieses Kapitels voneinander getrennt.

4.1.1 Inhalt

Zur Beschreibung der Außensicht von Komponenten im Rahmen einer Komponentenspezifikation nutzt der UNSCOM Spezifikationsrahmen das zuvor erläuterte Konzept des *Komponentenvertrags*. Als Basis für die Festlegung des Inhalts von Komponentenspezifikationen dienen somit vor allem die insgesamt neun *Vertragsebenen*, die im Rahmen der konzeptionellen Grundlegung als relevante Elemente zur vollständigen Beschreibung von Komponentenverträgen identifiziert wurden (vgl. Abschnitt 3.4.2) und den Kern der inhaltlichen Vorgaben des UNSCOM Spezifikationsrahmens bilden. Zur Festlegung der Funktionalität einer Komponente aus fachlich-konzeptioneller Sicht sind demnach die bearbeiteten *Informationsobjekte*, die mit der Informationsverarbeitung unterstützten *Funktionen* und die dabei durchlaufenen *Prozesse* zu beschreiben. Die logische Architektur, die die systemtechnische Beschaffenheit der Komponente beschreibt, ist anhand der jeweils vereinbarten *Signaturen*, den *Zusicherungen* und den *Reihenfolgebedingungen* der einzelnen Methoden festzulegen. Die Qualität der Komponente ist schließlich durch eine Reihe von nicht-funktionalen Merkmalen zu beschreiben. Diese Merkmale beschreiben die *Verwendung* der Komponente in einem konkreten Anwendungskontext sowie die *Funktionsweise* und *Per-*

formanz der von ihr angebotenen bzw. nachgefragten Dienste. Ergänzt werden die genannten Vertragsebenen durch zwei weitere Ebenen, die *allgemeine* bzw. für die *kommerzielle* Nutzung von Komponenten benötigte *Informationen* erfassen und die *Klassierung* von Komponenten im Rahmen der Katalogisierung unterstützen (vgl. Abb. 4.1).

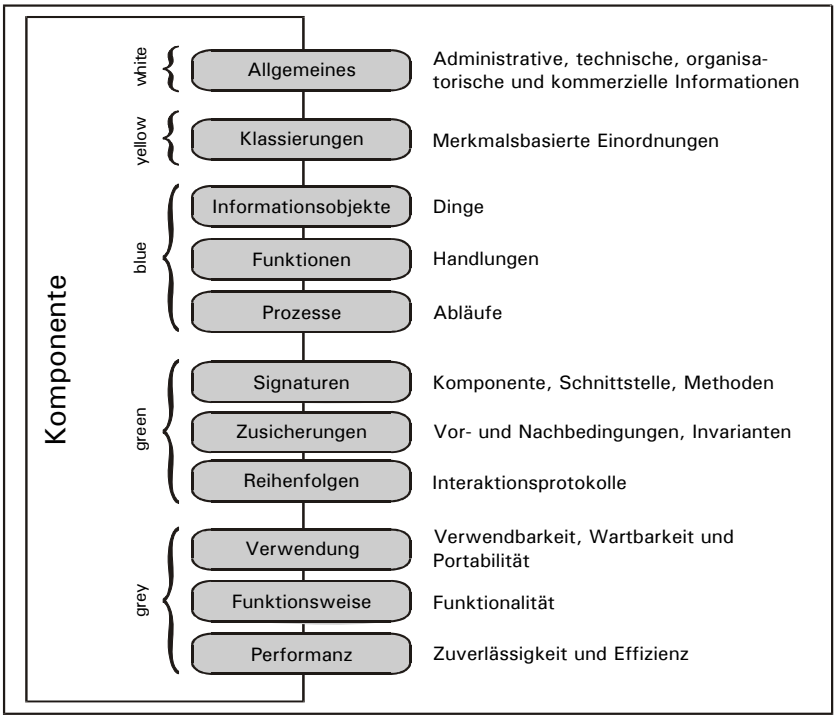


Abb. 4.1: Inhaltliche Struktur und Vertragsebenen des UNSCOM Spezifikationsrahmens.

Der besseren Übersichtlichkeit halber sind die einzelnen Vertragsebenen im UNSCOM Spezifikationsrahmen thematisch gruppiert und verschiedenen Kategorien zugeordnet. Die Kategorien werden in Anlehnung an den UDDI (Universal Description, Discovery, and Integration [CERAMI 2002:58; SZYPERSKI, et al. 2002:226-229]) Standard, einem Ansatz zur Beschreibung von XML Web-Services, durch unterschiedliche Farbgebungen voneinander abgegrenzt: Die *White Pages* beinhalten die allgemeinen und für den kommerziellen Gebrauch notwendigen Informationen, die *Yellow Pages* fassen die zur Katalogisierung vorzunehmenden Klassierungen zusammen, die *Blue Pages* beschreiben die fachliche Funktionalität der Komponente, die *Green Pages* beschreiben die Architekturmerkmale der Komponentenaußensicht und die *Grey Pages* beschreiben qualitative Komponenten- bzw. Diensteigenschaften.

Die inhaltlichen Vorgaben des UNSCOM Spezifikationsrahmens werden zu einem *Metaschema* zusammengefasst, das den Gesamtzusammenhang zwischen den einzelnen Beschreibungsteilen herstellt. Dieses Metaschema legt den Inhalt einer Komponentenspezifi-

kation *unabhängig* von den zur Repräsentation einzusetzenden Notationen fest. Es basiert auf dem Metaschema, das als Bestandteil des konzeptionellen Komponentenmodells im Rahmen der Grundlegung entwickelt wurde (vgl. Abschnitt 3.1.6) und ist damit von konkreten Komponententechnologien unabhängig.

4.1.2 Repräsentation

Bei der Repräsentation des spezifizierten Komponentenvertrags werden durch die Vorgaben des UNSCOM Spezifikationsrahmens *semiformale Spezifikationssprachen* als Darstellungsformate bevorzugt. Diese Sprachen unterscheiden sich von formalen Sprachen, deren Syntax und Semantik durch mathematisch-logische Kalküle eindeutig definiert ist, im Allgemeinen dadurch, dass die Sprachkonstrukte hinsichtlich ihrer Bedeutung entweder nicht eindeutig oder nicht in vollem Umfang festgelegt sind. Gleichwohl sind semiformale Sprachen jedoch hinreichend präzise, um eine geeignete Grundlage für die meisten Anwendungsentwicklungsprojekte der Praxis zu bilden. Somit bedeutet ihre bevorzugte Verwendung zwar eine Abweichung von der zuvor aufgestellten Forderung, dass zur Beschreibung von Software-Verträgen aus Gründen der Eindeutigkeit möglichst formale Sprachen zu verwenden sind [SZYPERSKI, et al. 2002:88]. Jedoch begünstigt – entgegen den Äußerungen von [HALL 1990], der für den ausschließlichen Einsatz formaler Methoden plädiert – gerade diese Abweichung die Anwendung des Spezifikationsrahmens in den Entwicklungsprojekten der Praxis. Wo höhere Qualitätsansprüche oder Sicherheitsanforderungen den Einsatz formaler Sprachen wie *Troll* oder *Z* erforderlich machen, können diese indes eine wichtige Ergänzung darstellen [COOK und DANIELS 1994:20].

Außerdem trägt bereits das Metaschema des UNSCOM Spezifikationsrahmens wesentlich zur inhaltlichen Klärung der darzustellenden Komponentenspezifikationen bei, die sich auf die verwendeten Sprachkonstrukte auswirkt. Durch die inhaltliche *Präzisierung* der eingesetzten Sprachen wird eine Eindeutigkeit erreicht, die die Formulierung von Komponentenverträgen ermöglicht. Das festgelegte Metaschema erlaubt es ferner, zur Repräsentation des Inhalts *mehrere Darstellungsformate* zuzulassen und deren Äquivalenz zueinander sicherzustellen. Dies gelingt aufgrund seiner Funktion als *normiertes Zwischenformat*, die das Metaschema beim Übersetzen zwischen verschiedenen Repräsentationen ausübt. Dabei werden die Konstrukte der verschiedenen Sprachen eindeutig aufeinander abgebildet, ohne dass zwischen den verschiedenen Formaten direkt übersetzt werden muss. Der UNSCOM Spezifikationsrahmen nutzt diese Möglichkeit und definiert drei unterschiedliche Formate, um Komponentenspezifikationen für die am Entwicklungsprozess beteiligten Zielgruppen in geeigneter Weise darzustellen (vgl. Abb. 4.2).

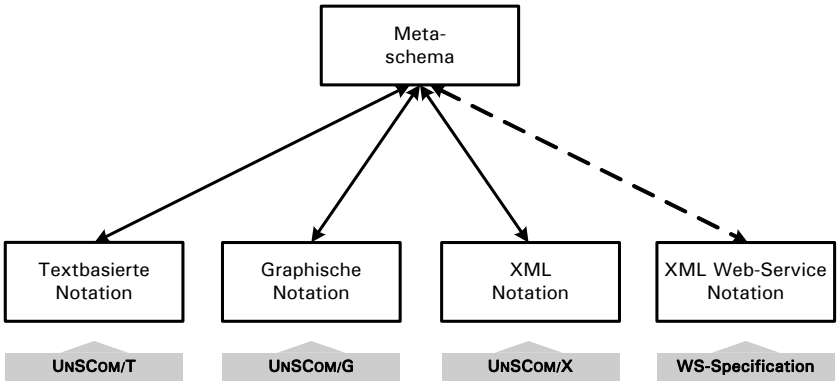


Abb. 4.2: Repräsentationsformate und Sprachhierarchie des UNSCOM Spezifikationsrahmens.

Das *textbasierte Format* (UNSCOM/T) stellt Komponentenspezifikationen unter Verwendung etablierter, technologieunabhängiger Notationen in einer Weise dar, die eine kompakte Repräsentation ermöglicht. Eine textbasierte Repräsentation wird bspw. von Programmierern genutzt. Durch das *graphische Format* (UNSCOM/G) werden Komponentenspezifikationen unter Verwendung der UML 2.0 [OMG 2003b] so dargestellt, dass vor allem eine übersichtliche Darstellung der Zusammenhänge ermöglicht wird. Eine solche Darstellung wird von Analytikern und Architekten gegenüber einer textbasierten Repräsentation bevorzugt. Dagegen ermöglicht das *XML Format* (UNSCOM/X) eine Repräsentation von Komponentenspezifikationen in einer datenorientierten Weise, die vor allem für Entwicklungswerkzeuge geeignet ist. Neben den genannten Formaten ist die Einbringung weiterer Formate möglich. So wird zurzeit an der Integration eines textbasierten Formats (WS-Specification) gearbeitet, welches die Spezifikation von XML Web-Services unter Verwendung der hierfür standardisierten Notationen ermöglicht. Eine Beschreibung dieses Formats findet sich in [OVERHAGE und THOMAS 2005].

4.1.3 Geltung

Mit der Spezifikation der Komponentenaußensicht ist zugleich ein gemeinsames *Verständnis* aller am Entwicklungsprozess beteiligter Personen über die beschriebenen Eigenschaften und deren Bedeutung herzustellen. Dieses Verständnis kann bspw. durch methodische Vorgaben erzielt werden, die die *Vorgehensweise* festlegen, mit der einzelne Spezifikationsteile zu erarbeiten und anschließend zu verwenden sind. Häufig unterscheiden sich jedoch gerade die in den Entwicklungsprojekten eingesetzten Vorgehensweisen zur Erstellung einer Spezifikation erheblich voneinander, so dass derartige Vorgaben nur mit einem unverhältnismäßig hohen Aufwand umzusetzen sind. Wegen der nur eingeschränkt vorhandenen Allgemeingültigkeit sind *aktivitätsorientierte Vorgaben*, die konkrete Aufgaben

und die zu ihrer Lösung durchzuführenden Arbeitsschritte festlegen [DOWSON 1987:37], für die Aufnahme in den universell einsetzbaren UNSCOM Spezifikationsrahmen demnach nicht geeignet.

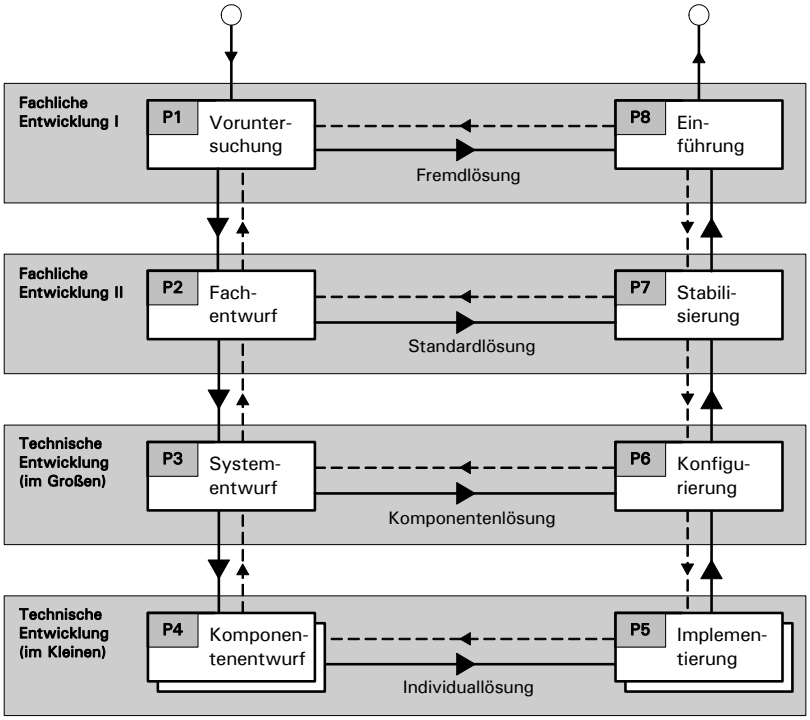


Abb. 4.3: Vereinheitlichtes Vorgehensmodell der komponentenorientierten Anwendungsentwicklung.

Um die Geltung der einzelnen Spezifikationsteile zu klären, verwendet der UNSCOM Spezifikationsrahmen deshalb vorzugsweise *ergebnisorientierte Vorgaben*, die lediglich die Eigenschaften der Teilbeschreibungen festlegen und angeben, wie die einzelnen Spezifikationsteile aufeinander aufbauen [DOWSON 1987:37]. Durch diese Vorgaben wird einerseits die *Bedeutung* der einzelnen Spezifikationsteile geklärt und normiert. Andererseits werden für jede Teilbeschreibung sowohl der Zeitpunkt ihrer *Entstehung* als auch die Zeitpunkte ihrer möglichen *Verwendungen* während des Entwicklungsprozesses angegeben. Für die Zuordnung von Spezifikationsteilen zu Entwicklungsphasen wird dabei auf das in der Gegenstandsbestimmung entwickelte vereinheitlichte Vorgehensmodell der komponentenorientierten Anwendungsentwicklung zurückgegriffen (vgl. Abschnitt 2.2.2), das eine geeignete Abstraktion etablierter Vorgehensmodelle bildet und in Abb. 4.3 noch einmal in der Zusammenschau dargestellt ist.

Aufbauend auf dieser zunächst überblicksartig vorgenommenen Einordnung werden die Vorgaben des UNSCOM Spezifikationsrahmens in den folgenden Abschnitten im Detail betrachtet. Die ausführliche Betrachtung der Vorgaben folgt der thematischen Gliederung des Spezifikationsrahmen aus Abb. 4.1, d.h. sie beginnt mit den White Pages, die die all-

gemeinen und kommerziellen Informationen beinhalten, und endet mit den Grey Pages, die die Qualität der Komponente und ihrer Dienste beschreiben. Zur Illustration der Vorgaben am Beispiel wird zudem auf die in den Abschnitten 2.4 und 3.1.7 betrachtete Fallstudie und die dort bereits voneinander abgegrenzten Komponenten zurückgegriffen, die im Folgenden zumindest auszugsweise spezifiziert werden.

4.2 Allgemeine und kommerzielle Informationen

Bei der Spezifikation einer Komponente sind neben anderen Eigenschaften auch allgemeine Merkmale zu beschreiben und Angaben hinsichtlich des (kommerziellen) Gebrauchs zu machen. Diese Informationen werden vom UNSCOM Spezifikationsrahmen durch die sog. *White Pages* zu einem eigenen thematischen Bereich zusammengefasst, der den festzulegenden Komponentenvertrag um hilfreiche, jedoch für die Entwicklungsarbeit im engeren Sinne nicht zwingend benötigte Daten ergänzt. Obwohl diese Daten damit besonders im Vergleich zu den anderen Vertragsebenen eine geringere Bedeutung besitzen, erweisen sie sich in der Praxis dennoch vor allem für die Handhabung, die Katalogisierung und den kommerziellen Gebrauch von Komponenten als wesentlich [ACKERMANN, et al. 2002:20]. Der UNSCOM Spezifikationsrahmen trägt dieser Bedeutung deshalb durch eine Reihe spezieller Vorgaben Rechnung, die den Inhalt, die Repräsentation sowie die Geltung der allgemeinen und kommerziellen Informationen über Komponenten festlegen.

4.2.1 Inhalt

Bei den inhaltlichen Vorgaben ist zunächst zwischen denjenigen, die sich auf die allgemeinen Informationen beziehen, und denjenigen, die die zu beschreibenden kommerziellen Informationen bestimmen, zu unterscheiden. Die Vorgaben hinsichtlich der zu spezifizierenden allgemeinen Informationen orientieren sich dabei vorwiegend an denen des UDDI Standards [CAULDWELL, et al. 2001:188-191; CERAMI 2002:162-166], der die Beschreibung von XML Web-Services und ihren jeweiligen Anbietern regelt. Dagegen stellen die Vorgaben zur Festlegung der für den kommerziellen Gebrauch notwendigen Informationen eine Weiterentwicklung dar, die auf den Ergebnissen eines Standardisierungsprojekts aufbaut [ACKERMANN, et al. 2002:21f.].

4.2.1.1 Allgemeine Informationen

Als allgemeine Informationen lassen sich mit dem UNSCOM Spezifikationsrahmen sowohl *administrative*, *technische* als auch *organisatorische Komponentenmerkmale* beschreiben.

Zu den administrativen Komponentenmerkmalen gehören der *Name* der Komponente, eine optionale eindeutige *Kennung*, eine *Versions*- sowie eine optionale *Variantenbezeichnung*, eine optionale *Beschreibung* und die ebenfalls optional anzugebende *Installationsbasis* (vgl. Abb. 4.4). Der Name der Komponente besteht aus einem alphanumerischen Bezeichner, mit dem die Komponente vor allem von Entwicklern identifiziert werden kann. Dieser lässt sich durch einen eindeutigen, maschinenlesbaren Schlüssel (einen sog. Universal Unique Identifier [CAULDWELL, et al. 2001:188]) ergänzen. Die Versionskennung beschreibt den aktuellen Entwicklungsstand der Komponente, der meist in Form einer Nummer angegeben wird [ACKERMANN, et al. 2002:21]. Die Versionskennung ist durch eine Variantenkennung zu ergänzen, falls die Komponente zu einer Baureihe von Komponenten mit variierenden funktionalen oder qualitativen Eigenschaften gehört. Als Varianten gelten auch solche Komponenten, die sich lediglich hinsichtlich der zur Realisierung eingesetzten Technologie oder der Nutzungsbedingungen voneinander unterscheiden. Die Installationsbasis gibt schließlich an, wie häufig die Komponente bereits in Anwendungsentwicklungsprojekten eingesetzt wurde.

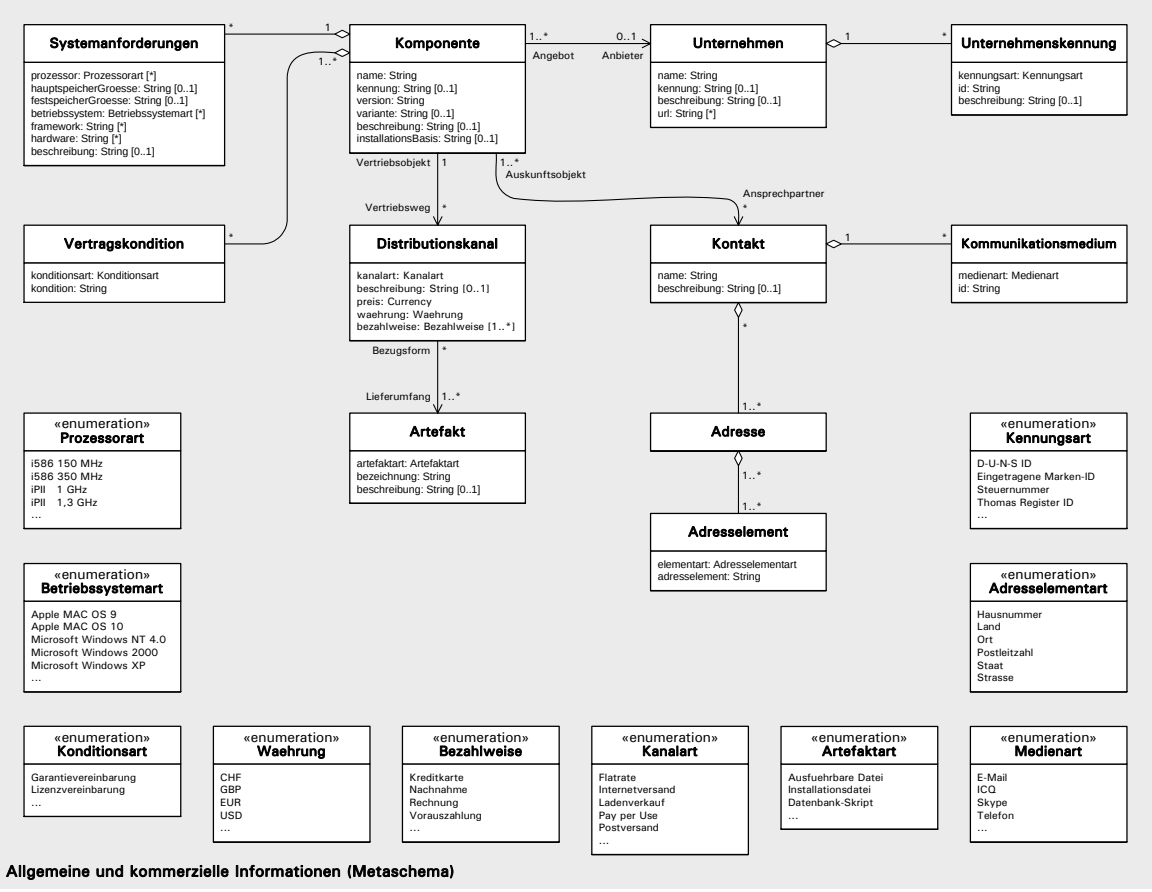


Abb. 4.4: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von allgemeinen und kommerziellen Komponentenmerkmalen (die Alternativenmengen sind nur auszugsweise dargestellt).

Die technischen Komponentenmerkmale geben Aufschluss über die Randbedingungen des Betriebs bzw. der Nutzung, die in Form von *Systemanforderungen* dargestellt werden. Durch eine Menge zusammengehöriger Systemanforderungen wird dabei jeweils eine von ggf. mehreren Systemkonfiguration beschrieben, die aus (alternativen) *Prozessorarchitekturen*, der vorausgesetzten *Haupt-* und *Festspeichergröße*, (alternativen) *Betriebssystemtypen* sowie den ggf. benötigten *Anwendungssystemteilen* besteht [ACKERMANN, et al. 2002:22]. Jede Systemkonfiguration lässt sich durch eine optionale *Beschreibung* näher charakterisieren und dabei bspw. als minimale oder optimale Konfiguration klassifizieren. Die organisatorischen Merkmale beschreiben schließlich den *Hersteller* der Komponente und die ggf. vorhandenen *Ansprechpartner*, die weitere Auskünfte über die Komponente geben können [CAULDWELL, et al. 2001:188f.]. Ähnlich wie die Komponente wird auch der Hersteller durch seinen Namen, optionale Unternehmenskennungen, eine ggf. vorhandene Internetadresse sowie eine optionale Beschreibung spezifiziert [CAULDWELL, et al. 2001:188]. Als Unternehmenskennungen können dabei neben einem maschinenlesbaren Schlüssel auch eindeutige Kennungen aus dem wirtschaftlichen Umfeld (wie bspw. die Umsatzsteueridentifikationsnummer) angegeben werden [CAULDWELL, et al. 2001:189]. Die Ansprechpartner sind durch ihre Adressen sowie die zur Kontaktaufnahme jeweils einsetzbaren Kommunikationsmedien zu beschreiben (vgl. Abb. 4.4).

4.2.1.2 Kommerzielle Informationen

Neben den zuvor genannten allgemeinen Informationen sieht der UNSCOM Spezifikationsrahmen auch die Beschreibung von Merkmalen vor, die beim (kommerziellen) Gebrauch einer Komponente nachgefragt werden. Hierzu gehört zum einen die Angabe der *Nutzungsbedingungen*, die in Form vertraglicher Konditionen angegeben werden können [ACKERMANN, et al. 2002:22]. Zum anderen lassen sich die verschiedenen (alternativen) *Distributionskanäle* spezifizieren, über die die Komponente bezogen bzw., im Falle einer physischen Wiederverwendung, in Anspruch genommen werden kann (vgl. Abb. 4.4). Jeder Distributionskanal ist dabei durch einen *Typ*, der die Bezugsform festlegt, sowie eine optionale *Beschreibung* zu charakterisieren. Darüber hinaus sind der für den Gebrauch der Komponente zu entrichtende *Preis*, die dazugehörige *Währung*, die akzeptierten *Bezahlmethoden* und die jeweils im Lieferumfang enthaltenen *Artefakte* anzugeben [ACKERMANN, et al. 2002:22].

Da sowohl die genannten allgemeinen als auch die kommerziellen Merkmale vorwiegend beschreibender Natur sind, wird bei der Spezifikation üblicherweise auf die Umgangssprache zurückgegriffen. Zur Gewährleistung der (automatisierten) Einordnung und Vergleichbarkeit wird die Beschreibung an zahlreichen Stellen durch *vorgegebene Alternativenmengen* eingeschränkt, die im Metaschema als Aufzählungen festgelegt sind (vgl. Abb. 4.4

unten). Unter Verwendung solcher vom UNSCOM Spezifikationsrahmen standardisierter Alternativenmengen sind bspw. die Währung, der Typ des Distributionskanals, die Art der vertraglichen Kondition oder der Prozessortyp zu beschreiben. Das Konzept der vorgegebenen Alternativenmengen, das dem UDDI Standard entnommen wurde [CAULDWELL, et al. 2001:189], zielt dabei vor allem auf eine stärkere Einheitlichkeit von Spezifikationen. Diese Einheitlichkeit wird jedoch durch eine Einschränkung der Freiheitsgrade bei der Spezifikation von Merkmalen erreicht. Um die praktische Einsetzbarkeit zu gewährleisten und die auftretenden Einschränkungen zu minimieren, wurde bei der Gestaltung des UNSCOM Metaschemas deshalb darauf geachtet, dass die standardisierten Alternativenmengen möglichst mit geringem Aufwand und ggf. in regelmäßigen Abständen ergänzt werden können.

4.2.2 Repräsentation

Entsprechend ihrer Funktion als ergänzende Daten eines Komponentenvertrags sind die allgemeinen und kommerziellen Informationen möglichst kompakt und übersichtlich darzustellen. Als geeignetes Repräsentationsformat ist deshalb sowohl im Rahmen von UNSCOM/T als auch von UNSCOM/G die *Tabellenform* zu wählen, wobei der Aufbau der Tabelle direkt aus dem Metaschema abzuleiten ist. Im Rahmen des UNSCOM/X Formats wird diese Tabelle auf der Basis eines festgelegten XML Schemas, das in Anhang B dargestellt ist, in ein maschinenlesbares XML Format abgebildet. Unter Verwendung der Tabellenform von UNSCOM/T bzw. UNSCOM/G werden die White Pages im Folgenden anhand eines Beispiels dargestellt, das allgemeine und kommerzielle Informationen über die Komponente Lagermanagement aus der Fallstudie (vgl. Abschnitt 3.1.7) beschreibt. Dabei werden die allgemeinen und kommerziellen Informationen wie schon bei der Darstellung der inhaltlichen Vorgaben getrennt voneinander betrachtet.

4.2.2.1 Allgemeine Informationen

Die vom UNSCOM Spezifikationsrahmen festgelegte Tabellenform für die Beschreibung der allgemeinen Informationen über eine Komponente ist zunächst in Abb. 4.5 beispielhaft dargestellt. Die abgebildete Tabelle umfasst die Elemente Komponente, Systemanforderungen, Unternehmen, Unternehmenskennung, Kontakt, Adresse, Adresselement und Kommunikationsmedium des Metaschemas mitsamt den dazugehörigen Alternativenmengen. Seiner jeweiligen Rolle entsprechend wird das Element Unternehmen in der Tabelle als *Anbieter* der Komponente und das Element Kontakt als *Ansprechpartner* aufgeführt. Der Metamodelltyp ist dahinter jeweils in Klammern genannt.

Name: Oversoft Lagermanagement
Kennung: 206B2F65-E7BC-47A0-8DCF-4E34347322AA
Version: 1.0.3755
Variante: Small Business Edition
Beschreibung: Oversoft Lagermanagement stellt Funktionen zur Lagerverwaltung zur Verfügung, die den Warenein- und -ausgang sowie die Bestandsverwaltung abdecken. Die Small Business Edition ist dabei für mittelständische Betriebe gedacht und erlaubt die Verwaltung von maximal fünf Lagerstätten ...
Installationsbasis: < 10
Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1GHz)
Hauptspeichergröße: 256 MB
Festspeichergröße: 512 MB
Betriebssystem (Microsoft Windows 2000)
Beschreibung: Minimale Systemanforderungen
Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1,5GHz)
Hauptspeichergröße: 1024 MB
Festspeichergröße: 2048 MB
Betriebssystem (Microsoft Windows XP)
Beschreibung: Optimale Systemanforderungen
Anbieter (Typ: Unternehmen)
Name: Oversoft Software
Kennung: AEFBE5B6-A13F-4CA4-B322-32ADB89EA14C
Beschreibung: Oversoft Software ist spezialisiert auf die Bereitstellung von Werkzeugen zur Unterstützung des komponentenorientierten Entwicklungsprozesses und auf die Bereitstellung spezialisierter Komponenten bzw. XML Web-Services ...
URL: http://www.oversoft.biz
Unternehmenskennung (Typ: Eingetragene Marken-ID)
ID: 30071881
Beschreibung: Eintrag ins deutsche Markenregister
Ansprechpartner (Typ: Kontakt)
Name: Oversoft Marketing
Adresse (Typ: Adresse)
Adresselement (Typ: Strasse): An der Steinmauer
Adresselement (Typ: Hausnummer): 2
Adresselement (Typ: Postleitzahl): 61191
Adresselement (Typ: Ort): Rosbach
Adresselement (Typ: Land): Deutschland
Kommunikationsmedium (Typ: E-Mail): sales@oversoft.biz
Kommunikationsmedium (Typ: Telefon): +49 6003 9300-71
Kommunikationsmedium (Typ: Telefax): +49 6003 9300-72

Abb. 4.5: Spezifikation der allgemeinen Komponentenmerkmale in UNSCOM/T und UNSCOM/G.

4.2.2.2 Kommerzielle Informationen

Die zur Spezifikation der kommerziellen Informationen zu verwendende Tabellenform ist in Abb. 4.6 beispielhaft dargestellt. Die gezeigte Tabelle ergänzt die allgemeinen Informationen und umfasst die Elemente Vertragskondition, Distributionskanal und Artefakt des

UNSCOM Metaschemas sowie die dazugehörigen Aufzählungen. Auch an dieser Stelle sind die aufgeführten Elemente entsprechend ihrer jeweiligen Rolle benannt.

- **Vertriebsweg** (Typ: Distributionskanal)
  - **Kanaltyp** (Typ: Internetversand)
  - **Preis**: 990.00
  - **Waehrung** (Typ: EUR)
  - **Bezahlweise** (Typ: Kreditkarte)
  - **Bezahlweise** (Typ: Vorauszahlung)
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.exe
    - **Typ** (Typ: Installationsdatei)
    - **Beschreibung**: Datei zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.hlp
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zum Installieren der Komponente
- **Vertriebsweg** (Typ: Distributionskanal)
  - **Kanaltyp** (Typ: Postversand)
  - **Preis**: 1190.00
  - **Waehrung** (Typ: EUR)
  - **Bezahlweise** (Typ: Kreditkarte)
  - **Bezahlweise** (Typ: Nachnahme)
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.exe
    - **Typ** (Typ: Installationsdatei)
    - **Beschreibung**: Datei zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.hlp
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: handbook.pdf
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zur Installation und Verwendung der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: demo.exe
    - **Typ** (Typ: Sonstige)
    - **Beschreibung**: Informationen über den Einsatz der Komponente und ihr Zusammenspiel mit anderen Komponenten
- **Vertragskondition** (Typ:Vertragskondition)
  - **Konditionstyp** (Typ: Lizenzvereinbarung)
  - **Kondition**: Oversoft End-User Lizenzvereinbarung (EULA) ...
- **Vertragskondition** (Typ:Vertragskondition)
  - **Konditionstyp** (Typ: Garantievereinbarung)
  - **Kondition**: Oversoft Garantiebestimmungen ...

Abb. 4.6: Spezifikation der kommerziellen Komponentenmerkmale in UNSCOM/T und UNSCOM/G.

4.2.3 Geltung

Die zuvor dargestellten allgemeinen und kommerziellen Informationen werden im Rahmen des Entwicklungsprozesses zu verschiedenen Zwecken genutzt, die voneinander zu unterscheiden sind. So bilden die *administrativen Merkmale* wie der Name oder die Beschreibung einer Komponente den Ausgangspunkt für die Spezifikation der Innensicht während des Komponentenentwurfs und die nach Abschluss der Implementierung durchzuführende Dokumentation der Realisierung. Sie sind als Teil des Systementwurfs festzulegen, in dessen Rahmen die Entscheidung zur Neu- bzw. Weiterentwicklung einer Komponente fällt. Die technischen Merkmale einer Komponente sind dagegen ggf. bereits aus den Vorgaben des Pflichtenhefts abzuleiten. Sie sind bei der Suche und Auswahl von Komponenten als Einschränkungen zu berücksichtigen und für den Fall, dass keine geeigneten Komponenten gefunden wurden, während des Systementwurfs in die Spezifikation(en) der neu zu entwickelnden Komponente(n) zu übernehmen. Dann beschreiben sie einen Soll-Zustand, der bei der Komponentenentwicklung einzuhalten ist. Nach Abschluss der Implementierung sind sie durch diejenigen Anforderungen zu ersetzen, die sich aufgrund der Realisierung tatsächlich ergeben und den vorgegebenen Soll-Zustand nicht überschreiten dürfen.

Wie die administrativen Merkmale werden auch die *organisatorischen Merkmale* einer Komponente während des Systementwurfs definiert, in dessen Rahmen die Komponentenentwicklung beschlossen wurde. Durch die Vorgabe eines Herstellers und eines Ansprechpartners werden dabei zunächst die Verantwortlichkeiten für die Realisierung festgelegt. Nach Abschluss der Implementierung ist dann ein Ansprechpartner zu beschreiben, der bei einer Wiederverwendung der Komponente mit Auskünften zur Verfügung steht. Dieser Kontakt kann bei einem unternehmensinternen Komponentenmarkt durchaus ein anderer sein als bei einem unternehmensexternen Markt. Insbesondere im letztgenannten Fall sind die allgemeinen Informationen bei der Einführung um Angaben hinsichtlich des kommerziellen Gebrauchs zu ergänzen. Die zu beschreibenden *kommerziellen Informationen* dienen somit primär der Dokumentation der Realisierung. Sie unterstützen den Entwickler bei der Suche bzw. Auswahl von Komponenten und den Einkäufer bei der Beschaffung.

4.3 Klassierungen

Die weiteren Merkmale, die zur Ergänzung eines Komponentenvertrags zu beschreiben sind, dienen der Klassierung von Komponenten. Diese *klassierenden Merkmale* werden aufgrund ihrer speziellen Bedeutung vom UNSCOM Spezifikationsrahmen zu einem eige-

nen thematischen Bereich zusammengefasst und in Anlehnung an den UDDI Standard als *Yellow Pages* bezeichnet [CAULDWELL, et al. 2001:190; CERAMI 2002:158]. Im Gegensatz zu diesem Standard schränkt der UNSCOM Spezifikationsrahmen das Vorgehen bei der Klassierung von Komponenten jedoch durch ein *Klassifikationsschema* ein, das verbindliche inhaltliche Vorgaben enthält. Diese Vorgaben werden anschließend durch weitere Vorgaben hinsichtlich der Repräsentation und Geltung von Klassierungen ergänzt.

4.3.1 Inhalt

Durch eine Klassierung werden verschiedene Komponenten mit gleichen Merkmalsausprägungen zu Gruppen (Klassen) zusammengefasst. Alle zu einer Gruppe gehörenden Komponenten besitzen folglich mindestens eine gemeinsame Merkmalsausprägung und sind bezüglich dieser Ausprägung austauschbar [PRIETO-DÍAZ und FREEMAN 1987:8]. Die Klassierung erfolgt anhand eines *Klassifikationsschemas*, das eine Reihe von Merkmalen (wie z.B. die Komponententechnologie und den Anwendungsbereich) vorgibt. Durch das Schema wird außerdem ein normiertes Vokabular von Indextermen (Klassen) als Ausprägungen der Merkmale vorgegeben, das durch hierarchische und syntaktische Beziehungen zwischen den Indextermen strukturiert ist. Hierarchische Beziehungen basieren auf der Inklusion und beschreiben Spezialisierungen zwischen Indextermen. Mit ihnen lassen sich die Ausprägungen eines Merkmals in einer Klassenhierarchie anordnen. Syntaktische Beziehungen basieren dagegen auf der Komposition und beschreiben Verbindungen zwischen Indextermen, die Ausprägungen unterschiedlicher Merkmale darstellen. Durch sie werden die verschiedenen Merkmale einer Klassifikation miteinander in Zusammenhang gesetzt [PRIETO-DÍAZ und FREEMAN 1987:8].

Klassifikationsschemata lassen sich prinzipiell auf zwei unterschiedliche Weisen gestalten: die traditionelle *aufzählende Vorgehensweise* gliedert einen Weltausschnitt streng hierarchisch und besteht aus einer einzigen Klassenhierarchie. Syntaktische Beziehungen zwischen den Indextermen werden dabei durch spezielle Unterklassen abgebildet, die durch Kombination der zu verbindenden Merkmalsausprägungen entstehen [PRIETO-DÍAZ und FREEMAN 1987:8]. Die *facettenorientierte Vorgehensweise* sieht dagegen für jedes Merkmal eine eigenständige Klassenhierarchie mit Ausprägungen vor, die als Facette bezeichnet wird. Syntaktische Beziehungen werden bei diesem Ansatz grundsätzlich zwischen den einzelnen Facetten (auf Schemaebene) festgelegt und sind für sämtliche Ausprägungen verbindlich. Bei der Klassierung nach einem facettenorientierten Schema ist für jede der Merkmale eine Auswahl aus der zugehörigen Klassenhierarchie zu treffen, die als *Klassifikationseintrag* bezeichnet wird [PRIETO-DÍAZ und FREEMAN 1987:8].

Obwohl beide Ansätze zur Gestaltung eines Klassifikationsschemas hinsichtlich der erreichbaren Ausdruckskraft prinzipiell gleichmächtig sind, ist die facettenorientierte Vorgehensweise der aufzählenden im Allgemeinen überlegen [PRIETO-DÍAZ 1991:89]. Hierfür spricht vor allem, dass bei jeder Kombination von Ausprägungen verschiedener Merkmale im Rahmen der aufzählenden Vorgehensweise eine Vielzahl zusammengesetzter Klassen entsteht (als Beispiel vgl. Abb. 4.7), die nicht nur das Klassifikationsschema unübersichtlich werden lässt, sondern auch das Hinzufügen von Merkmalen erschwert [PRIETO-DÍAZ und FREEMAN 1987:9].

Anwendungsbereiche	Technologien
Horizontale Anwendungsbereiche	CORBA Component Model
Datenbankmanagement	CORBA Component Model 1.0
Kommunikationsmanagement	Microsoft .NET
Verschlüsselung	Microsoft .NET 1.0
Workflowmanagement	Microsoft .NET 1.1
Vertikale Anwendungsbereiche	Sun Enterprise JavaBeans
Betriebswirtschaft	Sun Enterprise JavaBeans 1.0
Biotechnologie	Sun Enterprise JavaBeans 2.0

Komponentenklassen
- Horizontale Anwendungsbereiche
  - Datenbankmanagement
    - Datenbankmanagement mit CORBA Component Model 1.0
    - Datenbankmanagement mit Microsoft .NET 1.0
    - Datenbankmanagement mit Microsoft .NET 1.1
    - Datenbankmanagement mit Sun JavaBeans 1.0
    - Datenbankmanagement mit Sun JavaBeans 2.0

...

Abb. 4.7: Facettenorientiertes (oben) und aufzählendes (unten) Klassifikationsschema am Beispiel.

Der UNSCOM Spezifikationsrahmen nutzt zur Klassierung von Komponenten deshalb ein facettenorientiertes Klassifikationsschema, das eine Reihe standardisierter Merkmale vorsieht und sich bei Bedarf um weitere, proprietäre Merkmale erweitern lässt. Verpflichtend vorzunehmen sind dabei Klassierungen, die Aufschluss über die bei der Realisierung verwendete *Komponententechnologie*, die *Komponentenart* (entsprechend dem in Abschnitt 3.1.6 skizzierten konzeptionellen Komponentenmodell), die Art der *Wiederverwendung* sowie den *Anwendungsbereich* geben. Ergänzt werden können die genannten Klassierungen durch eine optionale Angabe von *Funktionsbereichen*, in denen die einzuordnende Komponente jeweils praktisch eingesetzt werden kann (vgl. Abb. 4.8).

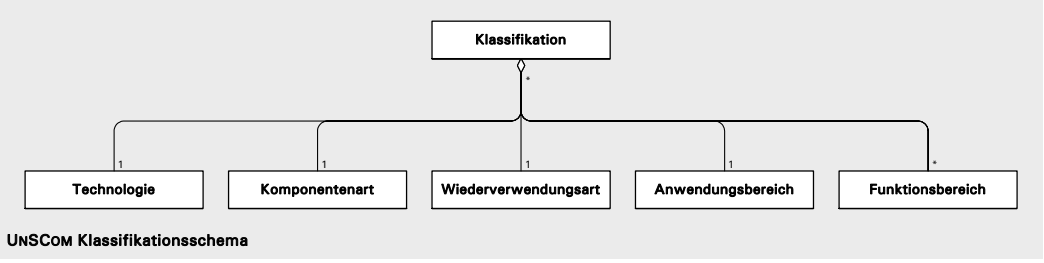


Abb. 4.8: Standardisiertes facettenorientiertes Klassifikationsschema des UNSCOM Spezifikationsrahmens.

Die angegebenen Funktionsbereiche konkretisieren dabei den abstrakt formulierten Anwendungsbereich einer Komponente. Eine solche Konkretisierung ist immer dann notwendig, wenn eine Komponente durch die Angabe ihres Anwendungsbereichs nicht ausreichend eingeordnet werden kann. So können sich bspw. die Dienste einer Komponente zur Produktionssteuerung (Anwendungsbereich) in Abhängigkeit davon unterscheiden, ob sie für den Einsatz in einer Chemiefabrik oder in einem Werk der Automobilindustrie (Funktionsbereich) konzipiert wurden [PRIETO-DÍAZ und FREEMAN 1987:11]. Für jedes der im Klassifikationsschema genannten Merkmale stellt der UNSCOM Spezifikationsrahmen ein normiertes Vokabular von Indextermen zur Verfügung, mit dem sich seine Ausprägungen beschreiben lassen. Die Indexterme sind jeweils in Form einer *Taxonomie*, d.h. als hierarchisch strukturiertes System von Bezeichnern angeordnet [STOJANOVIC 2005:1330]. Zur Beschreibung von Merkmalsausprägungen wird dabei möglichst auf standardisierte, in der Praxis etablierte Taxonomien zurückgegriffen.

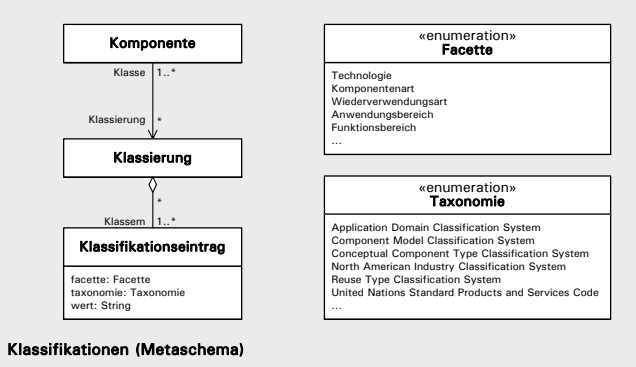


Abb. 4.9: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Klassierungen.

Das Metaschema des UNSCOM Spezifikationsrahmens sieht zur Einordnung von Komponenten zunächst die Angabe von *Klassierungen* vor, die jeweils aus einem oder mehreren *Klassifikationseinträgen* bestehen (vgl. Abb. 4.9 links). Prinzipiell können Klassierungen damit unabhängig von einem speziellen Klassifikationsschema nach einer facettenorientierten Vorgehensweise vorgenommen werden. Für jeden Klassifikationseintrag ist deshalb die zugehörige *Facette* anzugeben, damit die angegebene Ausprägung einem Merkmal zugeordnet werden kann. Darüber hinaus sind die zur Spezifikation der Merkmalsausprägung verwendete Taxonomie und die Bezeichnung der Ausprägung anzugeben. Grundsätzlich lassen sich also wie beim UDDI Standard verschiedene Taxonomien zur Beschreibung eines Merkmals einsetzen [CAULDWELL, et al. 2001:184f., 190f.]. Bei der Klassierung von Komponenten nach dem UNSCOM Klassifikationsschema sind die jeweils einzusetzenden Taxonomien jedoch verbindlich vorgegeben. In diesem Fall ist eine Mischung aus speziell entwickelten und etablierten Taxonomien wie dem *North American Industry Classification*

System (NAICS) oder dem *United Nations Standard Products and Services Code* (UNSPSC) zu verwenden, die in Abb. 4.9 rechts als Aufzählung aufgeführt sind.

4.3.2 Repräsentation

Wie die in den White Pages beschriebenen Informationen stellen auch die klassierenden Merkmale Angaben dar, die den Inhalt eines Komponentenvertrags ergänzen. Um die gewünschte kompakte Darstellung dieser zusätzlichen Informationen beizubehalten, sind im Rahmen der beiden Formate UNSCOM/T und UNSCOM/G deshalb auch die Yellow Pages in *Tabellenform* zu beschreiben. Die Struktur der Tabelle ist dabei wie zuvor aus dem Metaschema abzuleiten. Im Rahmen von UNSCOM/X ist die Tabellenform durch ein maschinenlesbares XML Dokument zu ersetzen, das hinsichtlich seiner Struktur ebenfalls mit dem Metaschema übereinstimmt. Ein XML Schema, das die geforderte Strukturanalogie bei der Übersetzung zwischen den Formaten gewährleistet, findet sich als Bestandteil der Definition von UNSCOM/X in Anhang B.

Klassierung (Typ: Klassierung)		
	Klassem (Typ: Klassifikationseintrag)	
		Facette (Typ: Technologie)
		Taxonomie: (Typ: Component Model Classification System)
		Wert: Microsoft .NET 1.1
	Klassem (Typ: Klassifikationseintrag)	
		Facette (Typ: Komponentenart)
		Taxonomie: (Typ: Conceptual Component Type Classification System)
		Wert: Application-Specific Component
	Klassem (Typ: Klassifikationseintrag)	
		Facette (Typ: Wiederverwendungsart)
		Taxonomie: (Typ: Reuse Type Classification System)
		Wert: Logical Reuse
	Klassem (Typ: Klassifikationseintrag)	
		Facette (Typ: Anwendungsbereich)
		Taxonomie: (Typ: Application Domain Classification System)
		Wert: Warehousing
	Klassem (Typ: Klassifikationseintrag)	
		Facette (Typ: Funktionsbereich)
		Taxonomie: (Typ: North American Industry Classification System)
		Wert: General Warehousing and Storage

Abb. 4.10: Spezifikation der klassierenden Komponentenmerkmale in UNSCOM/T und UNSCOM/G.

In Abb. 4.10 ist die Tabellenform einer Klassierung gemäß dem UNSCOM Klassifikationsschema am Beispiel dargestellt. Zur Illustration wird wiederum auf die Komponente La-

germanagement aus der in Abschnitt 3.1.7 betrachteten Fallstudie zurückgegriffen, deren Spezifikation nun um klassierende Merkmale ergänzt wird. Seiner Rolle im Metaschema entsprechend erscheint der Klassifikationseintrag in der Tabelle dabei als *Klassem*. Die zur Beschreibung der Merkmalsausprägungen zu verwendenden normierten Vokabulare sind in Anhang C aufgeführt, sofern es sich um speziell für den UNSCOM Spezifikationsrahmen entwickelte Taxonomien handelt. Die Indexterme der standardisierten Taxonomien NAICS und UNSPSC lassen sich den jeweiligen Standarddokumenten entnehmen.

4.3.3 Geltung

Die klassierenden Merkmale einer Komponente werden zusammen mit der Außensicht im Rahmen des Systementwurfs festgelegt. Während dieser Entwicklungsphase wird durch den Architekturentwurf vor allem die konzeptionelle Komponentenart sowie die Art der Wiederverwendung bestimmt. Die zur Realisierung einzusetzende Technologie sowie der Anwendungs- und Funktionalbereich sind dagegen unter Umständen bereits durch das Pflichtenheft determiniert. Diese Vorgaben sind somit in die jeweilige Komponentenspezifikation zu übernehmen. Nach Abschluss des Systementwurfs beschreibt die Komponentenspezifikation zunächst einen Soll-Zustand, der zur *Ausschreibung* eines entsprechenden Projektauftrags verwendet werden kann. Die vorgenommene Klassierung kann dabei zur Einstellung der Ausschreibung in einen Katalog genutzt werden, der nach mindestens einem klassierenden Merkmal gegliedert ist. Nach Vollendung der Implementierung kann die Klassierung dann genutzt werden, um die realisierte *Komponente* zur (Wieder-) Verwendung in anderen Projekten in einen entsprechend gegliederten Komponentenkatalog einzustellen. Die klassierenden Merkmale sind dann Bestandteil der Dokumentation und erleichtern das Auffinden der Komponente beim Durchsuchen des Katalogs.

Den klassierenden Merkmalen kommt eine fundamentale Bedeutung für die *Katalogisierung* von Komponenten und die Organisation der Kataloge zu, die die Komponentenspezifikationen enthalten [PRIETO-DÍAZ 1991:89]. Obwohl sie damit eine Schlüsselrolle zur Unterstützung des komponentenorientierten Entwicklungsparadigmas besitzen, wird ihre Angabe durch den UNSCOM Spezifikationsrahmen nicht zwingend vorgegeben. Ferner können das Klassifikationsschema und die Taxonomien mit den Merkmalsausprägungen bei Bedarf ersetzt werden. Damit wird vor allem der aktuellen Situation Rechnung getragen, in der sich Komponentenkataloge und Standards für die Klassierung von Komponenten erst noch in der Entstehung befinden. Das Klassifikationsschema des UNSCOM Spezifikationsrahmens stellt diesbezüglich jedoch zugleich einen Vorschlag dar, auf dessen Basis sich entsprechende Komponentenkataloge und Klassifikationsstandards entwickeln lassen.

4.4 Fachliche Funktionalität

Die im Rahmen der Yellow Pages durchgeführte Klassierung des Anwendungs- und Funktionsbereichs ermöglicht es zwar, die *Funktionalität* einer Komponente näher zu charakterisieren. Um sie aus fachlich-konzeptioneller Sicht umfassend und im Detail zu beschreiben, reicht eine Einordnung anhand zweier Merkmale, für die jeweils nur eine begrenzte Menge alternativer Ausprägungen bereitgestellt wird, im Allgemeinen jedoch nicht aus. Der UNSCOM Spezifikationsrahmen sieht deshalb zusätzlich eine formale Beschreibung der Funktionalität in Form eines *Begriffssystems* vor. Als Elemente dieses Begriffssystems sind die der Informationsverarbeitung zugrunde liegenden Informationsobjekte, Funktionen und Prozesse sowie die zwischen diesen Elementen bestehenden Zusammenhänge zu spezifizieren. Es deckt damit die in Abschnitt 3.4.2 zur Festlegung der Funktionalität identifizierten Ebenen eines Komponentenvertrags ab, die durch den UNSCOM Spezifikationsrahmen als sog. *Blue Pages* zu einem eigenen thematischen Bereich zusammengefasst werden. Die Spezifikation des Begriffssystems orientiert sich an einer Reihe von Vorgaben, mit denen sein Inhalt, seine Repräsentation und seine Geltung festgelegt werden.

4.4.1 Inhalt

Das mit der Einführung eines Anwendungssystems verbundene Ziel ist die Unterstützung des fachlichen Handelns bzw. der Informationsverarbeitung im jeweils zugrunde liegenden Anwendungsbereich [ORTNER 1997:20; D'SOUZA und WILLS 1999:548]. Aus fachlich-konzeptioneller Sicht wird die in einem Anwendungsbereich übliche Praxis des Handelns dabei durch das jeweils etablierte *Begriffssystem* dokumentiert [ORTNER 1997:17; GRIFFEL 1998:75f.]. Es besteht aus einer Menge von (Fach-) Begriffen, die in Form allgemeiner Aussagen miteinander in Beziehung gesetzt sind. Bei der in der Fallstudie betrachteten Lager- und Artikelorganisation besteht dieses System z.B. aus Begriffen wie »Lager«, »Lagerplatz«, »Artikel«, »Bestand«, »führen«, »zuweisen« sowie den aus diesen Begriffen gebildeten Aussagen wie etwa »Ein Lager unterteilt sich in mehrere Lagerbereiche«, »Für jeden Artikel ist ein Bestand zu führen« oder »Einem Artikel kann ein fester Lagerplatz zugewiesen werden« [BECKER und SCHÜTTE 2004:236-238].

In analoger Weise lässt sich die Funktionalität einer Komponente durch die Teilmenge des Begriffssystems beschreiben, die das jeweils von ihr unterstützte fachliche Handeln bzw. die Informationsverarbeitung dokumentiert. Zur Charakterisierung der Funktionalität tragen dabei zum einen die Bedeutung der einzelnen *Begriffe* und zum anderen die Bedeutung

der mit ihnen gebildeten Aussagen, d.h. der zwischen den Begriffen existierenden *Beziehungen* bei. Die Spezifikation des für die Komponente relevanten Begriffssystems basiert dementsprechend aus zwei Bestandteilen: einem *Lexikon*, das die Bedeutung der für die Informationsverarbeitung relevanten Begriffe klärt, sowie einer *Aussagensammlung*, die komplexe Sachverhalte beschreibt und hierzu Begriffe nach den Regeln einer Grammatik zu allgemeinen Aussagen mit einer jeweils spezifischen Bedeutung verbindet [ORTNER 1997:101f.; SCHIENMANN 1997:122].

Um zu einer formal eindeutigen Spezifikation des Begriffssystems zu gelangen, sind in einem ersten Schritt zunächst die im Lexikon aufgeführten Begriffe als *Terminologie* des Anwendungsbereichs sowohl hinsichtlich ihrer Bedeutung als auch ihrer Bezeichnung (dem sog. Begriffswort) eindeutig festzulegen [SCHIENMANN 1997:139]. Durch diese Festlegung lässt sich das Auftreten von Begriffsdefekten wie Synonymen, Homonymen oder Vagheiten vermeiden. In einem zweiten Schritt sind die Regeln der Grammatik in ihrer Bedeutung so zu normieren, dass von den in die Aussagensammlung aufzunehmenden Aussagen stets eindeutig auf den jeweils ausgedrückten Sachverhalt geschlossen werden kann [SCHIENMANN 1997:147]. Bei Einhaltung dieser beiden Voraussetzungen bildet die zur Dokumentation der Informationsverarbeitung spezifizierte Aussagensammlung ein eindeutig definiertes System von Begriffen, das auch als *Ontologie* bezeichnet wird [HESSE 2002:477; STOJANOVIC 2005:1330f.]. Sie beschreibt als *konzeptionelles Modell* einen Ausschnitt oder als *Domänen-* bzw. *Referenzmodell* einen gesamten Anwendungsbereich.

Jeder Begriff des spezifizierten Begriffssystems repräsentiert dabei durch sein *Begriffswort*, den Bezeichner (Prädikator), eine Klasse von Gegenständen aus dem Anwendungsbereich, die als Extension unter das Begriffswort fallen [BECKER und SCHÜTTE 2004:140]. Die Extension umfasst gemäß Abb. 4.11 links sowohl die physischen Gegenstände aus dem Anwendungsbereich (Extension I) als auch die Beschreibungen (Extension II), die die Gegenstände unter anderem im Rahmen der Informationsverarbeitung repräsentieren [BUNGE 1977:119-123]. So fallen unter das Begriffswort »Artikel« bspw. nicht nur alle realen Güter, die durch das in der Fallstudie betrachtete Handelsunternehmen zum Verkauf angeboten werden, sondern auch deren Beschreibungen, die in der Komponente Artikelmanagement abgelegt sind. Da andere Gegenstände des Anwendungsbereichs nicht unter dieses Begriffswort fallen, lässt sich ein Begriff prinzipiell als Wahrheitsfunktion betrachten, die bei der versuchten Assoziation eines Gegenstands aus dem Anwendungsbereich zum Begriffswort den Wert ›wahr‹ oder ›falsch‹ ergibt (vgl. [FREGE 1975]).

Die Zuordnung von Gegenständen und ihren Repräsentanten zu einem Begriffswort erfolgt anhand der Bedeutung des Begriffs, die sich aus der praktischen Verwendung des Beg-

riffsworts im Anwendungsbereich ergibt und als *Intension* bezeichnet wird [BECKER und SCHÜTTE 2004:140]. Die Intension bestimmt die gemeinsamen Merkmale derjenigen Gegenstände, die unter das Begriffswort fallen. Sie ist entweder durch eine explizite *Definition* der Bedeutung (epipraktisch) oder eine *exemplarische Einführung*, etwa anhand von Beispielen und Gegenbeispielen (empraktisch), festzulegen [SCHIENMANN 1997:124, 125-127, 131-139].

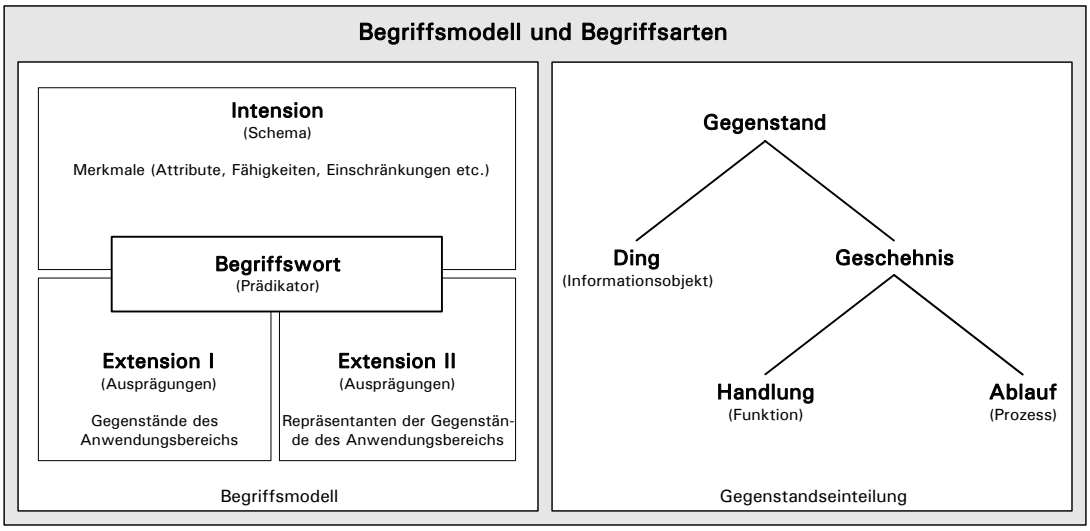


Abb. 4.11: Allgemeines Begriffsmodell und Einteilung in Begriffsarten nach ihrem jeweiligen Bezugsobjekt (in Anlehnung an [ORTNER 1997:59, 84f.; SCHIENMANN 1997:143, 112; SCHEER 1998:37]).

Nach ihrem jeweiligen Bezugsobjekt lassen sich Begriffe unterscheiden, die *Dinge*, *Handlungen* oder *Abläufe* eines Anwendungsbereichs unter einem Begriffswort zusammenfassen [ORTNER 1997:84f.; SCHIENMANN 1997:112]. Analog zu der in Abb. 4.11 rechts dargestellten Einteilung wird bei der Referenzmodellierung unter Bezugnahme auf die Extension II auch von Begriffen gesprochen, die *Informationsobjekte*, *Funktionen* oder *Prozesse* repräsentieren [SCHEER 1998:37]. Die explizite Unterscheidung von Begriffsarten erlaubt die Berücksichtigung von spezifischen Merkmalen und Beziehungen mit unterschiedlichen Wirkungen, die zwischen den Begriffsarten bestehen. Hierauf wird an späterer Stelle noch genauer eingegangen. Allgemein lassen sich zunächst *abstraktive Beziehungen*, die auf der (partiellen) Gleichheit von Begriffen basieren, und *kompositive Beziehungen*, die auf der Abhängigkeit von Begriffen basieren, voneinander abgrenzen [ORTNER 1997:123f., 128].

Das Metaschema des UNSCOM Spezifikationsrahmens baut auf diesem allgemeingültigen Begriffsmodell auf und sieht als relevante Elemente zur Spezifikation von Begriffen im Rahmen des *Lexikons* zunächst das Begriffswort, eine das Begriffswort ergänzende maschinenlesbare Kennung, sowie eine optionale Kurz- und Langdefinition vor (vgl. Abb. 4.12 rechts). Darüber hinaus können optional Beispiele und Gegenbeispiele angegeben

bzw. die Gegenstände der Extension durch Aufzählung oder Angabe eines Wertebereichs festgelegt werden. Neben den vom Metaschema unterstützten Verfahren zur Festlegung der Begriffsbedeutung anhand von Ausprägungen lassen sich mit einer Erweiterung des Metaschemas bei Bedarf weitere empraktische Verfahren einbinden, die die Verwendung eines Begriffsworts anhand der Extension festlegen. Ein Überblick über entsprechende Verfahren findet sich bei [SCHIENMANN 1997:126f.].

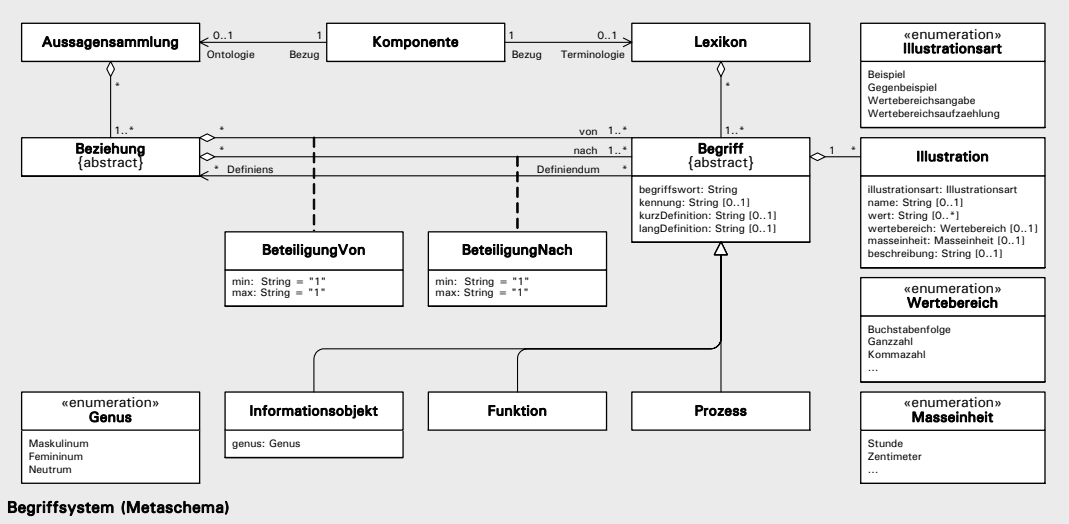


Abb. 4.12: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Begriffssystemen.

Zusätzlich zum Lexikon mit normierten Begriffen ist eine *Aussagensammlung* aufzubauen, mit der komplexe Sachverhalte des Anwendungsbereichs als allgemeine Aussagen beschrieben werden (vgl. Abb. 4.12 links). Die in der Aussagensammlung enthaltenen Aussagen setzen Begriffe des Lexikons auf der Basis einer normierten Grammatik miteinander in Beziehung, die die jeweils erlaubten Beziehungsarten vorgibt. Diese Grammatik wird in den folgenden Abschnitten im Detail vorgestellt. Zuvor ist noch darauf hinzuweisen, dass zwischen den Begriffen des Lexikons und den Aussagen der Aussagensammlung grundsätzlich ein enger *wechselseitiger Zusammenhang* besteht, der in Abb. 4.12 angedeutet ist.

Danach können anwendungsunabhängig gültige Aussagen auch zur Definition von Begriffen genutzt und dementsprechend zu Bestandteilen des Lexikons werden. Die Begriffsdefinitionen sind zu diesem Zweck, zumindest soweit Bezug auf andere Begriffe genommen wird, unter Verwendung der normierten Grammatik vorzunehmen [ORTNER 1997:101f.]. Das UNSCOM Metaschema sieht hierfür neben der Kurz- und Langdefinition einen zusätzlichen Definitionsteil vor, in den normiert dargestellte Beziehungen zu anderen Begriffen als *Definiens* aufzunehmen sind (vgl. Abb. 4.12). Die optionalen Kurz- bzw. Langdefinitionen bauen auf diesem Definitionsteil auf und ergänzen die Darstellung in Prosaform. Die

dem Definitionsteil zugeordneten Aussagen sind jedoch auch in die Aussagensammlung zu übernehmen, wenn sie Sachverhalte ausdrücken, die für die Spezifikation der Komponentenfunktionalität relevant sind. In der Regel sind deshalb alle im Lexikon aufgeführten Aussagen zugleich Bestandteil der Aussagensammlung. Die Definition von Begriffen, die bei der Spezifikation des Begriffssystems zuerst erfolgt, bildet damit den Ausgangspunkt für den sich anschließenden Aufbau der Aussagensammlung. Im Gegenzug können die dort neu spezifizierten Aussagen wiederum in die Definitionsteile des Lexikons einfließen und die Bedeutung von Begriffen weiter konkretisieren [ORTNER 1997:102].

Durch Beziehungen zu anderen Begriffen lassen sich im Lexikon vor allem Begriffsmerkmale beschreiben, die sich je nach Begriffsart unterscheiden. Das Metaschema wird deshalb nachfolgend um spezifische inhaltliche Vorgaben zur Spezifikation von Begriffsmerkmalen ergänzt, die entweder Informationsobjekte, Funktionen oder Prozesse charakterisieren. Dabei werden zugleich die zwischen den Begriffsarten erlaubten Beziehungsarten festgelegt, wodurch sich die normierte Grammatik und das im Folgenden erweiterte Metaschema ergeben. In dem in Abb. 4.12 gezeigten Ausschnitt sind dagegen zunächst nur solche Attribute in den Elementen BeteiligungVon und BeteiligungNach aufgeführt, die allen Beziehungsarten gemeinsam sind. Diese beschreiben, wie viele Gegenstände der jeweils in Bezug gesetzten Begriffe an einer Beziehung teilnehmen. Die mengenmäßige Beteiligung der Gegenstände wird durch ein Minimum und ein Maximum, das auch den Wert »beliebig viele« annehmen kann, beschrieben. Im Metaschema wird zunächst davon ausgegangen, dass an einer Beziehung im Normalfall jeweils genau eine Ausprägung teilnimmt. Die Beteiligung wird jedoch mit der jeweiligen Aussage explizit festgelegt: »*Ein* Artikel hat *einen* Namen und *eine* Artikelnummer«, »*Ein* Lagerbereich umfasst *einen bis beliebig viele* Lagerplätze«. In der Aussage wird nur ein Wert genannt, wenn die minimale und maximale Beteiligung identisch sind. Zu beachten ist dabei allerdings, dass die spezielle Aussage »*Ein* Lagerbereich umfasst *beliebig viele* Lagerplätze« ein Minimum von Null impliziert.

4.4.1.1 Informationsobjekte

Zwischen zwei oder mehreren Begriffen, deren Begriffswörter Dinge (Extension I) eines Anwendungsbereichs zu einer Dingklasse bzw. deren Beschreibungen (Extension II) zu einem Informationsobjekt (-typ) zusammenfassen, werden im Allgemeinen folgende Beziehungswirkungen durch die Einführung spezifischer Beziehungsarten voneinander unterschieden [SCHIENMANN 1997:189-191, 221-228; BECKER und SCHÜTTE 2004:161f.]: die *Inklusion* im Rahmen von Spezialisierungs-Generalisierungs-Beziehungen, die *Partizipation* bei Merkmalsvereinbarungen, die *Aggregation* im Rahmen von Teil-Ganzheits-Beziehungen, die *Konnexion* bei Begriffsverknüpfungen sowie die *Relation* im Rahmen von Assoziationsbeziehungen zwischen Begriffen.

Durch die Vereinbarung von *Spezialisierungs-Generalisierungs-Beziehungen* werden *begriffliche Unterordnungen* zwischen Begriffswörtern beschrieben [SCHIENMANN 1997:221; BECKER und SCHÜTTE 2004:161f.]. Sie basieren auf der *Abstraktion*, d.h. der partiellen Gleichheit von Begriffen. Mit ihnen lassen sich Unterordnungen festlegen, die sich entweder aus einer zeitlich veränderlichen Rollenbeziehung oder einer zeitlich unveränderlichen Art-Gattungs-Beziehung ergeben. Beispiele für eine auf einer Rollenbeziehung basierende Unterordnung sind Aussagen wie »Ein Lagerartikel ist ein spezieller Artikel« oder »Ein Verkaufsartikel ist ein spezieller Artikel«, während die Aussagen »Ein Lagerfestplatz ist ein spezieller Lagerplatz« und »Ein Kunde bzw. ein Lieferant ist ein spezieller Geschäftspartner« jeweils eine Art-Gattungs-Beziehung repräsentieren.

Die in der Sprachtheorie auch als Hyponomie-Beziehungen [LYONS 1972] bezeichneten Spezialisierungs-Generalisierungs-Beziehungen drücken aus, dass die untergeordneten Begriffe (die Hyponyme) den jeweils übergeordneten Begriff (das Hyperonym) intensional, also der Bedeutung nach einschließen [WESSEL 1976:44]. Für die Extensionen der Begriffe gilt dagegen umgekehrt, dass alle Gegenstände, die unter ein untergeordnetes Begriffswort fallen, auch unter das übergeordnete Begriffswort fallen. Über diesen prinzipiellen Zusammenhang hinaus lässt sich festlegen, dass auch jeder Gegenstand, der unter das übergeordnete Begriffswort fällt, unter eines der untergeordneten Begriffswörter fällt: »Ein Geschäftspartner ist ein Kunde oder ein Lieferant«. Die untergeordneten Begriffe stellen dann extensional eine *Partition* des übergeordneten Begriffs dar. So drückt die zuvor genannte Aussage aus, dass es außer Kunden und Lieferanten keine weiteren Geschäftspartner gibt.

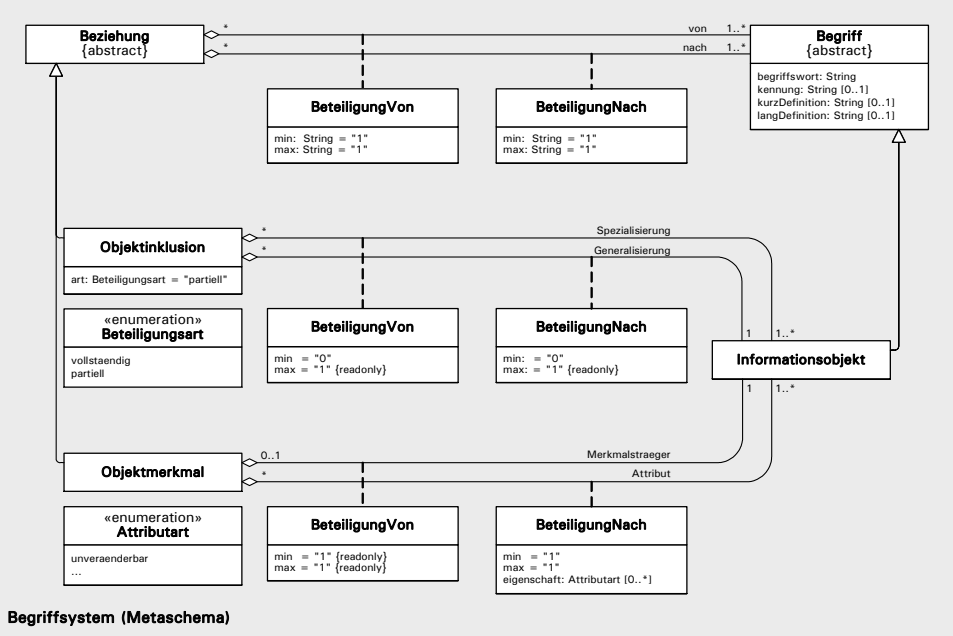


Abb. 4.13: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Begriffsbeziehungen.

Eine Begriffshierarchie zwischen Informationsobjekten wird im UNSCOM Metaschema durch eine mehrstellige Beziehung (Objektinklusion) dargestellt, die mindestens einem Hyponym, der jeweiligen *Spezialisierung*, ein Hyperonym, die *Generalisierung*, zuordnet (vgl. Abb. 4.13). Die einzelnen Spezialisierungen sind dabei durch ein exklusives logisches ›Oder‹ adjunktiv verknüpft und hinsichtlich ihrer Ausprägungen disjunkt [SCHIENMANN 1997:224]. Grundsätzlich handelt es sich bei Spezialisierungs-Generalisierungs-Beziehungen um 1:1-Beziehungen, die z.B. je *einen* Kunden (oder stattdessen einen Lieferanten) als *einen* speziellen Geschäftspartner identifizieren. Jede Ausprägung kann daher maximal an einer Beziehung teilnehmen, weshalb das zugehörige Attribut im Metaschema durch den Zusatz {readonly} vorab festgelegt ist. Über die mengenmäßige Beteiligung hinaus ist anhand der Beteiligungsart festzulegen, ob durch die Beziehung eine Partition begründet wird (vgl. Abb. 4.13). In diesem Fall werden von der Beziehung alle Gegenstände erfasst, die unter das generelle Begriffswort fallen. Die Ausprägungen der Generalisierung sind dann an der Beziehung *vollständig* beteiligt.

Durch *Merkmalsvereinbarungen* werden die essentiellen, konstituierenden Eigenschaften eines (Ding-) Begriffs beschrieben. Sie basieren auf der *Abhängigkeit* von Begriffen und drücken eine *begriffliche Zuordnung* aus. Merkmale werden dabei als Begriffe zweiter Ordnung betrachtet, die anderen Begriffen als Attribute zukommen [SCHIENMANN 1997:189]. Sie erscheinen bei Aussagen über einen Sachverhalt aus dem Anwendungsbereich ausschließlich gemeinsam mit dem zugehörigen Begriff erster Ordnung. Beispiele für Merkmalsvereinbarungen sind Aussagen wie »Ein Artikel hat einen Namen« oder »Ein Lagerplatz besitzt eine Tragfähigkeit«. Die Ausprägungen von Attributen werden üblicherweise durch Aufzählung oder durch Angabe eines Wertebereichs, der durch eine Maßeinheit ergänzt werden kann, festgelegt [SCHIENMANN 1997:191]: »Eine Tragfähigkeit ist eine in Gramm gemessene Zahl«. Die Wertebereiche und Maßeinheiten sind dabei im Metaschema durch entsprechende Vorgaben normiert (vgl. Abb. 4.12).

Im UNSCOM Metaschema werden Merkmalsvereinbarungen durch eine mehrstellige Beziehung (Objektmerkmal) dargestellt, die einem Begriff erster Ordnung, dem *Merkmalsträger*, mindestens einen Begriff zweiter Ordnung, das jeweilige *Attribut*, zuordnet (vgl. Abb. 4.13). Die Attribute sind dabei durch ein logisches ›Und‹ konjunktiv verknüpft. Auch bei Merkmalsvereinbarungen sind die mengenmäßigen Beteiligungen der Ausprägungen zumindest teilweise vorab festgelegt. So wird durch einen Attributwert jeweils genau eine Ausprägung eines Begriffs erster Ordnung charakterisiert: »*Ein* Artikel hat einen Namen«. Die mengenmäßige Beteiligung von Attributwerten kann indes schwanken, da ggf. mehrere (oder auch gar keine) Attributwerte benötigt werden, um eine Ausprägung eines Begriffs erster Ordnung zu charakterisieren: »Ein Artikel hat mehrere Preise«. Ferner lässt sich bei

der Vereinbarung von Merkmalen bspw. auch die Unveränderbarkeit von Attributwerten festlegen (vgl. Abb. 4.13).

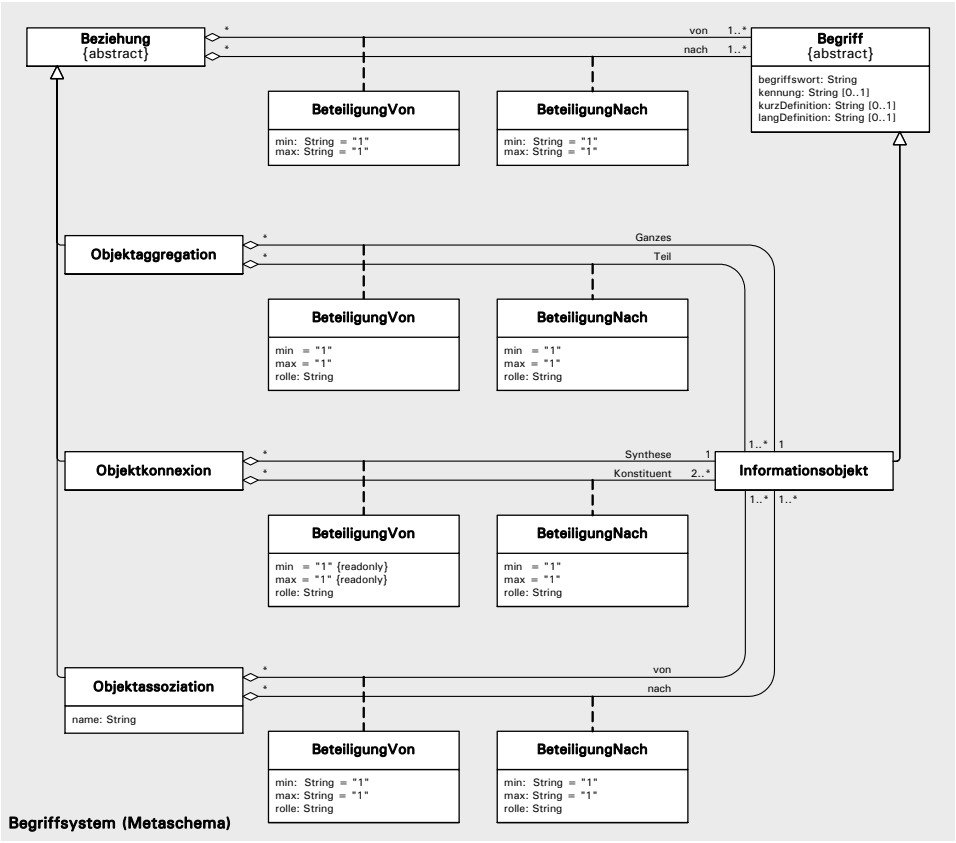


Abb. 4.14: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Begriffsbeziehungen (Forts.).

Mit Teil-Ganzheits-Beziehungen wird festgelegt, dass die Ausprägungen eines oder mehrerer Begriffe *Bestandteil* der Ausprägungen eines anderen Begriffs sind. Wie Merkmalsvereinbarungen basieren auch Teil-Ganzheits-Beziehungen auf der *Abhängigkeit* von Begriffen. Im Unterschied zu Merkmalsvereinbarungen werden jedoch *eigenständige* Begriffe erster Ordnung in einen kompositiven Zusammenhang gesetzt: »*Ein* Lager unterteilt sich in *mehrere* Lagerbereiche«. Der gebildete kompositive Zusammenhang stellt dabei eine Zusammenfassung dar [ORTNER 1997:25; SCHIENMANN 1997:195; BECKER und SCHÜTTE 2004:162f.]. Anders als bei einer Inklusion fallen die Gegenstände, die unter die Begriffswörter der Teile fallen, deshalb nicht unter das Begriffswort, das die Ganzheit repräsentiert [SCHIENMANN 1997:201f.]. Verbindungen von Teilen zu einem Ganzen werden im UNSCOM Metaschema jeweils durch eine mehrstellige Beziehung (Objektaggregation) dargestellt, die einem Begriff, der *Ganzheit*, mindestens einen Begriff, das *Teil*, zuordnet (vgl. Abb. 4.14). Die Teile sind dabei durch ein logisches ›Und‹ konjunktiv verknüpft und nehmen in der Beziehung ggf. spezifische *Rollen* ein. Über die mengenmäßige Beteiligung der

Ausprägungen kann indes keine allgemeingültige Festlegung mehr getroffen werden. So kann es im Gegensatz zu Attributen durchaus Ausprägungen von Teilen geben, die keiner Ausprägung (oder sogar mehreren Ausprägungen) einer Ganzheit zugeordnet sind.

Als spezielle Teil-Ganzheits-Beziehungen sind *Begriffsverknüpfungen* einzustufen, bei denen die Verbindung von Begriffswörtern erst zur Einführung eines neuen Begriffs führt: »Eine Kundenauftragsposition ist eine Verknüpfung von einem Kunden und einem Verkaufsartikel«. Durch eine solche Verknüpfung werden Beziehungen, die zwischen Dingen eines Anwendungsbereichs existieren, selbst wiederum als Gegenstände betrachtet und zu einem Dingbegriff zusammengefasst [BECKER und SCHÜTTE 2004:162]. Diese Verdinglichung erlaubt es, die speziellen Merkmale einer Inbeziehungsetzung durch Attribute zu beschreiben, die dann dem neu eingeführten Begriff zugeordnet werden [ORTNER 1997:25; SCHIENMANN 1997:227]. Im UNSCOM Metaschema wird eine Begriffsverknüpfung durch eine mehrstellige Beziehung (Objektkonnexion) dargestellt, die einem Begriff, der *Synthese*, mindestens zwei konjunktiv verknüpfte Begriffe, die *Konstituenten*, zuordnet. An der Beziehung nimmt stets eine Ausprägung des neu eingeführten Begriffs (»*Eine* Kundenauftragsposition«) teil, die die Merkmale der verdinglichten Beziehung beschreibt und dabei ggf. eine bestimmte *Rolle* einnimmt (vgl. Abb. 4.14). Die mengenmäßige Teilnahme der Ausprägungen der jeweiligen Konstituenten (»Kunde«, »Verkaufsartikel«), die an der verdinglichten Beziehung ebenfalls in verschiedenen Rollen partizipieren können, lässt sich hingegen nicht für alle Objektverknüpfungen vorab festlegen.

Assoziationsbeziehungen beschreiben schließlich in ihrer Wirkung nicht weiter geklärte *Abhängigkeiten* zwischen den Gegenständen, die unter die miteinander verknüpften Begriffswörter fallen [SCHIENMANN 1997:227; BECKER und SCHÜTTE 2004:162]. Sie werden im UNSCOM Metaschema jeweils durch eine mehrstellige Beziehung (Objektassoziation) dargestellt, wobei im Voraus weder der Name der Beziehung, noch die Rollen der Begriffe oder die Beteiligung ihrer Ausprägungen festgelegt werden kann: »Ein Lieferant ist Zulieferer mehrerer Artikel«. Die Vereinbarung einer Assoziationsbeziehung sollte deshalb nur erfolgen, wenn die anderen Beziehungsformen zur Darstellung ungeeignet sind.

Bevor zur Betrachtung der zwischen den Funktionen existierenden Beziehungsformen übergegangen wird, bleibt anzumerken, dass sich ein Sachverhalt bei der Fülle der eingeführten Beziehungsformen zumindest in drei Fällen auf unterschiedliche Weise erfassen lässt. Dies gilt erstens für Spezialisierungs-Generalisierungs-Beziehungen, die sich auf die zuvor beschriebene Weise oder durch die Vereinbarung eines Typattributs darstellen lassen [SCHIENMANN 1997:193]. Die letztgenannte Vorgehensweise empfiehlt sich jedoch nur dann, wenn in der Hierarchie keine spezifischen Merkmale zu betrachten sind und die Ver-

einbarung eines Typattributs folglich ausreicht. Daneben ist es zweitens möglich, konstituierende Merkmale eines Begriffs nicht als Merkmalsvereinbarungen, sondern mit Teil-Ganzheits-Beziehungen darzustellen. Diese Vorgehensweise ist einzuschlagen, wenn die Merkmale durch einen Begriff erster Ordnung angemessener repräsentiert sind. Aufschluss darüber kann neben der zu klärenden Eigenständigkeit des Merkmals vor allem die Beantwortung der Frage liefern, ob das Merkmal über Werte (Attribut) oder eine komplexe Intension (ggf. Teil) definiert ist [SCHIENMANN 1997:195]. Abschließend ist drittens festzuhalten, dass sich die Semantik von zwei- und mehrstelligen Beziehungen unterscheidet. So lässt sich vor allem die mengenmäßige Beteiligung von Ausprägungen der Ganzheit nur in zweistelligen Teil-Ganzheits-Beziehungen individuell (also pro Teil) festlegen. Mehrstellige Beziehungen sind deshalb ggf. in zweistellige aufzulösen [SCHIENMANN 1997:228].

4.4.1.2 Funktionen

Auch die Begriffe, deren Begriffswörter Handlungen (Extension I) eines Anwendungsbereichs zu einer Klasse bzw. deren Beschreibungen (Extension II) zu einer Funktion (einem Funktionstyp) zusammenfassen, lassen sich durch verschiedene Beziehungsformen mit anderen Begriffen in Zusammenhang setzen. Prinzipiell ist dabei zwischen Beziehungsformen, die Funktionen miteinander verbinden, und solchen, die einen Bezug zwischen Funktionen und Informationsobjekten herstellen, zu unterscheiden. Zu den Funktionsbeziehungen gehören Spezialisierungs-Generalisierungs-Beziehungen auf Basis der *Inklusion* und Teil-Ganzheits-Beziehungen, die auf der *Aggregation* basieren [SCHIENMANN 1997:241].

Diese beiden Beziehungsarten, auf die bereits im vorigen Abschnitt genauer eingegangen wurde, werden vor allem im Rahmen einer schrittweisen funktionalen Verfeinerung zur strukturierten Darstellung von Funktionen eingesetzt [WIRTH 1971]. Die strukturierte Darstellung basiert auf der Spezialisierung und der Zerlegung in Teilfunktionen: »Das Auslagern der Lagerartikel setzt sich aus dem Erstellen des Kommissionierplans, dem Kommissionieren der Lagerartikel, dem Erfassen der Lagerartikel und dem Buchen des Warenausgangs zusammen«. Im Gegensatz zu Informationsobjekten wird bei einer kompositiven Zerlegung jedoch üblicherweise keine Aussage über die mengenmäßige Beteiligung der Teilfunktionen gemacht. Im UNSCOM Metaschema ist diese bei der Funktionsaggregation deshalb zunächst als „unspezifiziert“ festgelegt (vgl. Abb. 4.15).

Anhand von Beziehungen zu Informationsobjekten lassen sich hingegen die Merkmale von Funktionen (bzw. Handlungen) beschreiben. Dabei sind folgende *Merkmalsarten* zu unterscheiden [SCHIENMANN 1997:241]: *Handlungssubjekte* führen Handlungen aus, *Handlungsobjekte* sind von der Ausführung der Handlung betroffen und *Handlungssituationen* beschreiben den Kontext der Ausführung durch Zustände, die jeweils vor bzw. nach der

Ausführung im Anwendungsbereich herrschen. Die Beschreibung der Zustände erfolgt dabei durch Prädikation über die betroffenen Handlungsobjekte. Unter ihnen lassen sich direkt betroffene Handlungsobjekte, die entweder primär durch die Handlung verändert (Objektfälle) oder als ihr Ergebnis erzeugt (Werkfälle) werden, von indirekt betroffenen Handlungsobjekten unterscheiden, die lediglich als Mittel zur Handlungsdurchführung (Mittelfälle) dienen [LORENZEN 1987:46]. Damit lassen sich Handlungen des Anwendungsbereichs bspw. wie folgt charakterisieren: »Ein Kunde…« (Handlungssubjekt) »…erteilt…« (Handlung) »… einen Kundenauftrag…« (Objektfall) »…anhand eines Internetkatalogs…« (Mittelfall) »…, falls für alle Verkaufsartikel gilt, dass deren Verfügbarkeit ‚lieferbar' ist« (Handlungssituation).

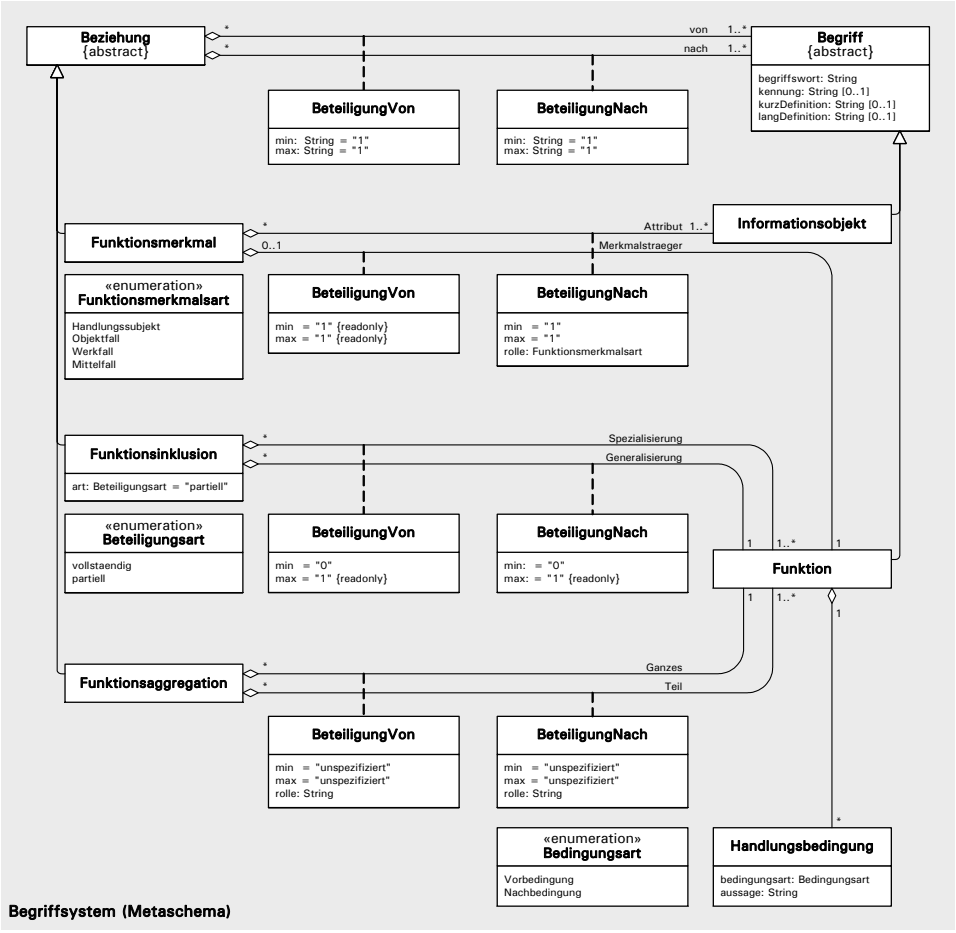


Abb. 4.15: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Begriffsbeziehungen (Forts.).

Im UNSCOM Metaschema wird die Vereinbarung von Funktionsmerkmalen als mehrstellige Beziehung (Funktionsmerkmal) dargestellt, die einer Funktion, dem *Merkmalsträger*, je ein Informationsobjekt in der Rolle als *Handlungssubjekt* und mindestens eines in der Rolle als *Handlungsobjekt* zuordnet (vgl. Abb. 4.15). Ist zur Charakterisierung der Funktion kein Verweis auf ein spezielles Handlungssubjekt notwendig, wird unter Wegfall dieses

Beziehungsteils »Irgendjemand« als Subjekt zugeordnet. Die Rolle des Handlungsobjekts ist bei der Inbeziehungsetzung gemäß der zuvor genannten Einteilung als Objekt-, Werk- oder Mittelfall festzulegen, wobei jeder Funktion exakt ein primäres Handlungsobjekt als Objektfall zuzuordnen ist. Hingegen lässt sich die mengenmäßige Beteiligung der einzelnen Ausprägungen nicht allgemeinverbindlich vorab festlegen, wie folgende Teilaussagen zeigen: »…erteilt *einen* Kundenauftrag…«, »…kommissioniert *mehrere* Lagerartikel…«.

Zur Charakterisierung der Handlungssituation sind schließlich Aussagen auf Basis der *Prädikatenlogik* erster Stufe zu formulieren. Sie legen jeweils vor bzw. nach der Ausführung zutreffende Eigenschaften von Ausprägungen der beteiligten Handlungsobjekte fest (z.B. »Verfügbarkeit des Artikels ist ‚lieferbar'«). Damit ist auch die Grammatik der Prädikatenlogik Bestandteil der normierten Grammatik des UNSCOM Spezifikationsrahmens. Wegen ihrer speziellen Anwendung zur Beschreibung der Handlungssituation werden die mit dieser Grammatik festgelegten Beziehungen im Metaschema jedoch nicht mehr weiter unterschieden. Stattdessen werden die prädikatenlogischen Aussagen als Ganzes der zugehörigen Handlung in Form einer *Vor- oder Nachbedingung* zugeordnet (vgl. Abb. 4.15). Durch die Inbeziehungsetzung einer Handlung mit Vor- oder Nachbedingungen ergibt sich ein subjunktiver Zusammenhang der Form »Falls die Vorbedingungen erfüllt sind, wird die Handlung ausgeführt« bzw. »Falls die Handlung mit erfüllten Vorbedingungen ausgeführt wurde, werden die Nachbedingungen erfüllt« [LORENZEN 1987:71f.]. Die Vor- und Nachbedingungen sind dabei jeweils durch ein logisches ›Und‹ konjunktiv verknüpft.

4.4.1.3 Prozesse

Ein Prozess (-typ) setzt Handlungen (Extension I) bzw. deren Beschreibungen (Extension II) zu einem schematischen Ablauf zusammen und bezeichnet diesen mit einem Begriffswort. Die Zusammensetzung, die selbst wiederum eine komplexe Handlung darstellt, erfolgt durch die Vereinbarung von Reihenfolgebeziehungen. Diese legen eine zeitliche *Präzedenzstruktur*, d.h. eine Vorrangstruktur zwischen den zum Prozess gehörenden Elementen fest. Durch entsprechende Reihenfolgebeziehungen lässt sich bspw. der Ablauf »Auslagern der Lagerartikel« bei der Abwicklung eines Kundenauftrags beschreiben: »Nach dem Erstellen des Kommissionierplans folgen das Kommissionieren der Lagerartikel, dann das Erfassen der Lagerartikel und schließlich das Buchen des Warenausgangs«.

Die bei der Beschreibung des Ablaufs verwendeten *Reihenfolgebeziehungen* stellen spezielle kompositive Beziehungen dar, die sich aus den zwischen den Handlungen existierenden *kausalen Abhängigkeiten* ergeben [SCHIENMANN 1997:268]. Von anderen Prozessen wird ein Prozess dabei durch das ihn prägende *Handlungsobjekt* (»Lagerartikel«) abgegrenzt, das ein wichtiges Prozessmerkmal darstellt [BECKER und SCHÜTTE 2004:150].

Anders als Interaktionsprotokolle, die einen Ablauf aus Sicht eines beteiligten Handlungssubjekts beschreiben, orientieren sich Prozesse somit stets an einem Objekt der Verrichtung und dessen Wandlungen, die durch die Ausführung zusammengehöriger Handlungen eintreten. Als weitere Prozessmerkmale sind eine *Startaktivität* sowie mindestens eine *Schlussaktivität*, die jeweils das Erreichen eines Endzustands beschreibt, festzulegen.

Zur Beschreibung der Präzedenzstruktur werden Handlungen miteinander in Beziehung gesetzt, die als *Aktivitäten* die elementaren Elemente eines Prozesses bilden. Eine Aktivität besitzt aufgrund der eingenommenen ablauforientierten Perspektive gegenüber einer Handlung zusätzliche Merkmale. Zu diesen Merkmalen zählt ein *Statusattribut* „Ausführung", das die Werte »durchzuführen« und »abgeschlossen« annehmen kann. Darüber hinaus gehört zu jeder Aktivität ein auslösendes *Ereignis*, dessen Auftreten die Ausführung der Handlung während eines Ablaufs bewirkt [BECKER und SCHÜTTE 2004:151]. Das Ereignis ist durch aussagenlogische Prädikation über die beteiligten Handlungsobjekte und die Aktivitäten des Prozesses zu beschreiben. Folgen zwei Aktivitäten unmittelbar aufeinander, so besteht das auslösende Ereignis der Folgeaktivität lediglich aus der Aussage »Vorgänger ist abgeschlossen«. Außer auf die elementaren Aktivitäten wird bei der Beschreibung der Präzedenzstruktur auch Bezug auf weitere Prozesselemente genommen, die jeweils eine Menge von Aktivitäten zu einem Teilablauf gruppieren. Zu diesen komplexen Prozesselementen gehören Schnittstellen zu anderen *Prozessen* sowie zeitliche *Verknüpfungen*, die als Junktoren andere Prozesselemente entsprechend einer gemeinsamen temporalen Vorgabe zu Teilabläufen verbinden [BECKER und SCHÜTTE 2004:151]. Als Bestandteil eines solchen Teilablaufs kann schließlich auch die *Leeraktivität* auftreten.

Im UNSCOM Metaschema wird ein spezifizierter Prozess durch das gleichnamige Schemaelement dargestellt, das das *Prozessobjekt* und die *Prozesselemente* festlegt (vgl. Abb. 4.16). Die durch den Prozess festgelegte *Präzedenzstruktur* wird durch den zeitlichen *Ablauf*, d.h. die Abfolge der Prozesselemente beschrieben. Als Prozesselemente dürfen *Aktivitäten* und *Schnittstellen* zu anderen Prozessen auftreten, die im Metaschema durch die Elemente Aktivitaet bzw. Prozessschnittstelle dargestellt werden und jeweils einen Verweis auf die zugrunde liegende Handlung enthalten. Darüber hinaus treten neben der Start- und den Schlussaktivitäten vor allem *Verknüpfungen* als Prozesselemente auf, die im Metaschema durch das Element Verknuepfung dargestellt werden. Jede Verknuepfung wird in *Präfixnotation* (z.B. als Sequenz(Kommissionierplan erstellen, Lagerartikel kommissionieren, Lagerartikel erfassen, Warenausgang buchen)) gespeichert und verbindet ein Prozesselement mit einem oder mehreren Prozesselementen zu einem *Teilablauf* (vgl. Abb. 4.16). Die verwendete Präfixnotation stellt dabei zugleich die korrekte Schachtelung von Verknüpfungen sicher. Prinzipiell ist zwischen *konjunktiven* und *adjunktiven* Verknüpfun-

gen zu unterscheiden [BECKER und SCHÜTTE 2004:151f.]: zu den konjunktiven Beziehungen gehören sequenzielle (geordnete) und parallele Abläufe, während inklusive und exklusive Verzweigungen zu den adjunktiven Beziehungen zählen. Bei der Verbindung zu Teilabläufen wird zudem angenommen, dass jedes dargestellte Prozesselement exakt einmal auszuführen ist, sobald es durch den Kontrollfluss erreicht wird. Die mengenmäßige Beteiligung ist deshalb für alle Ausprägungen der beteiligten Prozessobjekte vorab festgelegt.

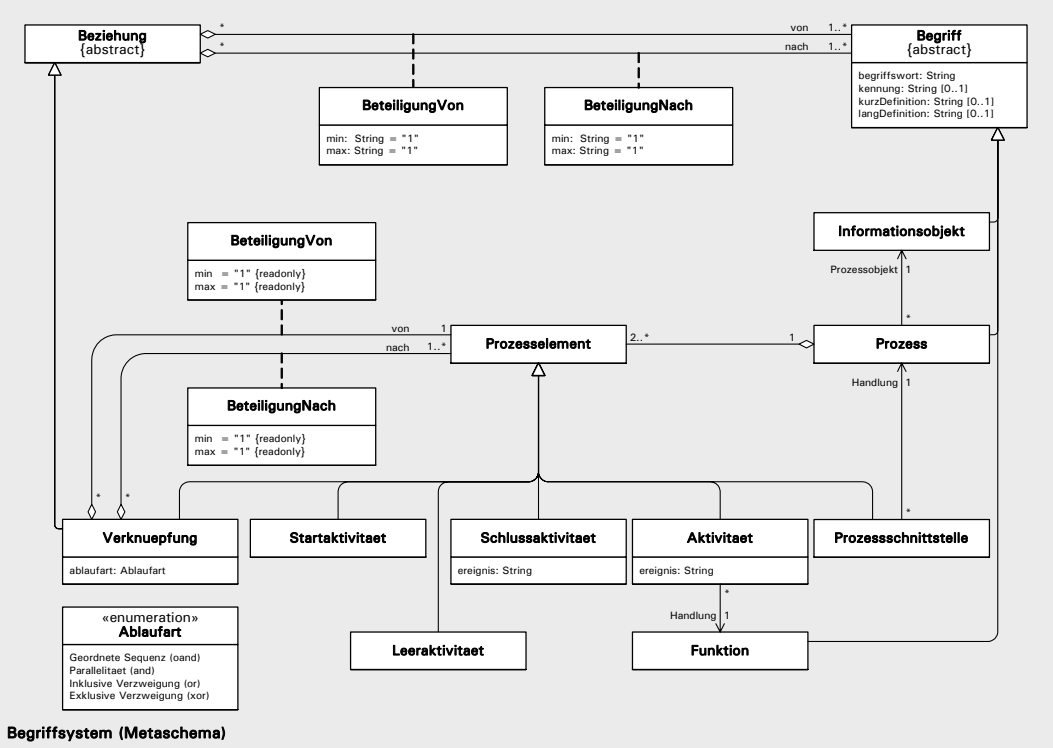


Abb. 4.16: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Begriffsbeziehungen (Forts.).

Bei der Konzeption des Metaschemas wurde vor allem das Ziel einer möglichst *anschaulichen Darstellung* des Kontrollflusses verfolgt. Die ablauforientierte Darstellung von Präzedenzstrukturen basiert deshalb auf den Konzepten der *Ereignisgesteuerten Prozesskette*, die die gewünschte hohe Anschaulichkeit gewährleisten [BECKER und SCHÜTTE 2004:110-114]. Insbesondere für Simulationszwecke ist jedoch eine Darstellung der festgelegten Abläufe anzustreben, die auf den alternativen Konzepten der *Petri-Netze* basiert. Bei der Weiterentwicklung des UNSCOM Spezifikationsrahmens wird deshalb die Einbindung weiterer inhaltlicher Vorgaben angestrebt, die eine eindeutige Umsetzung zwischen den beiden Konzepten gewährleisten. Die Erarbeitung der hierzu benötigten Vorgaben ist derzeit jedoch noch nicht abgeschlossen und Gegenstand wissenschaftlicher Untersuchungen.

Anzumerken bleibt, dass die bislang diskutierte Spezifikation von vollständigen *Abläufen* nicht die einzige Möglichkeit zur Beschreibung zeitlicher Präzedenzstrukturen darstellt.

Hierzu reicht es bereits aus, lediglich die *kausalen Abhängigkeiten* der einzelnen Aktivitäten als Einschränkungen zu beschreiben. Aus den beschriebenen Einschränkungen ergeben sich dann die zulässigen Abläufe. Anstelle des zu Beginn dargestellten Ablaufs ließe sich bspw. spezifizieren: »Das Kommissionieren der Lagerartikel folgt nach dem Erstellen des Kommissionierplans«, »Das Erfassen der Lagerartikel folgt nach dem Kommissionieren der Lagerartikel«, »Das Buchen des Warenausgangs folgt nach dem Erfassen der Lagerartikel«. Eine Beschreibung der bestehenden kausalen Abhängigkeiten lässt sich in der Regel leichter erreichen als eine Spezifikation vollständiger Abläufe. Kausale Abhängigkeiten können im UNSCOM Metaschema als Verknüpfungen (Verknuepfung) dargestellt werden, die jeweils zwei Aktivitäten zu einer Sequenz verbinden. Zusammen mit dem auslösenden Ereignis des Nachfolgers ist in diesem Fall zusätzlich festzulegen, ob dieser unmittelbar, irgendwann oder unter gewissen anderen Umständen nach dem Vorgänger auszuführen ist.

Zur formalen Spezifikation derartiger zeitlicher Bedingungen ist allerdings die für die Beschreibung von Ereignissen genutzte Prädikatenlogik mit einer *temporalen Logik* (vgl. [MANNA und PNUELI 1991]) zu erweitern. Temporale Logiken stellen spezielle Sprachelemente für die Beschreibung zeitlicher Bedingungen zur Verfügung. Ein Vorschlag zur Erweiterung der Prädikatenlogik, auf den bspw. im Rahmen einer Erweiterung der bislang vorgestellten Konzepte zurückgegriffen werden könnte, wird in [TUROWSKI 2003:127-136] vorgestellt. Ungeachtet der höheren Komplexität und der (beim Vorhandensein vieler alternativer Stränge) schlechteren Lesbarkeit ist zur Darstellung von Präzedenzstrukturen mit dem UNSCOM Spezifikationsrahmen jedoch die Spezifikation von *Abläufen* anzustreben. Diese Darstellung ist der Angabe kausaler Abhängigkeiten vor allem im Hinblick auf die Verlässlichkeit überlegen, da ein (irrtümliches) Weglassen eines Teilablaufs im Gegensatz zum Weglassen einer Abhängigkeit nicht dazu führen kann, dass auch ungültige Abläufe in der Beschreibung der Präzedenzstruktur zugelassen werden.

4.4.2 Repräsentation

Hinsichtlich der Repräsentation der in den Blue Pages spezifizierten Informationen ist zwischen den Aussagen, die Bestandteil der Aussagensammlung bzw. des Definitionsteils der Begriffe im Lexikon sind, und den übrigen Bestandteilen des Lexikons zu unterscheiden. Während die *Aussagen* Begriffe auf der Basis der zuvor standardisierten Grammatik in Beziehung setzen, um die der Informationsverarbeitung zugrunde liegenden fachlichen Zusammenhänge zu beschreiben, dienen die übrigen Bestandteile der *Lexikoneinträge* als Nachschlagewerk und klären die Verwendung einzelner Begriffswörter. Wie bei Catalysis wird das Lexikon deshalb im UNSCOM/T und UNSCOM/G Format in Form eines *Glossars*

repräsentiert, das in Tabellenform darzustellen ist [D'SOUZA und WILLS 1999:199, 548]. Die Tabellenstruktur des in Abb. 4.17 beispielhaft dargestellten Lexikons ergibt sich dabei direkt aus den Festlegungen des Metaschemas (vgl. Abb. 4.12). Die Repräsentation des Lexikons im UNSCOM/X Format wird durch das in Anhang B aufgeführte XML Schema festgelegt.

Terminologie (Typ: Lexikon)
Begriff (Typ: Informationsobjekt)
Begriffswort: Verkaufsartikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Verkaufsartikel ist ein Artikel, der vom Handelsunternehmen verkauft wird.
Illustration: (Typ: Illustration)
Illustrationstyp: (Typ: Beispiel)
Name: Buch »Component Software« von Clemens Szyperski et al.
Illustration: (Typ: Illustration)
Illustrationstyp: (Typ: Beispiel)
Name: Buch »Handelsinformationssysteme« von Jörg Becker und Reinhard Schütte
Begriff (Typ: Funktion)
Begriffswort: Bestandsveränderung buchen
Kurzdefinition: Durch das Buchen einer Bestandsveränderung wird eine Veränderung im Bestand eines Lagerartikels erfasst.
Langdefinition: Die Buchung einer Bestandsveränderung erfasst eine Veränderung im Bestand eines Lagerartikels. Bei dieser Veränderung kann es sich um eine Bestandsminderung (Warenausgang) oder eine Bestandsmehrung (Wareneingang) handeln.
Begriff (Typ: Prozess)
Begriffswort: Lagerartikel auslagern
Kurzdefinition: Das Auslagern von Lagerartikeln umfasst alle logistischen Aktivitäten zur Bearbeitung eines Kundenauftrags.

Abb. 4.17: Repräsentation des Lexikons in UNSCOM/T und UNSCOM/G.

Bei Bedarf ist in der in Abb. 4.17 beispielhaft dargestellten Tabelle für jeden Begriff eine zusätzliche Rubrik mit der Bezeichnung Definiens zu ergänzen. In dieser Rubrik sind die Aussagen aufzuführen, die unter Verwendung der zuvor vorgestellten Grammatik spezifiziert wurden und als Definiens in die jeweilige Begriffsdefinition eingehen. Bei der Spezifikation der Komponentenfunktionalität sind diese Aussagen jedoch in der Regel zugleich Bestandteil der Aussagensammlung und deshalb dort aufzuführen. Im Lexikon sind somit lediglich solche Aussagen zu repräsentieren, die nicht Bestandteil der Aussagensammlung sind.

Die Darstellung der in der Aussagensammlung definierten begrifflichen Beziehungen erfolgt im Rahmen der verschiedenen Formate auf jeweils unterschiedliche Weise. Im UNSCOM/G Format werden die Aussagen analog zur Vorgehensweise bei Catalysis (Business Model) und UML Components (Business Concept Model) als *graphisches Modell* dargestellt [D'SOUZA und WILLS 1999:198f., 548f.; CHEESMAN und DANIELS 2001:43f.]. Zur Darstellung wird dabei auf die Modellierungskonstrukte der *UML 2.0* [OMG 2003b] zu-

rückgegriffen, aus denen durch die Formatvorgaben des UNSCOM Spezifikationsrahmens eine geeignete Untermenge ausgewählt und hinsichtlich der Semantik präzisiert wird.

Zur Repräsentation von Aussagen in UNSCOM/T sind hingegen wie bei der Verwendung der Gebrauchssprache *Sätze* zu formulieren. Die Formulierung der Sätze erfolgt in UNSCOM/T jedoch auf Basis einer *Normsprache*, die standardisierte *Satzbaupläne* und einen festgelegten *Wortschatz* für die Bildung formal eindeutiger Sätze vorgibt [SCHIENMANN 1997:15]. Durch diese beiden Vorgaben vermeidet sie die Inkonsistenzen und Vagheiten, die sich bei Verwendung der Gebrauchssprache ergeben würden, und ermöglicht den Aufbau einer Ontologie. Anderen Ontologiedefinitionssprachen wie bspw. dem *Resource Description Framework* (RDF [LASSILA und SWICK 1999]) oder *Topic Maps* [WIDHALM und MÜCK 2002] ist die Normsprache dabei durch ihre bessere Lesbarkeit überlegen, die auch eine Auswertung des beschriebenen Sachverhalts durch Fachexperten zulässt.

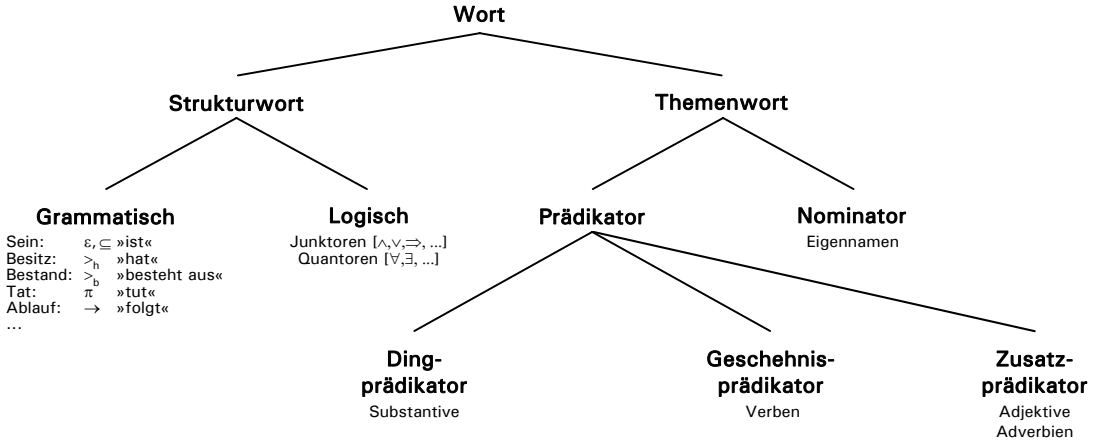


Abb. 4.18: Normsprachlicher Wortschatz (in Anlehnung an [SCHIENMANN 1997:150, 160]).

Im normsprachlichen Wortschatz werden analog zur Gegenstandseinteilung der indoeuropäischen Sprachen zunächst sog. Struktur- und Themenwörter unterschieden [LORENZEN 1987:52]. *Themenwörter* bezeichnen als Prädikatoren Gegenstände des Anwendungsbereichs, die durch das jeweils festzulegende Lexikon normiert und bei Aussagen ohne Flexion (in der Grundform) verwendet werden [SCHIENMANN 1997:157]. Wie in Abb. 4.18 dargestellt, lassen sich dabei Ding- und Geschehnisprädikatoren, die die Gegenstände in ihrer Hauptsache als Dinge bzw. Geschehnisse bezeichnen, sowie Zusatzprädikatoren, die weitere Eigenschaften von Gegenständen bezeichnen, unterscheiden. Strukturwörter (Partikel) werden dagegen zur Bildung von Aussagen verwendet und besitzen eine Bedeutung, die unabhängig vom Anwendungsbereich normiert ist. Ihre Bedeutung ist entweder wahrheitsfunktional oder strukturell (vgl. Abb. 4.18). Mit dem so festgelegten Wortschatz lässt sich nicht nur die Struktur normsprachlicher Sätze als Satzbauplan vorgeben. Gleichzeitig

lassen sich grundsätzliche *Regeln für die Prädikation von Begriffen* ableiten, die für alle Repräsentationsformate verbindlich sind. So sind Informationsobjekte (Dinge) durch Substantive und Funktionen bzw. Prozesse (Geschehnisse) durch Verben zu bezeichnen [LORENZEN 1987:52; D'SOUZA und WILLS 1999:549]. Zur besseren Verständlichkeit ist die Bezeichnung eines Geschehnisses dabei um das primäre Handlungsobjekt zu ergänzen [BECKER und SCHÜTTE 2004:154].

Auf diesen allgemeinen Regeln für die Prädikation aufbauend werden im Folgenden die Formatelemente vorgestellt, die zur Repräsentation der Beziehungen zwischen Begriffen in UNSCOM/T und UNSCOM/G verwendet werden. In UNSCOM/T sind die als normsprachliche Sätze repräsentierten Beziehungen zu einer Tabelle mit dem Titel „Aussagensammlung“ (bzw. zu einem Tabelleneintrag mit dem Titel „Definiens“ im Lexikon) zu gruppieren, während das graphische Modell in UNSCOM/G als gleichnamiges UML Paket zusammenzufassen ist. Grundsätzlich ist bei der Repräsentation zwischen Beziehungen zu unterscheiden, die Informationsobjekte, Funktionen oder Prozesse charakterisieren. Nachfolgend werden die jeweils zueinander korrespondierenden Elemente des UNSCOM/T und UNSCOM/G Formats gegenübergestellt, um deren Äquivalenz aufzuzeigen. In UNSCOM/X werden die spezifizierten Aussagen durch ein *XML Dokument* dargestellt, das gemäß dem in Anhang B gezeigten XML Schema zu erstellen ist.

4.4.2.1 Informationsobjekte

Spezialisierungs-Generalisierungs-Beziehungen zwischen Informationsobjekten lassen sich in UNSCOM/T durch das normsprachliche Strukturwort $\subseteq$ (»ist«) ausdrücken, das ein oder mehrere adjunktiv verknüpfte Spezialisierungen mit einer Generalisierung verbindet [LORENZEN 1987:188]. Hieraus ergibt sich der Satzbauplan »Ein(e) *Informationsobjekt*$_1$ (oder ein(e) *Informationsobjekt*$_2$)* ist ein(e) *Informationsobjekt*$_3$«, der Rollen- und Art-Gattungs-Beziehungen in normierter Weise darstellt [SCHIENMANN 1997:223]: »Ein Lagerartikel oder ein Verkaufsartikel ist ein Artikel« bzw. »Ein Kunde oder ein Lieferant ist ein Geschäftspartner«. Zusätzlich lässt sich durch den Satzbauplan »Ein(e) *Informationsobjekt*$_3$ ist ein(e) *Informationsobjekt*$_1$ (oder ein(e) *Informationsobjekt*$_2$)*« feststellen, dass jede Ausprägung der Generalisierung unter eine der Spezialisierungen fällt: »Ein Geschäftspartner ist ein Kunde oder ein Lieferant«.

Zur graphischen Darstellung von Beziehungen zwischen Informationsobjekten wird in UNSCOM/G ein Klassendiagramm verwendet, bei dem jedes UML Klassenkonstrukt mit einem Stereotyp «information object» zu kennzeichnen ist. Eine Spezialisierungs-Generalisierungs-Beziehung wird demzufolge durch eine UML Spezialisierung zwischen den jeweils beteiligten Informationsobjektklassen dargestellt (vgl. Abb. 4.19). Dabei lässt sich

durch das optionale Beziehungsattribut {complete} für den Fall einer Partition festhalten, dass jede Ausprägung der Generalisierung unter eine der Spezialisierungen fällt [OMG 2003b:122f.].

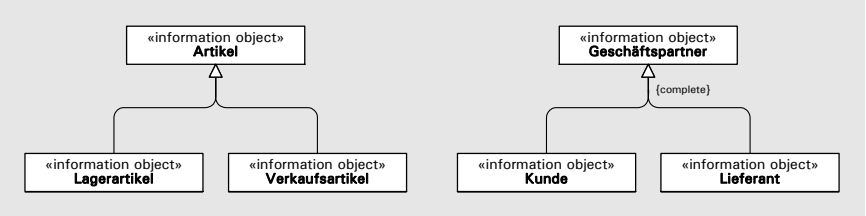


Abb. 4.19: Spezifikation der Spezialisierungs-Generalisierungs-Beziehungen in UNSCOM/G.

Merkmale von Informationsobjekten lassen sich in UNSCOM/T durch eine normsprachliche Partizipationsbeziehung ausdrücken, die unter Verwendung des Strukturworts $>_h$ (»hat«) gebildet wird [SCHIENMANN 1997:189]. Dabei werden einem Prädikator als Repräsentanten eines Begriffs erster Ordnung ein oder mehrere Prädikatoren zugeordnet, die Begriffe zweiter Ordnung repräsentieren. Der normierte Satzbauplan zur Darstellung dieser Beziehung lautet demgemäß »Ein(e) *Informationsobjekt*$_1$ hat einen / eine / ein *Informationsobjekt*$_2$ (und einen / eine / ein *Informationsobjekt*$_3$)*« [SCHIENMANN 1997:190]. Damit lässt sich bspw. aussagen: »Ein Lagerplatz hat eine Tragfähigkeit und einen Ort« und »Ein Verkaufsartikel hat eine Verfügbarkeit«. Die unbestimmten Artikel sind dabei durch ein Minimum und ein Maximum zu ersetzen, falls das Informationsobjekt durch mehrere Attributwerte charakterisiert wird. Darüber hinaus lässt sich durch einen weiteren Satzbauplan aussagen, dass die Werte des vereinbarten Attributs unveränderlich sind (vgl. hierzu die vollständige Darstellung in Abb. 4.23).

Die Ausprägungen eines Attributs lassen sich entweder durch Aufzählung oder die Angabe eines Wertebereichs festlegen. Eine entsprechende Aussage lässt sich normsprachlich unter Verwendung des Strukturworts ε (»ist«) und des Satzbauplans »Ein(e) *Informationsobjekt*$_1$ ist ((ein(e) *Wertebereich* [und wird gemessen in *Maßeinheit*]) | (*Merkmalswert*$_1$ (oder *Merkmalswert*$_2$)*)« ausdrücken, wobei die Merkmalswerte normsprachlichen Zusatzprädikatoren (Adjektiven, Adverbien) entsprechen. Unter Nutzung standardisierter Vorgaben für den Wertebereich und die Maßeinheit lässt sich bspw. aussagen »Eine Tragfähigkeit ist eine Ganzzahl und wird gemessen in Gramm«, »Eine Verfügbarkeit ist ‚lieferbar' oder ‚vergriffen'«. Strukturierte Attribute sind dagegen unter Verwendung des normsprachlichen Strukturworts $>_z$ (»ist zusammengesetzt aus«) und des Satzbauplans »Ein(e) *Informationsobjekt* ist zusammengesetzt aus einem / einer *Element*$_1$ (und einem / einer *Element*$_2$)$^+$« zunächst in ihre Bestandteile zu zerlegen [SCHIENMANN 1997:195]. Anschließend sind die Werte der Bestandteile unter Verwendung der genannten Satzbaupläne darzustellen. So gilt

für den Ort eines Lagerplatzes »Ein Ort ist zusammengesetzt aus einer Gangnummer und einer Regalnummer und einer Ebenennummer«, »Eine Gangnummer ist eine Zeichenkette«.

Im Rahmen der graphischen Repräsentation in UNSCOM/G sind die vereinbarten Merkmale jeweils als Attribute der im Diagramm enthaltenen Informationsobjektklasse darzustellen. Die Vereinbarung eines Attributs erfolgt durch die Nennung des Prädikators des Begriffs zweiter Ordnung in der Attributregion des UML Klassenkonstrukts (vgl. Abb. 4.20). Jedes Attribut kann dabei entweder durch ein Minimum und ein Maximum ([min..max]) oder eine feste Mengenangabe ([Menge]) ergänzt werden, die die Anzahl der für eine Charakterisierung jeweils benötigten Attributwerte festlegen [OMG 2003b:43, 64]. Der normsprachliche Ausdruck »beliebig viele« wird dabei im Diagramm durch einen * dargestellt. Darüber hinaus lässt sich durch die Angabe der optionalen Eigenschaft {readonly} festhalten, dass die Werte eines Attributs unveränderlich sind [OMG 2003b:64].

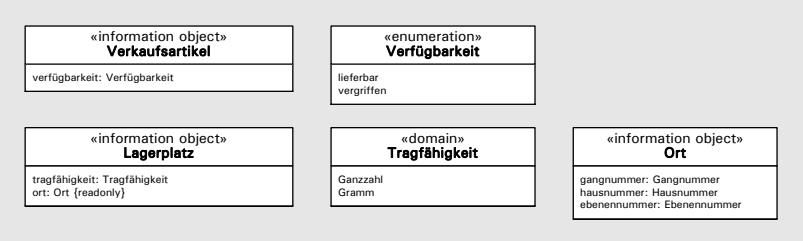


Abb. 4.20: Spezifikation der Merkmalsvereinbarungen in UNSCOM/G.

Die Aufzählung von Attributwerten erfolgt im Rahmen einer speziellen UML Aufzählung [OMG 2003b:96f.], die mit dem Stereotyp «enumeration» gekennzeichnet und dem Begriffswort des Attributs benannt wird. Der Name der UML Aufzählung ist zugleich als Typ des Attributs zu hinterlegen (vgl. Abb. 4.20). Die Vereinbarung eines Wertebereichs erfolgt hingegen unter Verwendung einer spezialisierten UML Aufzählung, die mit dem Stereotyp «domain» zu kennzeichnen ist. Diese beinhaltet neben dem Wertebereich ggf. auch die zugehörige Maßeinheit. Zur Darstellung eines strukturierten Attributs ist schließlich eine Informationsobjektklasse zu verwenden, die seine einzelnen Bestandteile wiederum als *Attribute* aufführt (vgl. Abb. 4.20).

Mit dem normsprachlichen Strukturwort $>_b$ (»besteht aus«) und dem Satzbauplan »Ein(e) *Informationsobjekt*$_1$ besteht aus einem / einer *Informationsobjekt*$_2$ (und einem / einer *Informationsobjekt*$_3$)*« sind Teil-Ganzheits-Beziehungen zwischen eigenständigen Informationsobjekten in UNSCOM/T darzustellen [SCHIENMANN 1997:225f.]. Dabei sind die unbestimmten Artikel ggf. durch ein Minimum und ein Maximum zu ersetzen, um eine speziel-

le mengenmäßige Beteiligung der Teile festzulegen: »Ein Lagerbereich besteht aus 1 bis beliebig vielen Lagerplatz«. Mit einem weiteren Satzbauplan lässt sich außerdem angeben, dass ein Informationsobjekt Bestandteil mehrerer Ganzheiten ist bzw. dass es Teile gibt, die zu keiner Ganzheit gehören. Schließlich ist der zuvor vorgestellte Satzbauplan noch um Rollenangaben für die Ganzheit und die Teile zu erweitern. Die zur Beschreibung von Aggregationen zu nutzenden Satzbaupläne sind in Abb. 4.23 ausführlich dargestellt.

In UNSCOM/G ist eine Teil-Ganzheits-Beziehung als UML Aggregation zwischen zwei oder mehreren Informationsobjektklassen darzustellen (vgl. Abb. 4.21). Dabei sind die optionalen Rollennamen und die ebenfalls optionalen mengenmäßigen Beteiligungen direkt den jeweils beteiligten Informationsobjekten zuzuordnen [OMG 2003b:83]. Zu beachten ist, dass beim Weglassen einer mengenmäßigen Beteiligung die *Standardvorgabe* des Metaschemas gilt. Bei einer mehrfachen Aggregation kann zudem nur eine Beteiligung für die Ganzheit festgelegt werden, die für *alle* Teilbeziehungen gilt. Schwankt die Beteiligung der Ganzheit je nach Teil, ist die mehrfache Aggregation deshalb in mehrere Beziehungen mit jeweils unterschiedlicher Beteiligung der Ganzheit aufzuteilen. Eine mehrfache UML Aggregation ist in Abb. 4.21 rechts beispielhaft dargestellt. Zur ausführlichen Darstellung vgl. Abb. 4.23.

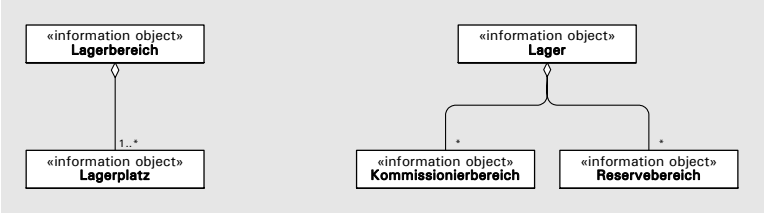


Abb. 4.21: Spezifikation der Teil-Ganzheits-Beziehungen in UNSCOM/G.

Als spezielle Teil-Ganzheits-Beziehungen werden Konnexionen in UNSCOM/T durch das normsprachliche Strukturwort $>_v$ (»verknüpft«) beschrieben und gemäß dem Satzbauplan »Ein(e) *Informationsobjekt*$_1$ verknüpft einen / eine / ein *Informationsobjekt*$_2$ (und einen / eine / ein *Informationsobjekt*$_3$)$^+$« dargestellt [SCHIENMANN 1997:227]. Der dargestellte Satzbauplan ist noch zur Darstellung der Rollen, die von den Informationsobjekten ggf. eingenommen werden, zu erweitern Wie schon bei der Teil-Ganzheits-Beziehung wird außerdem ein weiterer Satzbauplan benötigt, damit eine mengenmäßige Beteiligung der Konstituenten festgelegt werden kann, die von der Standardvorgabe des Metaschemas abweicht: »Ein Kunde setzt beliebig viele Kundenauftragsposition zusammen«, »Ein Verkaufsartikel setzt beliebig viele Kundenauftragsposition zusammen«. Die damit zur Beschreibung von Konnexionen insgesamt zu verwendenden Satzbaupläne sind in Abb. 4.23 zusammengefasst.

Konnexionen sind in UNSCOM/G jeweils durch eine UML Assoziation darzustellen, der eine Assoziationsklasse (durch eine gestrichelte Verbindung) hinzugefügt ist. Die Assoziationsklasse entspricht der zu einem Informationsobjekt verdinglichten Beziehung und beschreibt deren jeweilige Attribute [OMG 2003b:118f.]. Besteht die Konnexion im einfachen Fall aus zwei Konstituenten (vgl. Abb. 4.22), so ist für die zugrunde liegende Beziehung eine zweistellige UML Assoziation in Form einer durchgezogenen Verbindung in das Diagramm aufzunehmen [OMG 2003b:82]. Bei drei oder mehr Konstituenten ist diese durch eine mehrstellige Assoziation zu ersetzen, die mit einer Raute gekennzeichnet wird [OMG 2003b:82]. Zur Darstellung komplexer Konnexionen in UNSCOM/G vgl. Abb. 4.23.

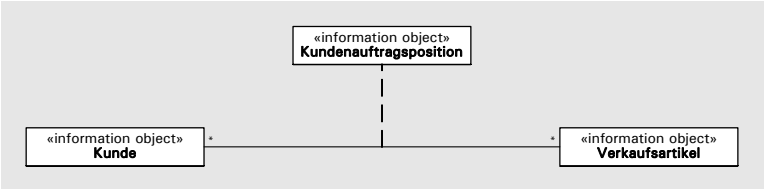


Abb. 4.22: Spezifikation der (einfachen) Konnexionen in UNSCOM/G.

Assoziationen sind in UNSCOM/T schließlich durch den allgemeinen Satzbauplan »Ein(e) *Informationsobjekt*$_1$ steht in einer *Name* Beziehung zu einem / einer *Informationsobjekt*$_2$ (und einem / einer *Informationsobjekt*$_3$)*« darzustellen, wobei der Beziehungsname zur Charakterisierung der Beziehungswirkung einzusetzen ist. Wie die Satzbaupläne zur Beschreibung von Teil-Ganzheits-Beziehungen und Konnexionen ist auch dieser Satzbauplan zur Angabe der mengenmäßigen Beteiligungen und der jeweils von den Informationsobjekten eingenommenen Rollen zu erweitern. Die vervollständigte Version findet sich in Abb. 4.23. In UNSCOM/G sind zur Darstellung von Assoziationen jeweils UML Assoziationen zu verwenden, die auch der Konnexion zugrunde liegen und deshalb in Abb. 4.22 (unter Weglassung der Assoziationsklasse) bereits dargestellt wurden. Im Gegensatz zur Konnexion ist dabei grundsätzlich der Name der Assoziationsbeziehung in der Mitte der Verbindung zu nennen [OMG 2003b:83]. Ferner ist nach dem UML Standard wiederum zwischen der Darstellung einer zweistelligen und einer mehrstelligen Assoziation zu unterscheiden [OMG 2003b:82].

Die zur Repräsentation von den UNSCOM/T und UNSCOM/G Formaten verwendeten Elemente sind in Abb. 4.23 im Detail aufgeführt und einander jeweils gegenübergestellt. Die in der Abbildung gezeigten Satzbaupläne sind dabei gegenüber der zuvor durchgeführten einführenden Betrachtung jeweils verallgemeinert worden, so dass sich sowohl die mengenmäßige Beteiligung der Ausprägungen als auch die jeweils eingenommenen Rollen darstellen lassen.

Abb. 4.23: Formatelemente zur Darstellung der Informationsobjekte in UNSCOM/T und UNSCOM/G.

4.4.2.2 Funktionen

Die im Rahmen einer schrittweisen Verfeinerung zwischen den Funktionen festzulegenden Spezialisierungs-Generalisierungs- bzw. Teil-Ganzheits-Beziehungen lassen sich in UNSCOM/T und UNSCOM/G analog zu den entsprechenden Beziehungen zwischen Informationsobjekten darstellen. An die Stelle der unbestimmten Artikel tritt dabei jedoch gemäß der eingangs genannten Regel zur Bezeichnung von Funktionen das primäre Handlungsobjekt, wodurch sich in UNSCOM/T folgende Satzbaupläne ergeben: »*Informationsobjekt*$_1$ *Funktion*$_1$ (oder *Informationsobjekt*$_2$ *Funktion*$_2$)* ist *Informationsobjekt*$_3$ *Funktion*$_3$« (Inklusion) und »*Informationsobjekt*$_1$ *Funktion*$_1$ besteht aus *Informationsobjekt*$_2$ *Funktion*$_2$ (und *Informationsobjekt*$_3$ *Funktion*$_3$)*« (Aggregation). In normierter Weise lassen sich so bspw. diese Aussagen als Ergebnis einer schrittweisen funktionalen Verfeinerung darstellen: »Lagerartikel auslagern besteht aus Kommissionierplan erstellen und Lagerartikel kommissionieren und Lagerartikel erfassen und Warenausgang buchen«, »Lagerartikel anlegen ist Artikel anlegen«.

In UNSCOM/G wird die funktionale Verfeinerung durch ein Klassendiagramm dargestellt, dessen Elemente mit dem Stereotyp «function» auszuzeichnen sind. Jede Funktionsklasse ist mit einem Namen zu versehen, der sich aus dem primären Handlungsobjekt und der Bezeichnung der Handlung zusammensetzt. Anstelle eines Leerzeichens wie bei UNSCOM/T wird hier jedoch aus notationstechnischen Gründen das Zeichen ‚_' für die Zusammensetzung des Namens verwendet. Zur Darstellung der beiden oben genannten Beziehungen ist – wie bei den entsprechenden Beziehungen zwischen Informationsobjekten – die UML Spezialisierung bzw. Aggregation zu verwenden (vgl. Abb. 4.24).

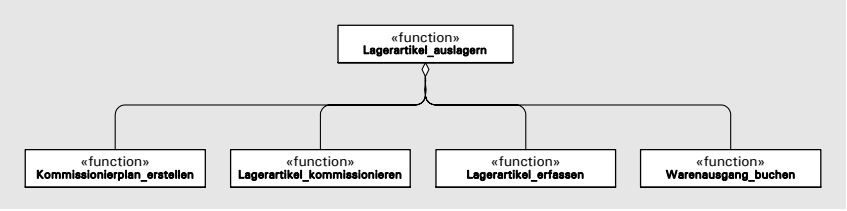


Abb. 4.24: Spezifikation einer funktionalen Verfeinerung in UNSCOM/G.

Funktionsmerkmale werden in UNSCOM/T gemäß einem Satzbauplan dargestellt, der auf dem Strukturwort π (»tut«) basiert und die detaillierte Festlegung des Handlungssubjekts, der Handlungsobjekte sowie der Handlungssituation erlaubt [SCHIENMANN 1997:242f.]. Das Handlungssubjekt steht dabei an der Subjektstelle des Satzes. Ihm folgen das primäre Objekt der Handlung, das als Objektfall an der ersten Objektstelle genannt wird, sowie der Bezeichner der Handlung, der an der Prädikatstelle genannt wird: »Ein(e) *Informationsobjekt*$_1$ tut einen / eine / ein *Informationsobjekt*$_2$ *Handlung*…«. Die sekundären Objekte der

Handlung stehen dagegen an der zweiten Objektstelle. Ihr jeweiliger Fall wird durch ein normsprachliches Kasusmorphem angezeigt [LORENZEN 1987:46]: die Mittelfälle stehen hinter einem »mit« und die Werkfälle hinter einem »zu«. Die Mittel- und Werkfälle sind dabei jeweils konjunktiv verknüpft. Unter Verwendung dieses Satzbauplans lassen sich zunächst folgende Sachverhalte normsprachlich darstellen: »Ein Lagerverwalter tut einen Lagerartikel kommissionieren mit einem Kommissionierplan«, »Ein Lagerverwalter tut einen Lagerartikel erfassen mit einem Kommissionierplan zu einem Warenausgang«, »Ein Kunde tut einen Kundenauftrag erteilen mit einem Internetkatalog« etc.

Zu beachten ist dabei zum einen, dass jede Funktion gemäß der UNSCOM Grammatik durch *ein* Handlungssubjekt und *einen* Objektfall zu charakterisieren ist, der das primäre Handlungsobjekt darstellt. Dagegen können im Allgemeinen mehrere Mittel- und Werkfälle auftreten. Das Auftreten mehrerer Werkfälle stellt bei der Darstellung von Funktionen allerdings die Ausnahme dar. Zum anderen ist der dargestellte Satzbauplan so zu erweitern, dass eine von den Vorgaben des Metaschemas abweichende Beteiligung der einzelnen Handlungsobjekte beschrieben werden kann. Der sich durch eine entsprechende Erweiterung ergebende Satzbauplan ist in Abb. 4.27 dargestellt. Mit ihm lassen sich Sachverhalte wie »Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel kommissionieren mit 1 bis beliebig vielen Kommissionierplan«, »Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel erfassen mit einem Kommissionierplan zu einem Warenausgang« darstellen.

Die Darstellung der Handlungssituation erfolgt schließlich durch mehrere prädikatenlogische Aussagen, die der Funktionsbeschreibung als Vor- oder Nachbedingungen angehängt werden [LORENZEN 1987:70-72]. Vorbedingungen stehen dabei hinter einem normsprachlichen »falls« und Nachbedingungen hinter einem »sodass«: »Ein Kunde tut einen Kundenauftrag erteilen mit einem Internetkatalog, falls für alle Verkaufsartikel des Kundenauftrag gilt, dass die Verfügbarkeit ‚lieferbar' ist«. Die einzelnen Vor- und Nachbedingungen sind dabei jeweils konjunktiv miteinander verknüpft. Die Form der prädikatenlogischen Aussagen ist durch die Grammatik der Prädikatenlogik erster Stufe festgelegt.

Bei der Darstellung in UNSCOM/G sind die Funktionsmerkmale in das Funktionsklassendiagramm zu integrieren. Die Handlungsobjekte sind hierzu der Funktionsklasse als Attribute hinzuzufügen, wobei mit dem Zusatz [min..max] bei Bedarf eine spezielle Beteiligung festgelegt wird. Bei der Zuordnung der Handlungsobjekte wird standardmäßig durch verschiedene Regionen zwischen dem primären Objektfall sowie den Mittel- und Werkfällen unterschieden (vgl. Abb. 4.25). Alternativ können jedoch auch alle Handlungsobjekte in einer Region dargestellt werden. In diesem Fall ist die Rolle der Handlungsobjekte allerdings explizit zu unterscheiden, weshalb das primäre Handlungsobjekt mit dem Stereo-

typ «inout», die Mittelfälle mit dem Stereotyp «in» und die Werkfälle mit dem Stereotyp «out» zu kennzeichnen sind.

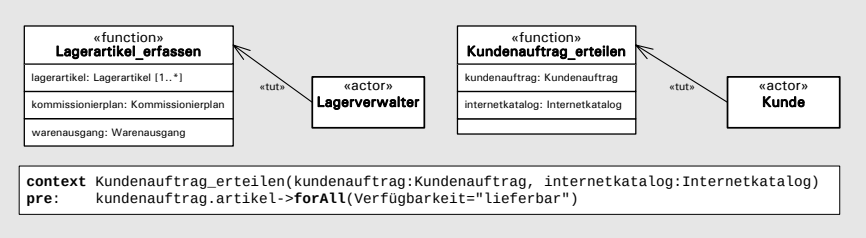


Abb. 4.25: Spezifikation der Funktionsmerkmale in UNSCOM/G.

Das Handlungssubjekt ist als *Akteur* [OMG 2003b:512f.] zu modellieren, der auf die von ihm jeweils ausgeführten Handlungen verweist. Hierdurch wird zugleich die *rollenspezifische Zusammenfassung* von Funktionen zu Funktionsgruppen unterstützt. Die Vor- und Nachbedingungen sind schließlich mit der *Object Constraint Language* (OCL [OMG 2003a]) als Einschränkungen zu formulieren und in das Modell aufzunehmen. Dabei kann bei Bedarf einer der auftretenden Werkfälle als Rückgabetyp dargestellt werden (vgl. Abb. 4.27). Standardmäßig werden jedoch alle Handlungsobjekte als Funktionsparameter in Klammern dargestellt, wobei deren jeweilige Rolle dem Funktionsklassendiagramm zu entnehmen ist.

4.4.2.3 Prozesse

Die zwischen den Elementen eines Prozesses existierenden Reihenfolgebeziehungen lassen sich in UNSCOM/T mit dem normsprachlichen Strukturwort → (»folgt«) und dem Satzbauplan »Nach *Prozesselement*$_1$ folgt *Prozesselement*$_2$« darstellen [ORTNER 1998:334]. Als unterliegende Ablaufstruktur wird dabei zunächst ein sequenzieller Ablauf von Aktivitäten angenommen: »Nach Kommissionierplan erstellen folgt Lagerartikel kommissionieren«. Nicht-sequenzielle Abläufe sind demgegenüber durch logische Operatoren (sog. *Junktoren*) explizit festzulegen, wobei jeweils der Beginn sowie das Ende eines nicht-sequenziellen Ablaufs durch einen Junktor zu definieren ist [BECKER und SCHÜTTE 2004:111]. Als mögliche nicht-sequenzielle Teilabläufe können in einem Prozess Parallelabläufe, inklusive und exklusive Verzweigungen auftreten. Zur Darstellung dieser Abläufe sind jeweils spezielle Varianten des oben eingeführten allgemeinen Satzbauplans zu verwenden, die in Abb. 4.27 (in Anlehnung an [ORTNER 1998:334]) zusammengestellt sind.

Die Junktoren stellen selbst wiederum Prozesselemente dar, für die in UNSCOM/T standardisierte Bezeichnungen nach dem Muster »Ablauftyp starten« und »Ablauftyp beenden« vordefiniert sind (vgl. Abb. 4.27). Darüber hinaus können als Prozesselemente vor allem

Verweise auf andere Prozesse sowie elementare Aktivitäten auftreten, die jeweils aus einer Funktion und einem Startereignis für deren Ausführung bestehen. Eine Sonderstellung nehmen dagegen die Start- und die Schlussaktivität sowie die (bei der Spezifikation von alternativen Verzweigungen) ggf. auftretende Leeraktivität ein.

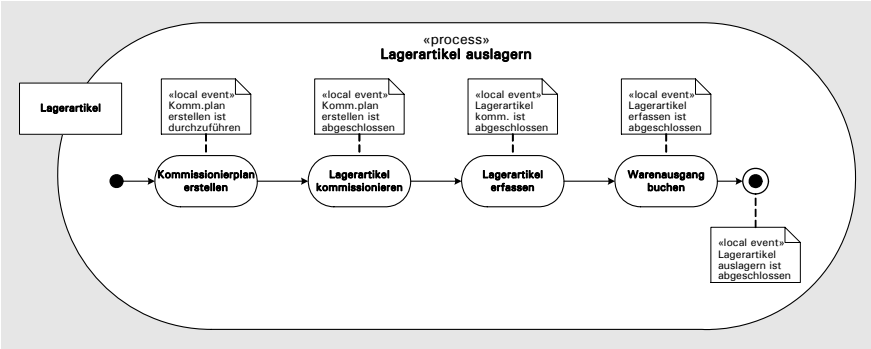


Abb. 4.26: Spezifikation der Prozesse in UNSCOM/G.

In UNSCOM/G werden die Reihenfolgebeziehungen eines Prozesses jeweils durch ein UML Aktivitätsdiagramm [OMG 2003b:265] dargestellt. Verweise auf andere Prozesse und elementare Aktivitäten sind dementsprechend durch UML Aktivitäten graphisch zu repräsentieren, wobei jeder elementaren Aktivität zusätzlich ein Startereignis zuzuordnen ist. Anders als bei den zuvor eingeführten Funktionsklassendiagrammen werden die Namen von Aktivitäten unter Verwendung eines Leerzeichens zusammengesetzt (vgl. Abb. 4.26). Die zeitliche Nachfolge eines Prozesselements auf ein anderes wird im Aktivitätsdiagramm durch einen Pfeil angedeutet. Darüber hinaus sind sowohl für die vordefinierte Start- und Schlussaktivität [OMG 2003b:331f.] als auch für den Beginn und das Ende nicht-sequenzieller Abläufe jeweils spezielle Symbole festgelegt [OMG 2003b:316-318], die in Abb. 4.27 zusammengefasst sind.

In Abb. 4.27 werden die zur Repräsentation von den UNSCOM/T und UNSCOM/G Formaten verwendeten Elemente im Detail gezeigt und dabei die einander jeweils entsprechenden Elemente gegenübergestellt. Anzumerken bleibt abschließend, dass es mit den vorgestellten Formatelementen grundsätzlich durchaus möglich ist, auch ungültige Aussagen zu formulieren, die nicht den Festlegungen der UNSCOM Grammatik entsprechen. So lassen sich bspw. Prozesse definieren, bei denen die Teilabläufe nicht in der vom Metaschema geforderten geschachtelten Weise auftreten, sondern sich gegenseitig überlappen. Deshalb ist bei der Nutzung der dargestellten Formate während der Spezifikation in jedem Fall die Einhaltung der im UNSCOM Metaschema festgelegten Grammatik sicherzustellen. Dies lässt sich ggf. mit einem Compiler automatisiert überprüfen, der (idealerweise) mit der jeweiligen Entwicklungsumgebung bereitzustellen ist.

Abb. 4.27: Formatelemente zur Darstellung der Funktionen und Prozesse in UNSCOM/T und UNSCOM/G.

4.4.3 Geltung

Die in den Blue Pages zu beschreibende Funktionalität wird mit dem Fachentwurf (bzw. dem Detailed Requirements Engineering) festgelegt, in dessen Rahmen zunächst das für die Entwicklung relevante fachliche Handeln beschrieben wird. Dabei wird der jeweils entwicklungsrelevante Ausschnitt aus dem Begriffssystem des Anwendungsbereichs rekonstruiert und als *konzeptionelles Modell* expliziert [D'SOUZA und WILLS 1999:528; CHEESMAN und DANIELS 2001:68-70]. Die mit dem Fachentwurf verfolgte Klärung des im Anwendungsbereich herrschenden Begriffsverständnisses bildet die Grundlage für den sich anschließenden System- und Komponentenentwurf [D'SOUZA und WILLS 1999:528]. Bei der Erstellung des konzeptionellen Modells kann ggf. auf *Referenzmodelle* zurückgegriffen werden, die als sog. *Domänenstandards* allgemeingültige Zusammenhänge zwischen den Informationsobjekten, Funktionen und Prozessen eines Anwendungsbereichs beschreiben.

Um von dem im Rahmen des Fachentwurfs festgelegten konzeptionellen Modell zu dem für die Spezifikation der Komponentenfunktionalität relevanten Modellausschnitt zu gelangen, ist während des Systementwurfs eine Projektion durchzuführen. Dabei ist das konzeptionelle Modell analog zur Strukturierung des Anwendungssystems in Komponenten zunächst in Teilmodelle zu gliedern, die das für die jeweiligen Komponenten relevante Begriffssystem beschreiben. In einem zweiten Arbeitsschritt ist dann zwischen den Teilen eines Begriffssystems zu trennen, die die interne Funktionalität (Realisierung von Diensten) und die nach außen sichtbare Funktionalität (Eigenschaften von Diensten) beschreiben. Lediglich die Beschreibung der nach außen sichtbaren Funktionalität wird dabei zu einem Bestandteil der Komponentenspezifikation, während die Spezifikation der internen Funktionalität vor allem eine Grundlage für den Komponentenentwurf bildet.

In Abhängigkeit vom *Detaillierungsgrad*, der bei der Erstellung des konzeptionellen Modells während des Fachentwurfs realisiert wurde, kann die Beschreibung der Komponentenfunktionalität nach der Projektion auf verschiedene Weise vorliegen. Im einfachsten Fall (*Detailstufe 1: Terminologie*) besteht sie lediglich aus einem Lexikon, in dem die einzelnen Fachbegriffe durch ihren normierten Bezeichner sowie eine umgangssprachliche Kurz- und Langdefinition charakterisiert werden [D'SOUZA und WILLS 1999:528]. Bei der üblicherweise anzustrebenden formal-präzisen Beschreibung (*Detailstufe 2: Ontologie*) sind hingegen auch die Zusammenhänge zwischen den Begriffen als normierte Aussagen festgehalten [D'SOUZA und WILLS 1999:528]. Eine vollständige Fassung (*Detailstufe 3: Ontologie mit Verhalten*) beinhaltet schließlich außerdem eine Beschreibung der Handlungssituationen, in denen die einzelnen Funktionen erfolgreich ausgeführt werden können.

Bis zum Abschluss der Implementierung legt die in den Blue Pages spezifizierte Komponentenfunktionalität einen Soll-Zustand fest, der bei der Realisierung einzuhalten ist. Nach Abschluss der Implementierung und der Überprüfung der materialen Korrektheit (d.h. der Übereinstimmung mit dem festgelegten Begriffssystem) sind die Blue Pages zu verwenden, um die jeweils realisierte Komponentenfunktionalität zu beschreiben. Sie dokumentieren dann den nach der Komponentenentwicklung erreichten Ist-Zustand. Dieser kann später bei der Auswahl von Komponenten im Rahmen von Entwicklungsprojekten ausgewertet werden, um die fachliche Eignung einer Komponente zu beurteilen [D'SOUZA und WILLS 1999:583]. Durch die explizite Dokumentation des fachlichen Begriffssystems von Komponenten können sich im Laufe der Zeit neue Domänenstandards herausbilden – etwa wenn ein Begriffssystem auch von anderen Komponenten als fachliche Grundlage verwendet wird [D'SOUZA und WILLS 1999:528]. Damit kann die Dokumentation der Komponentenfunktionalität zur Entstehung fachlicher Referenzmodelle beitragen.

In diesem Zusammenhang ist zu erwähnen, dass das UNSCOM Metaschema bewusst darauf ausgelegt wurde, auch Beschreibungen aufzunehmen, die in den bei der Referenzmodellierung häufiger verwendeten Spezifikationssprachen [BECKER und SCHÜTTE 2004:87, 103, 110f.] verfasst wurden. Deshalb lässt sich bei Bedarf ein Repräsentationsformat (z.B. als UNSCOM/R) definieren, das Entity-Relationship-Diagramme, Funktionsbäume und Ereignisgesteuerte Prozessketten zur Darstellung von Begriffssystemen verwendet. Zur vollständigen Beschreibung einer Komponente ist nach dem in Abschnitt 3.1.6 vorgestellten Komponentenmodell außerdem zwischen angebotenen und nachgefragten Diensten zu unterscheiden, und jeweils eine rollenspezifische Gruppierung von Diensten vorzunehmen. Da die Dienste der Komponente in den Blue Pages durch Funktionen (Handlungen) beschrieben werden, lässt sich Letzteres durch deren Zusammenfassung nach Handlungssubjekten erreichen. Der Charakter einer Funktion kann im Lexikon dagegen erst durch einen zusätzlichen Eintrag als ‚angeboten' oder ‚nachgefragt' festgelegt werden. Mit dem UNSCOM Spezifikationsrahmen lässt sich diese Einordnung allerdings auf eine andere Weise treffen, die im nächsten Kapitel dargestellt wird. Auf die Aufnahme eines Lexikoneintrags zur Klassifikation von Funktionen in das Metaschema wurde deshalb verzichtet.

4.5 Logische Architektur

Um eine Komponente und die von ihr jeweils angebotene Funktionalität zu nutzen, ist diese während der Anwendungsentwicklung mit anderen Komponenten zu einem Anwendungssystem zu koppeln. Dabei werden vor allem Informationen über die nach außen sichtbare *systemtechnische Beschaffenheit* der Komponente benötigt, die in den bislang

vorgestellten thematischen Bereichen des UNSCOM Spezifikationsrahmens noch nicht enthalten sind. Deshalb sind neben den bereits erwähnten Eigenschaften auch architekturspezifische Merkmale der Komponentenaußensicht zu spezifizieren. Sie beinhalten Angaben zu den Signaturen der Komponente und ihrer Schnittstellen, zu den Vor- und Nachbedingungen der Methoden sowie zu den Reihenfolgebedingungen, die zwischen den Methoden existieren und bei der Interaktion einzuhalten sind. Die Beschreibung der architekturspezifischen Merkmale deckt die in Abschnitt 3.4.2 zur Festlegung der logischen Sicht identifizierten Ebenen eines Komponentenvertrags ab. Sie werden vom UNSCOM Spezifikationsrahmen als sog. *Green Pages* wiederum zu einem eigenen thematischen Bereich zusammengefasst. Die Spezifikation orientiert sich dabei an Vorgaben, die den Inhalt, die Repräsentation und die Geltung der Merkmale während des Entwicklungsprozesses festlegen.

4.5.1 Inhalt

Eine Software-Architektur beschreibt schwerpunktmäßig die (software-technische) Organisation eines Anwendungssystems im Großen, umfasst häufig aber auch eine Betrachtung im Kleinen. Im Großen steht die *Strukturierung* eines Anwendungssystems in Komponenten und deren Zusammenwirken im Mittelpunkt der Betrachtung [BASS, et al. 1998:27; BOOCH, et al. 1999:31; IEEE 2000:9]. Eine entsprechende Strukturierung in Komponenten wurde bspw. im Rahmen der Fallstudie für einen Teil des Verkaufssystems durchgeführt (vgl. Abschnitt 3.1.7). Im Kleinen wird dagegen die Organisation der einzelnen Komponenten aus Implementierungseinheiten (wie z.B. Modulen, Klassen) festgelegt [BOOCH, et al. 1999:31]. Software-Architekturen stellen komplexe Entwicklungsergebnisse dar, die unterschiedliche Aspekte umfassen und im Allgemeinen aus verschiedenen *Sichten* bestehen. Zur Beschreibung von Software-Architekturen werden in der Literatur zwar meist unterschiedliche Sichten mit teilweise voneinander abweichendem Inhalt empfohlen [KRUCHTEN 1995; HOFMEISTER, et al. 2000; CLEMENTS, et al. 2003:15-17]. Häufig werden dabei jedoch Architektursichten genannt, die den in Abb. 4.28 links dargestellten ähneln. Diesen vier idealtypischen Sichten kommt somit eine grundlegende Bedeutung zu.

Die *Struktursicht* (auch als Logical bzw. Conceptual View bezeichnet) beschreibt den Aufbau eines Anwendungssystems aus Komponenten. Dies schließt die Beschreibung der Komponentenschnittstellen und der zwischen den Komponenten jeweils bestehenden Abhängigkeiten ein. Die *Kontrollsicht* (auch als Process bzw. Execution View bezeichnet) stellt den Kontrollfluss dar, der zur Laufzeit durchlaufen wird. Dabei wird neben der ggf. vorhandenen Nebenläufigkeit vor allem auf die Mechanismen zur Fehlertoleranz und zur Sicherstellung der Systemintegrität eingegangen. Die *Verteilungssicht* (auch als Physical

bzw. Deployment View bezeichnet) beschreibt die physische Architektur des Anwendungssystems. Hierzu gehört vor allem die Festlegung der (hardware-) technischen Infrastruktur, die Verteilung der Komponenten auf verschiedene Rechner und die Entscheidung für eine Implementierungsplattform. Die *Abbildungssicht* (auch als Development bzw. Module View bezeichnet) stellt schließlich den inneren Aufbau der einzelnen Komponenten dar und beschreibt deren jeweilige Implementierungseinheiten.

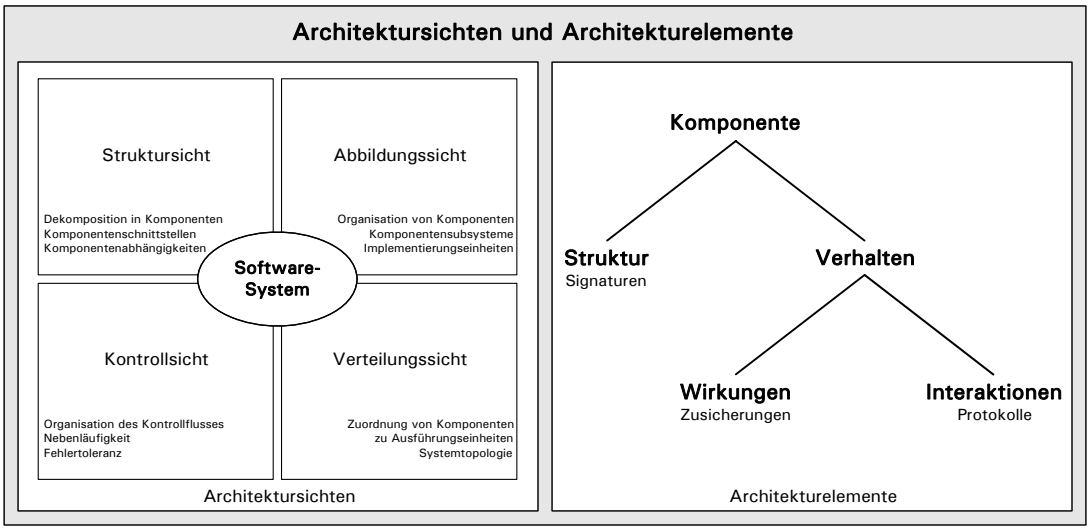


Abb. 4.28: Sichten zur Beschreibung von Software-Architekturen und Einteilung von Architekturelementen zur Spezifikation der Komponentenaußensicht (in Anlehnung an [CLEMENTS, et al. 2003:15-17]).

In Bezug auf die Spezifikation der architekturspezifischen Merkmale von Komponenten lassen sich vor allem der Struktur- und der Kontrollsicht Informationen entnehmen, die Aufschluss über die nach außen sichtbare systemtechnische Beschaffenheit der *einzelnen Komponenten* geben. Die Eigenschaften des Systems sind für die Spezifikation von Komponenten dagegen ebenso wenig von Interesse wie die Informationen der Verteilungssicht, die das Zusammenwirken der Komponenten mit der systemspezifischen (hardware-) technischen Infrastruktur darstellt, oder die Informationen der Abbildungssicht. Die Abbildungssicht legt vielmehr die Realisierung der Komponenten fest und *ergänzt* somit die entsprechenden Komponentenspezifikationen um eine Innensicht (vgl. Abschnitt 2.3.1).

Die für die Spezifikation von Komponenten relevanten architekturspezifischen Informationen lassen sich prinzipiell in *strukturspezifische* (statische) und *verhaltensspezifische* (operationale bzw. dynamische) Merkmale unterteilen [CLEMENTS, et al. 2003:4]. Die nach außen sichtbare systemtechnische Struktur einer Komponente wird durch die *Signaturen* ausgedrückt, die als Komponentensignatur die Angebots- und Nachfrageschnittstellen der Komponente (den Komponententyp), als Schnittstellensignaturen den Aufbau der Schnittstellen und als Methodensignaturen schließlich die Beschaffenheit der Methoden definie-

ren. Zu den verhaltensspezifischen Architekturmerkmalen einer Komponente gehören demgegenüber die für die einzelnen Methoden festgelegten Vor- und Nachbedingungen, die die systemtechnischen Wirkungen der Komponente als vertragliche *Zusicherungen* beschreiben (vgl. Abschnitt 3.3.1). Darüber hinaus sind die Reihenfolgebedingungen anzugeben, die zwischen den einzelnen Methoden existieren. Aus ihnen lässt sich das Interaktionsprotokoll ableiten, das den Verlauf von Interaktionen zur Laufzeit vorgibt.

4.5.1.1 Signaturen

Die für die Beschreibung der Komponentenaußensicht relevanten Signaturen sind in der Regel zu *Modulen* zusammengefasst, die abhängig von der eingesetzten Notation auch als Pakete oder Gruppen bezeichnet werden. Jedes Modul enthält thematisch zusammengehörige Signaturen, die einen Teil der systemtechnischen Komponentenaußensicht festlegen [OMG 2004:3-20]. Im UNSCOM Metaschema werden die bei der Architekturbeschreibung erstellten Module durch das Schemaelement Definitionsmodul dargestellt (vgl. Abb. 4.29). Jedes Modul besitzt einen eindeutigen Namen und verweist auf eine Menge von Untermodulen und Signaturen. Letztere definieren *Typen*, *Konstanten*, *Ausnahmen, Ereignisse*, *Schnittstellen* oder *Komponenten (-typen)* [GRIFFEL 1998:60f.; OMG 2004:3-12]. Bei der Definition von Signaturen sind ggf. andere Module zu importieren, um auf die dort enthaltenen Signaturen Bezug zu nehmen. Eine der Komponententypdefinitionen legt darüber hinaus den Typ der zu spezifizierenden Komponente fest (vgl. Abschnitt 3.2). Diese spezielle Definition ist im UNSCOM Metaschema in Beziehung mit der zu spezifizierenden Komponente zu setzen. Ferner sind der zu spezifizierenden Komponente sämtliche Module zuzuordnen, die mit ihren Signaturen zur Beschreibung der äußeren Struktur beitragen.

Durch die in den Modulen enthaltenen Typdefinitionen werden *anwendungsspezifische Typen* festgelegt, die als Bestandteile in die Definition der Komponentenschnittstellen bzw. ihrer Methoden eingehen. Als Ausgangspunkt für die Definition von anwendungsspezifischen Typen dienen elementare Datentypen, die sowohl von der UML als auch den gängigen Programmiersprachen unterstützt werden und vom Metaschema vorgegeben sind (vgl. Abb. 4.29). Zu diesen gehören Ganzzahlen (Integer), Fließkommazahlen (Real), Zeichenketten (String) und Wahrheitswerte (Boolean). Weitere, anwendungsspezifische Typen können zunächst durch *einfache Typdefinitionen* vereinbart werden. Mit ihnen lassen sich die Werte eines elementaren Datentyps, eines bereits definierten anwendungsspezifischen Typs oder eine Gruppe von Werten desselben Typs unter einem neuen Namen zusammenfassen [OMG 2004:3-36]. Durch entsprechende Definitionen lassen sich bspw. die Typen Artikelnummer, Artikelname als spezielle Zeichenketten, Lagerbestand als Ganzzahl und Artikelnummern als Artikelnummer-Gruppe definieren: Artikelnummer, Artikelname $=_{DF}$ String; Lagerbestand $=_{DF}$ Integer; Artikelnummern $=_{DF}$ (Artikelnummer)*.

Die Vereinbarung anwendungsspezifischer Typen kann darüber hinaus mittels *komplexer Typdefinitionen* erfolgen. Diese setzen neue Typen aus mehreren elementaren Datentypen oder bereits definierten anwendungsspezifischen Typen zusammen. Durch komplexe Typdefinitionen lassen sich *strukturierte Typen* vereinbaren, die aus mehreren Teilen bestehen [OMG 2004:3-40]. Ein solcher strukturierter Typ entsteht bspw. bei der Definition des anwendungsspezifischen Datums Artikel, das die Artikelnummer, den Artikelnamen und die Steuerklasse als charakteristische Eigenschaften von Artikeln zusammenfasst: Artikel $=_{DF}$ {(nummer: Artikelnummer), (name: Artikelname), (stklasse: Steuerklasse)}. Komplexe Typdefinitionen werden außerdem zur Vereinbarung von *Aufzählungstypen* verwendet, die nur eine Menge jeweils vorab festgelegter Werte annehmen dürfen [OMG 2004:3-43f.]. Bei der Definition eines Aufzählungstyps sind diese Werte folglich als charakteristische Eigenschaften aufzuführen: Steuerklasse $=_{DF}$ {'voll', 'ermaessigt'}.

Im UNSCOM Metaschema werden einfache Typdefinitionen durch das Schemaelement EinfacherTyp und komplexe Typdefinitionen durch das Element KomplexerTyp dargestellt. Dabei ist zunächst ein *Name* für den neu vereinbarten Typ anzugeben. Bei einfachen Typdefinitionen ist zudem ein Verweis auf denjenigen *Typen* herzustellen, der als Grundlage für die Ableitung des neuen Typs dient (vgl. Abb. 4.29). Daneben ist ggf. festzuhalten, dass der neue Datentyp mehrere Werte desselben Typs zu einer *Gruppe* zusammenfasst. Solche Wertegruppen können auf zwei miteinander kombinierbare Weisen gebildet werden[12]: *Felder*, die auch als *Arrays* bezeichnet werden, verbinden eine feste Anzahl von Werten zu einer Gruppe [OMG 2004:3-46]. Durch *Sequenzen* werden hingegen Wertegruppen gebildet, bei denen die Anzahl der Elemente zur Laufzeit (wie bspw. beim Typ Artikelnummern) bis zum Erreichen einer ggf. festgelegten Obergrenze beliebig schwanken darf [OMG 2004:3-44]. Bei komplexen Typdefinitionen ist im Metaschema zunächst festzulegen, ob ein *strukturierter Typ* oder ein *Aufzählungstyp* vereinbart wird (vgl. Abb. 4.29). Daneben ist für jede charakteristische Eigenschaft (Typelement) zumindest ein *Name* zu spezifizieren. Bei der Vereinbarung von strukturierten Typen ist außerdem der *Typ* der Eigenschaften anzugeben und ggf. festzulegen, ob von ihnen jeweils eine *Gruppe* von Werten zusammengefasst wird.

Unter Verwendung von elementaren Datentypen und anwendungsspezifischen Typen, die entweder durch eine einfache Definition aus einem elementaren Datentyp abgeleitet oder als Aufzählungstypen vereinbart wurden, lassen sich *Konstanten* definieren [OMG 2004:3-

[12] Eine kombinierte Verwendung von Sequenzen und Feldern im Rahmen einer Typdefinition bedeutet, dass zunächst eine Gruppierung zu Sequenzen und im Anschluss eine Gruppierung zu Feldern erfolgt.

33]. Konstanten besitzen einen unveränderlichen Wert, der durch eine Berechnungsvorschrift festzulegen ist. Im einfachsten Fall besteht diese Berechnungsvorschrift direkt aus dem jeweiligen Wert. Dies gilt bspw. für den Umrechnungskurs zwischen Euro und D-Mark, der von der Komponente Rechnungsabwicklung verwendet wird: FixEURDEM $=_{DF}$ (Real, 1.95583). Im UNSCOM Metaschema werden Konstanten durch das gleichnamige Schemaelement Konstante dargestellt (vgl. Abb. 4.29). Zusätzlich zum *Namen* der Konstante sind dabei ihr jeweiliger *Typ* sowie die *Berechnungsvorschrift* anzugeben, aus der sich der Wert der Konstante ableiten lässt.

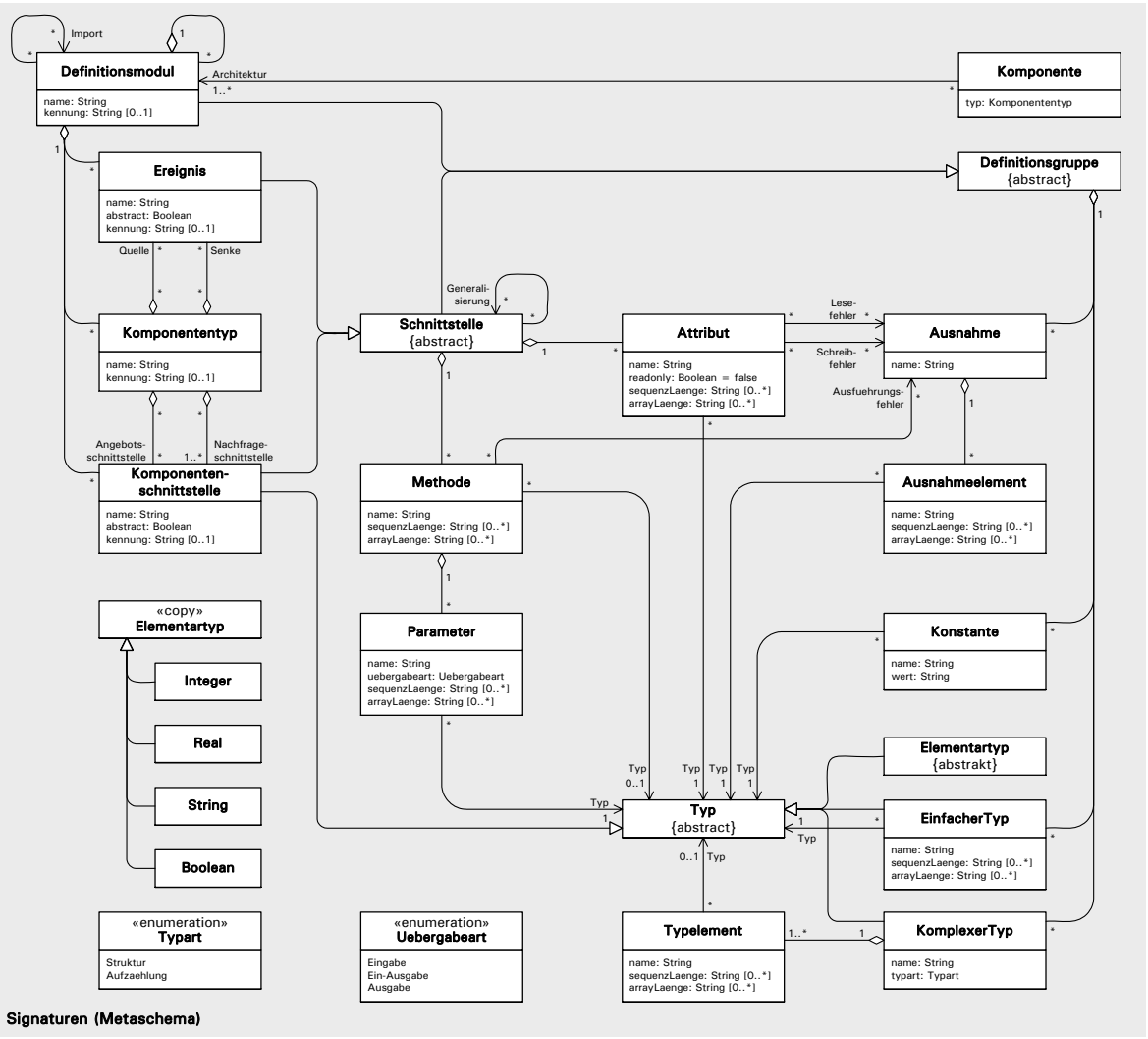


Abb. 4.29: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Signaturen.

Durch die Definition von *Ausnahmen* lassen sich Rückmeldungen festlegen, die beim Eintreten von unerwarteten bzw. außerordentlichen Zuständen während der Informationsverarbeitung von der zu spezifizierenden Komponente ausgegeben werden [MEYER 1992:411; OMG 2004:3-49]. So wird von der Komponente Auftragsabwicklung bspw. die Ausnah-

me UngueltigerArtikel ausgegeben, falls ein nicht gelisteter Verkaufsartikel bestellt wird. Im UNSCOM Metaschema werden Ausnahmen durch das Schemaelement Ausnahme dargestellt (vgl. Abb. 4.29). Sie besitzen neben einem Namen ggf. eine Reihe von charakteristischen Eigenschaften, die Details über den jeweils aufgetretenen Fehlerzustand enthalten. Von ihrer Signatur her ähnelt die Definition einer Ausnahme dabei der Vereinbarung eines strukturierten Typs [OMG 2004:3-49]. So sind neben dem *Namen* der Ausnahme für jede charakteristische Eigenschaft (Ausnahmeelement) ein eindeutiger *Name*, ein *Typ* und Informationen über die ggf. zu einer *Gruppe* zusammengefassten Werte anzugeben.

Unter Verwendung von zuvor vereinbarten Typen, Konstanten und Ausnahmen lassen sich *Komponentenschnittstellen* spezifizieren, die als Angebots- oder Nachfrageschnittstellen in die Definition von Komponententypen eingehen. Beispiele für Schnittstellen sind ILagervwt, ILogistik und IBestandsvwt, die in Abschnitt 3.1.7 definiert und der Komponente Lagermanagement als Angebotsschnittstellen zugeordnet wurden. Durch die Definition einer Schnittstelle entsteht ein anwendungsspezifischer Typ, der im Rahmen von anderen Definitionen (z.B. zur Ableitung neuer anwendungsspezifischer Typen) verwendet werden kann [OMG 2004:3-21, 23-37]. Im Gegensatz zu Datentypen, deren Ausprägungen zur Laufzeit stets Werte enthalten, besitzen Ausprägungen von Schnittstellentypen Verweise auf Laufzeitobjekte, die die jeweilige Schnittstelle implementieren [OMG 2004:3-21]. Die definierten Schnittstellentypen unterscheiden sich also durch ihre *Referenzsemantik* von Datentypen, die über eine *Wertsemantik* verfügen. Komponentenschnittstellen bestehen hauptsächlich aus *Methoden*, die die software-technischen Realisierungen der von einer Komponente angebotenen bzw. nachgefragten Dienste darstellen [OMG 2004:3-22]. Zudem können Komponentenschnittstellen *Attribute* besitzen, mit denen sich Eigenschaften festlegen lassen, die für die Durchführung der Informationsverarbeitung von Bedeutung sind [OMG 2004:3-22]. Unter anderem kann die Funktionalität der an den Schnittstellen aufgeführten Methoden durch die Definition entsprechender Attribute parametrisiert und durch die Veränderung der Attributwerte zur Laufzeit beeinflusst werden. So lässt sich bspw. die Rechnungsschreibung, die durch die Komponente Rechnungsabwicklung durchgeführt wird, durch Veränderung der Attribute UmsatzsteuersatzVoll und UmsatzsteuersatzErmaessigt dynamisch beeinflussen. Bei der Vereinbarung einer Komponentenschnittstelle können ferner weitere Typen, Konstanten bzw. Ausnahmen definiert werden [OMG 2004:3-22], die als *lokale Definitionen* jedoch ausschließlich zur Vereinbarung von Bestandteilen *dieser* Schnittstelle verwendet werden dürfen.

Vereinbarte Komponentenschnittstellen werden im UNSCOM Metaschema durch das Schemaelement Komponentenschnittstelle dargestellt (vgl. Abb. 4.29). Bei der Definition ist der *Name* festzulegen und anzugeben, ob es sich um eine *abstrakte Schnittstelle* handelt

[OMG 2004:3-21]. Abstrakte Schnittstellen können nicht von einer Komponente realisiert werden, sondern dienen lediglich der Gruppierung zusammengehöriger Dienste und sind durch Vererbungsbeziehungen von spezielleren Schnittstellen zu konkretisieren. Dazu können bei der Vereinbarung jeder Schnittstelle Verweise auf andere Schnittstellen angegeben werden, die im Rahmen von *Vererbungsbeziehungen* spezialisiert werden [OMG 2004:3-21]. Darüber hinaus sind Verweise auf sämtliche Signaturvereinbarungen hinzuzufügen, die Bestandteile der Schnittstelle darstellen. Hierzu zählen lokale Definitionen von *Typen*, *Konstanten* und *Ausnahmen* ebenso, wie die *Methoden* und *Attribute*, die mit der jeweiligen Schnittstelle vereinbart werden [OMG 2004:3-22].

Die *Methoden* der einzelnen Schnittstellen werden im Metaschema durch das Schemaelement Methode dargestellt (vgl. Abb. 4.29). Für jede Methode ist ein *Name* festzulegen, der sich von den Namen der anderen Schnittstellenelemente unterscheidet. Daneben sind Verweise auf die bei der Ausführung ggf. erzeugten *Ausnahmen* und den *Typ* des ggf. erzeugten Rückgabewerts zu spezifizieren [OMG 2004:3-50]. Schließlich ist eine geordnete Liste von Parametern anzugeben, die beim Aufruf einer Methode zur Laufzeit zu übergeben sind. Die vereinbarten Parameter werden im Metaschema durch das Schemaelement Parameter dargestellt (vgl. Abb. 4.29). Bei der Definition eines Parameters ist der *Name*, der *Typ* sowie die Art der *Übergabe* festzulegen [OMG 2004:3-51f.]. Daneben sind Informationen über die ggf. als *Gruppe* zusammengefassten Werte anzugeben. Standardmäßig wird ein Parameter als *Eingabe-Parameter* beim Aufruf vom Dienstnachfrager an den Dienstanbieter übergeben. Daneben ist jedoch auch die Vereinbarung sog. *Ausgabe-Parameter*, die vom Dienstanbieter nach Ausführung der Methode an den Dienstnachfrager übergeben werden, und *Ein-Ausgabe-Parameter*, die sowohl beim Aufruf als auch nach der Ausführung übergeben werden, möglich [OMG 2004:3-51]. Die *Attribute* der Schnittstellen werden im UNSCOM Metaschema schließlich durch das Schemaelement Attribut dargestellt. Bei der Definition eines Attributs ist der *Name*, ein Verweis auf den *Typ* des Attributs sowie auf die beim (Lese- und Schreib-) Zugriff ggf. erzeugten *Ausnahmen* zu spezifizieren [OMG 2004:3-53f.]. Des Weiteren ist anzugeben, ob das Attribut ggf. mehrere Werte zu einer Gruppe zusammenfasst und ob es sich um ein veränderliches oder unveränderliches Attribut handelt. Letzteres kann nach der Initialisierung mit einem Wert zur Laufzeit nicht mehr geändert werden [OMG 2004:3-54].

Eine besondere Form der Schnittstellendefinition stellt die Vereinbarung von *Ereignissen* dar, die zur Laufzeit von Komponenten ausgesandt und von anderen Komponenten empfangen werden können. Verglichen mit einer Interaktion über Komponentenschnittstellen wird dabei jedoch die Richtung der Kommunikation *invertiert*. Der Dienstnachfrager tritt also nicht in Kontakt mit dem Dienstanbieter, sondern wird von diesem unaufgefordert

über das Eintreten eines bestimmten Zustands informiert [SZYPERSKI, et al. 2002:173]. So benachrichtigt bspw. die Komponente Lagermanagement die anderen Komponenten durch ein Ereignis MindestbestandUnterschritten, dass der Lagerbestand eines Artikels aufzufüllen ist. Durch das Versenden von Ereignissen findet eine lose Form der Kommunikation statt. Anders als bei einer Kommunikation über Schnittstellen ist nämlich weder der Dienstanbieter noch der Dienstnachfrager darauf angewiesen, dass sich die jeweils andere Partei an der Kommunikation beteiligt [SZYPERSKI, et al. 2002:348; OMG 2004:3-57]. Nach den Vorgaben des UNSCOM Metaschemas entspricht die Definition von Ereignissen der Definition von Schnittstellen (vgl. Abb. 4.29). Ereignisse, die im Metaschema durch das Schemaelement Ereignis dargestellt werden, stellen als sog. *ausgehende Schnittstellen* somit lediglich eine spezielle Schnittstellenart dar [SZYPERSKI, et al. 2002:348; OMG 2004:3-57f.].

Unter Rückgriff auf die vereinbarten Komponentenschnittstellen und Ereignisse lassen sich schließlich *Komponententypen* definieren. Ein Komponententyp legt vor allem die *Angebots- und Nachfrageschnittstellen* einer Komponente fest [OMG 2004:3-60f.]. Im Rahmen der Fallstudie werden bspw. die Komponententypen Auftragsabwicklung, Kundenmanagement, Lagermanagement, Artikelmanagement und Rechnungsabwicklung mitsamt ihrer jeweiligen Angebots- bzw. Nachfrageschnittstellen definiert (vgl. Abschnitt 3.1.7). Neben den Angebots- und Nachfrageschnittstellen werden durch den Komponententyp ggf. auch die Ereignisse festgelegt, die von einer Komponente (als *Quelle*) ausgesandt bzw. (als *Senke*) empfangen werden können [OMG 2004:3-62f.]. Im UNSCOM Metaschema werden die vereinbarten Komponententypen durch das gleichnamige Schemaelement Komponententyp dargestellt (vgl. Abb. 4.29). Zur Definition eines Komponententyps ist ein *Name*, eine Liste von *Angebots- und Nachfrageschnittstellen* sowie eine Liste von *ausgesandten und empfangbaren Ereignissen* anzugeben [OMG 2004:3-58].

Anders als Schnittstellen- und Datentypen werden die vereinbarten Komponententypen im UNSCOM Metaschema jedoch *nicht* als anwendungsspezifische Typen betrachtet, die im Rahmen anderer Definitionen verwendet werden dürfen. Referenzen auf Komponenteninstanzen ergeben sich zur Laufzeit somit ausschließlich aus den Schnittstellentypen, die bei der Definition der Angebots- und Nachfrageschnittstellen entstehen und im Rahmen weiterer Signaturvereinbarungen genutzt werden. Ferner schließen die Vorgaben des Metaschemas jede Form der (Implementierungs-) Vererbung zwischen Komponenten aus, da Komponenten grundsätzlich als *eigenständige Strukturierungseinheiten* von Anwendungssystemen angesehen werden (vgl. Abschnitt 2.1.1 bzw. 2.1.3). Damit unterscheiden sich die Vorgaben des UNSCOM Metaschemas insbesondere von denen des CORBA Component Model Standards, der sowohl Vererbungsbeziehungen zwischen Komponenten als auch die

Verwendung von Komponententypen im Rahmen weiterer Definitionen zulässt (vgl. [OMG 2002:1-6; OMG 2004:3-59]).

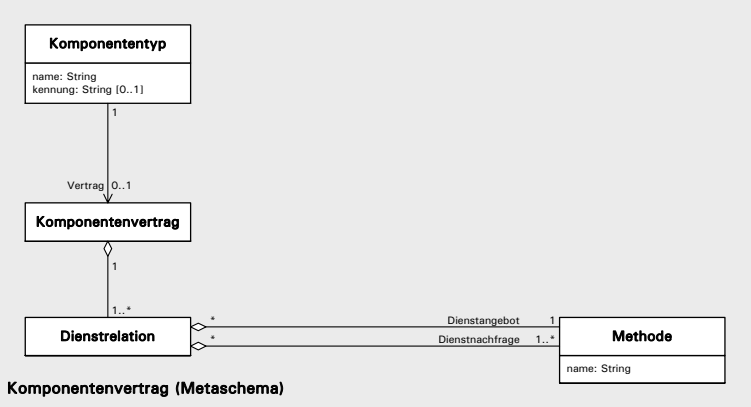


Abb. 4.30: Metaschema mit inhaltlichen Vorgaben zur Spezifikation parametrisierter Komponentenverträge.

Mit der Spezifikation der Angebots- und Nachfrageschnittstellen ist auch der *Komponentenvertrag* festzulegen (vgl. Abschnitt 3.3.2), der bei der Komposition mit anderen Komponenten zu erfüllen ist. Die Nachfrageschnittstellen beschreiben dabei diejenigen Methoden, die von den anderen Komponenten mit den spezifizierten Eigenschaften bereitzustellen sind. Unter der Voraussetzung, dass die nachgefragten Methoden zur Verfügung gestellt werden, bietet die Komponente im Gegenzug die durch die Angebotsschnittstellen festgelegten Methoden samt der dort spezifizierten Eigenschaften an. Der festgelegte Komponentenvertrag lässt sich in Form eines *parametrisierten Komponentenvertrags* spezifizieren, indem für jede von der Komponente angebotene Methode angegeben wird, welche der nachgefragten Methoden dafür im Gegenzug durch andere Komponenten bereitzustellen sind (vgl. Abschnitt 3.3.3). Hierdurch lässt sich dynamisch ermitteln, welche Nachfrageschnittstellen durch die Umgebung zu realisieren sind, um die Angebotsschnittstellen einer Komponente teilweise zu nutzen bzw. welche Angebotsschnittstellen nutzbar sind, falls nur ein Teil der festgelegten Nachfrageschnittstellen von der Umgebung realisiert werden kann [REUSSNER 2001:61]. Im UNSCOM Metaschema werden parametrisierte Komponentenverträge durch *Relationen* zwischen den Methoden der Angebotsschnittstellen und den Methoden der Nachfrageschnittstellen der jeweiligen Komponente dargestellt (vgl. Abb. 4.30). Diese werden durch das Schemaelement Komponentenvertrag zusammengefasst und einem bereits definierten Komponententyp hinzugefügt.

4.5.1.2 Zusicherungen

Als architekturspezifische Merkmale der Komponente sind auch die *Dienstverträge* zu beschreiben, die die (systemtechnischen) Bedingungen für die erfolgreiche Inanspruchnahme einzelner Methoden vorgeben. Jeder Dienstvertrag enthält eine Reihe von *Vorbe-*

dingungen, die vor dem Aufruf einer Methode vom Dienstnachfrager zu erfüllen sind, und *Nachbedingungen*, die vom Dienstanbieter nach Ausführung der Methode garantiert werden, falls ein korrekter Aufruf erfolgte (vgl. Abschnitt 3.3.1). Ein Dienstvertrag ist bspw. für die Methode buche zu definieren, über die sich Veränderungen im Lagerbestand eines Artikels erfassen lassen. Als Vorbedingung ist dabei festzulegen, dass der Betrag der zu dokumentierenden Veränderung stets größer als Null und im Falle einer Minderung (also einem Warenausgang) nicht größer als der aktuelle Lagerbestand zu sein hat. Im Gegenzug ist zu garantieren, dass der Lagerbestand nach der Ausführung des Dienstes um den beim Aufruf genannten Betrag erniedrigt bzw. erhöht wurde und nicht kleiner als Null ist. Die Beschreibung der Vor- und Nachbedingungen erfolgt durch *prädikatenlogische Aussagen*. Jede Aussage kann mit einem Namen versehen werden und legt einen Zustand fest, der vor bzw. nach der Ausführung der Methode zu gelten hat [MEYER 1997:337]. Zusammen beschreiben die Vor- und Nachbedingungen einer Methode so die (systemtechnischen) *Wirkungen* ihrer Ausführung [CHEESMAN und DANIELS 2001:125].

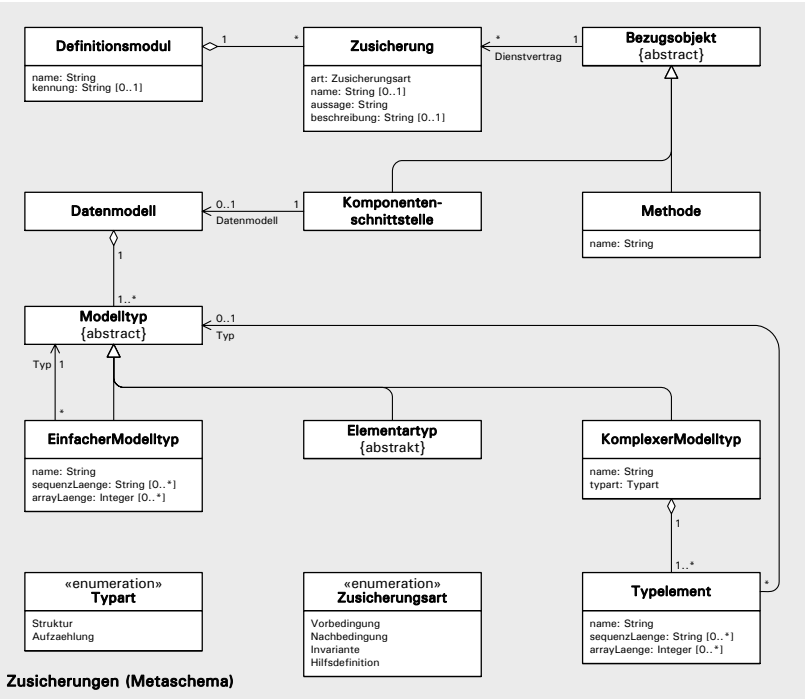


Abb. 4.31: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Zusicherungen.

Im UNSCOM Metaschema werden die logischen Aussagen durch das Schemaelement Zusicherung dargestellt und dem Modul zugeordnet, das die durch die Dienstverträge näher zu dokumentierende Komponentenschnittstelle enthält (vgl. Abb. 4.31). Für jede Zusicherung ist dabei zunächst festzulegen, ob es sich um eine *Vor- oder Nachbedingung* handelt. Aussagen (-teile), die sowohl vor als auch nach der Ausführung aller Methoden einer Schnitt-

stelle gelten, können außerdem als *Invarianten* gesondert dargestellt werden. Ferner ist ein Verweis auf den *Kontext* (das sog. Bezugsobjekt) anzugeben, auf den sich die Aussage bezieht. Bei Vor- und Nachbedingungen stellt das Bezugsobjekt stets eine der Methoden dar, die als Bestandteile der Schnittstellen vereinbart wurden. Invarianten beziehen sich dagegen ausschließlich auf Komponentenschnittstellen. Schließlich ist im Metaschema die eigentliche *prädikatenlogische Aussage* der Zusicherung festzuhalten und ggf. durch eine umgangssprachliche Beschreibung zu ergänzen. Die formale Aussage muss dabei der Grammatik gehorchen, die durch die standardisierte *Object Constraint Language* vorgegeben wird [OMG 2003a:60].

Bei der Formulierung der Aussagen darf grundsätzlich auf alle Komponenten-, Schnittstellen- und Methodenelemente des jeweiligen Kontexts Bezug genommen werden, die zuvor im Rahmen der vereinbarten Signaturen definiert worden sind. Darüber hinaus kann es zur Spezifikation von Zusicherungen allerdings notwendig sein, auf weitere (Hilfs-) Definitionen Bezug zu nehmen, etwa um verschiedene Laufzeitzustände anhand der von der Komponente verwalteten Daten darzustellen [CHEESMAN und DANIELS 2001:123-125]. Solche zusätzlichen Definitionen werden ggf. benötigt, um die Veränderungen des Zustands zu beschreiben, die beim Ausführen einer Methode jeweils verursacht werden [CHEESMAN und DANIELS 2001:123]. Das UNSCOM Metaschema erlaubt deshalb die Definition zusätzlicher anwendungsspezifischer Typen, die ausschließlich zur Spezifikation von Zusicherungen benötigt werden (vgl. Abb. 4.31). Diese Typdefinitionen werden von speziellen Modulen (unter der Bezeichnung Datenmodell) zusammengefasst und somit explizit als Hilfsdefinitionen gekennzeichnet. Aus Modularitätsgründen dürfen diese Module außerdem jeweils nur Definitionen enthalten, die für die Spezifikation der Zusicherungen *einer* Komponentenschnittstelle benötigt werden. Der Schnittstelle ist bei der Vereinbarung eines Datenmodells im UNSCOM Metamodell ein Verweis auf dieses hinzuzufügen. Anzumerken bleibt jedoch, dass sich die unter Verwendung des Datenmodells definierten Zusicherungen auch zur Laufzeit nicht mehr überprüfen lassen, da vor allem die festgelegten komponenteninternen Sachverhalte des Datenmodells wegen des Blackbox-Prinzips *nicht* der tatsächlichen Realisierung entsprechen müssen [CHEESMAN und DANIELS 2001:124].

4.5.1.3 Interaktionsprotokolle

Für die Kopplung von Komponenten und den Aufruf ihrer Methoden zur Laufzeit werden neben Signaturen und Zusicherungen auch Informationen über die *Reihenfolgeabhängigkeiten* benötigt, die zwischen den einzelnen Methoden bestehen. Typischerweise lassen sich die an den Schnittstellen aufgeführten Methoden nämlich nicht in beliebigen, sondern nur in bestimmten Reihenfolgen aufrufen, die sich alleine aus den vereinbarten Signaturen jedoch nicht ermitteln lassen [NIERSTRASZ 1995:102]. Für jede Komponente sind deshalb

Interaktionsprotokolle zu vereinbaren, die die erlaubten Reihenfolgen beim Aufruf von Methoden festlegen. Dabei ist zwischen Angebots- und Nachfrageprotokollen zu unterscheiden: *Angebotsprotokolle* legen die Aufrufreihenfolge der Methoden fest, die an den Angebotsschnittstellen für die Benutzung von außen bereitgestellt werden, während *Nachfrageprotokolle* die Aufrufreihenfolge der Methoden beschreiben, die während der Informationsverarbeitung über die Nachfrageschnittstellen in Anspruch genommen werden.

Im Allgemeinen besitzen Komponenten mehrere voneinander unabhängige Angebots- und Nachfrageprotokolle. Da Reihenfolgebeziehungen häufig nur zwischen thematisch zusammengehörigen Methoden auftreten, die jeweils zu einer Schnittstelle zusammengefasst sind, lassen sich die Schnittstellen von Komponenten bei Interaktionen vielfach unabhängig voneinander verwenden. Jede Schnittstelle ist somit zunächst als eigenständiger Ausgangs- bzw. Endpunkt für Interaktionen zu betrachten, der im konzeptionellen Komponentenmodell als *Port* bezeichnet wird (vgl. Abschnitt 3.1.4). So werden bspw. die beiden Angebotsschnittstellen ILagervwt und ILogistik der Komponente Lagermanagement unabhängig voneinander verwendet. Dementsprechend ist jeder Schnittstelle ein eigenständiges Interaktionsprotokoll zuzuordnen, das den Verlauf der jeweils unterstützten Interaktionen vorgibt. Bestehen hingegen Abhängigkeiten zwischen den Methoden mehrerer Angebots- oder mehrerer Nachfrageschnittstellen, bilden diese einen komplexen Port. Ihre Angebotsprotokolle sind dann mit einem sog. *Verbindungsprotokoll*, das die Abhängigkeiten zwischen den Methoden der verschiedenen Schnittstellen beschreibt, zu einem gemeinsamen Protokoll zu verbinden. Abhängigkeiten bestehen bspw. zwischen den Methoden der Angebotsschnittstellen ILogistik und IBestandsvwt, die zur Komponente Lagermanagement gehören. Nach jeder Aus- bzw. Einlagerung von Artikeln über die Schnittstelle ILogistik ist nämlich die Bestandsführung über die Schnittstelle IBestandsvwt fortzuschreiben.

Mit dem UNSCOM Spezifikationsrahmen sind die Interaktionsprotokolle einer Komponente grundsätzlich als *deterministische endliche Automaten* [SCHÖNING 1995:27f.] zu beschreiben. Für jedes Protokoll ist im Metaschema folglich eine Menge von *Zuständen* und *Übergängen* anzugeben (vgl. Abb. 4.32). Darüber hinaus sind ein Zustand als *Startzustand* und mindestens ein Zustand als *Endzustand* zu kennzeichnen. Die festgelegten Übergänge stellen die Zustandswechsel dar, die bei den Instanzen einer Komponente zur Laufzeit durch die Benutzung von Methoden im Rahmen der kooperativen Informationsverarbeitung ausgelöst werden. Anders als bei einem Prozess stehen beim Interaktionsprotokoll somit nicht das Verrichtungsobjekt, sondern das *Handlungssubjekt* und seine jeweiligen Zustände im Mittelpunkt. Bei der Modellierung von Übergängen wird in Bezug auf die jeweils zugrunde liegende Benutzung einer Methode unterschieden zwischen der (vollständigen) *Ausführung*, dem *Aufruf* und dem *Abschluss*. Die Methodenausführung beinhal-

tet sowohl den Aufruf als auch den Abschluss und stellt die Benutzung einer Methode nach außen als atomare Aktion dar. Durch eine getrennte Modellierung des Aufrufs und des Abschlusses lässt sich beschreiben, welche weiteren Aktionen ggf. erlaubt sind, während die Methode ausgeführt wird. Dies kann bspw. notwendig sein, um Nebenläufigkeitsaspekte darzustellen. Jedem Übergang lässt sich außerdem eine Reihe von *Wahrscheinlichkeitswerten* zuordnen. Diese geben die (bei der Qualitätsvorhersage genutzte) Wahrscheinlichkeit an, mit der ein Übergang während einer Interaktion auftritt, und erlauben es, das spezifizierte Protokoll als sog. *Markow-Kette* zu interpretieren.

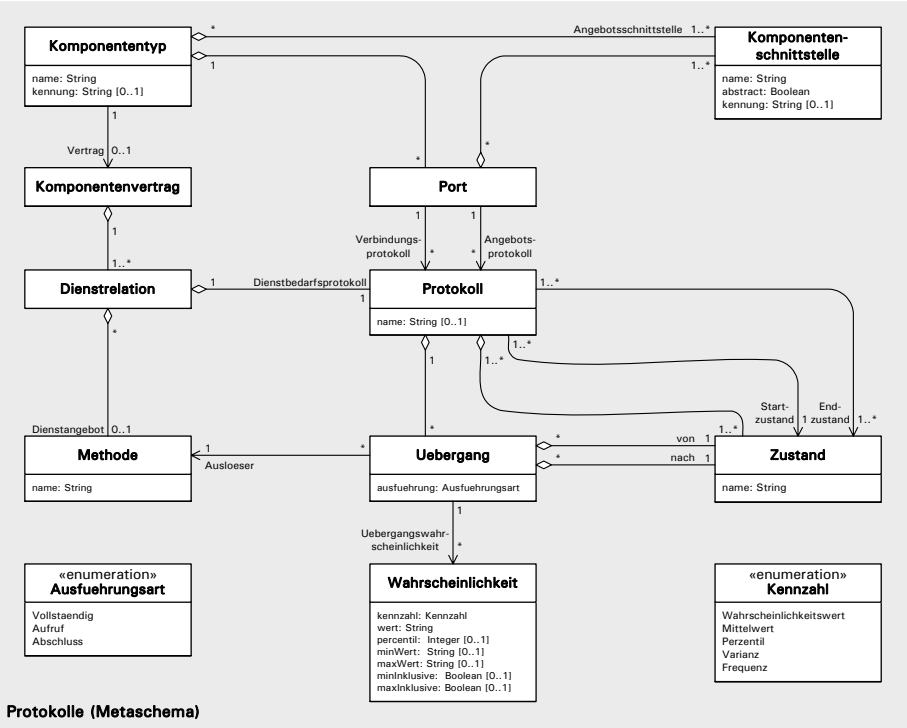


Abb. 4.32: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von Protokollen.

Im UNSCOM Metaschema wird ferner zwischen Angebotsprotokollen und Verbindungsprotokollen, die mehrere Angebotsprotokolle zu einem komplexen Interaktionsprotokoll zusammenfassen, unterschieden (vgl. Abb. 4.32). *Angebotsprotokolle* beschreiben Reihenfolgeabhängigkeiten, die zwischen den Methoden einzelner Angebotsschnittstelle bestehen. Sie werden im Metaschema den entsprechenden Angebotsschnittstellen zugeordnet, die als elementare Ports der Komponente betrachtet werden. *Verbindungsprotokolle* beschreiben Reihenfolgeabhängigkeiten zwischen den Methoden mehrerer Angebotsschnittstellen. Sie werden im Metaschema den entsprechenden Schnittstellen zugeordnet, die zusammen jeweils einen komplexen Port bilden. Die *Nachfrageprotokolle* einer Komponente werden dagegen nicht statisch festgelegt, sondern nach dem Konzept des parametrisierten Komponentenvertrags [REUSSNER 2001:120] in Abhängigkeit von den tatsächlich benutz-

ten Methoden der Angebotsschnittstellen berechnet. Für die Methoden der Angebotsschnittstellen ist deshalb jeweils anhand eines Protokolls zu beschreiben, in welcher Reihenfolge im Rahmen ihrer Ausführung Methoden von anderen Komponenten in Anspruch genommen werden. Diese sog. *Dienstbedarfsprotokolle* [REUSSNER 2001:114] werden im Metaschema dem parametrisierten Komponentenvertrag zugeordnet (vgl. Abb. 4.32). Da sie zur Spezifikation der Dienstrelationen genutzt werden, ersetzen sie die Liste der nachgefragten Dienste, die in Abb. 4.30 als alternative Beschreibungstechnik vorgestellt wurde. Aus den Dienstbedarfsprotokollen und den bei der Interaktion genutzten Angebots- bzw. Verbindungsprotokollen können die *Nachfrageprotokolle* der Komponente berechnet werden [REUSSNER 2001:120]. Sie müssen zu den Angebotsprotokollen der externen Komponenten kompatibel sein, damit eine Komposition durchgeführt werden kann.

Anzumerken bleibt, dass die Ausdruckskraft deterministischer endlicher Automaten grundsätzlich eingeschränkt ist, so dass viele Aspekte nicht in die Protokollbeschreibung einbezogen werden können. Bspw. lässt sich nicht spezifizieren, dass im Rahmen der Auftragsabwicklung nur Kundenaufträge gelöscht werden dürfen, die zuvor angelegt wurden. Ebenso lässt sich nicht festlegen, dass einem Kellerspeicher nur so viele Elemente entnommen werden dürfen wie zuvor abgelegt wurden [NIERSTRASZ 1995:109f.]. Im Allgemeinen beschreiben deterministische endliche Automaten somit nur eine Obermenge der erlaubten Aufrufreihenfolgen, die ggf. auch unzulässige Reihenfolgen enthalten kann [NIERSTRASZ 1995:109]. Deswegen sind die Protokollbeschreibungen durch zusätzliche Vor- und Nachbedingungen zu präzisieren, mit denen sich weitere Einschränkungen formulieren lassen.

Für die Verwendung von Automaten zur Protokollbeschreibung spricht indes vor allem ihre effiziente Auswertbarkeit, etwa im Rahmen statischer Kompatibilitätstests und Qualitätsvorhersagen [GRIFFEL 1998:72]. Infolgedessen bilden sie die Grundlage für zahlreiche Methoden, die derzeit im Rahmen der anzustrebenden Entwicklungsmethodik entstehen [YELLIN und STROM 1994; NIERSTRASZ 1995; REUSSNER und SCHMIDT 2002; SCHMIDT und REUSSNER 2002; WALLNAU 2003]. Im Rahmen des UNSCOM Spezifikationsrahmens wurde somit ein Kompromiss zwischen der Ausdruckskraft der eingesetzten Notation und ihrer Anwendbarkeit eingegangen. Die Einbeziehung derzeit erforschter Notationen, mit denen sich (bspw. auf Basis einer Prozessalgebra, eines Petrinetzes oder einer temporalen Logik) auch komplexere Protokolle beschreiben lassen, bleibt demgegenüber eines der Ziele, die im Rahmen einer Weiterentwicklung längerfristig umgesetzt werden sollen.

Bei der Einbeziehung derartiger Notationen ist vor allem auf ihre Anwendbarkeit zu achten, da mit Vor- und Nachbedingungen bereits eine Spezifikationstechnik vorhanden ist, mit der prinzipiell auch komplexe Protokolle beschrieben werden könnten [MEYER

1997:982; CHEESMAN und DANIELS 2001:42]. Gerade Vor- und Nachbedingungen lassen sich aufgrund ihrer Ausdrucksmächtigkeit jedoch nur sehr eingeschränkt statisch auswerten [MEYER 1997:578], was deren Verwendung im Rahmen der angestrebten Entwicklungsmethodik erschwert. Der mit dem UNSCOM Spezifikationsrahmen eingegangene Kompromiss zwischen Ausdruckskraft und Anwendbarkeit stellt dagegen eine Vorgehensweise dar, die die Protokollspezifikation zumindest teilweise einer statischen Auswertung zugänglich macht und sie durch Vor- bzw. Nachbedingungen ergänzt. Diese Vorgehensweise wird auch im Rahmen der UML 2.0 mit der Einführung sog. *Protokollzustandsautomaten* umgesetzt [OMG 2003b:466].

4.5.2 Repräsentation

Zur Repräsentation wird sowohl im UNSCOM/T als auch im UNSCOM/G Format jeweils eine Reihe von Notationen genutzt, die die geeignete Darstellung der nach außen sichtbaren systemtechnischen Struktur und des Verhaltens gewährleisten. Während hierfür im UNSCOM/G Format mit Komponenten- und Typdiagrammen (zur Darstellung der Struktur) sowie OCL Bedingungen und Zustandsdiagrammen (zur Darstellung des Verhaltens) jedoch Notationen verwendet werden, die zur UML 2.0 [OMG 2003a; OMG 2003b] gehören, kann zur Darstellung im UNSCOM/T Format nur bedingt auf standardisierte Notationen zurückgegriffen werden. Mit der CORBA Interface Definition Language (IDL) [OMG 2004] wird eine standardisierte Sprache zur Repräsentation der Struktur verwendet. Als Programmiersprache unterstützt die CORBA IDL eine plattformunabhängige Beschreibung von Signaturen, da ihre Konstrukte mittels vordefinierter Übersetzungsregeln (sog. *Mappings* [OMG 2004:2-8f.]) in äquivalente Konstrukte der gängigen Programmiersprachen überführt werden können. Für die Darstellung der Interaktionsprotokolle ist dagegen eine neue Notation zu definieren. Neben der standardisierten OCL, mit der die Vor- und Nachbedingungen dargestellt werden, kommt bei der Beschreibung des Verhaltens in UNSCOM/T somit auch eine proprietäre Notation zum Einsatz.

Nachfolgend werden die Formatelemente vorgestellt, die zur Repräsentation der Architekturmerkmale in UNSCOM/T und UNSCOM/G verwendet werden. Grundsätzlich ist in Bezug auf die Darstellung zwischen Signaturen, Zusicherungen und Protokollen zu unterscheiden, die getrennt voneinander betrachtet werden. Bei der Vorstellung werden zueinander korrespondierende Elemente des UNSCOM/T und UNSCOM/G Formats jeweils gegenübergestellt, um deren Äquivalenz aufzuzeigen. Die Repräsentation der architekturspezifischen Komponentenmerkmale in UNSCOM/X wird dagegen im Anhang betrachtet. Sie orientiert sich an den Vorgaben eines XML Schemas, das in Anhang B aufgeführt ist.

4.5.2.1 Signaturen

Thematisch zusammengehörige Signaturen sind in UNSCOM/T durch das Schlüsselwort module der CORBA IDL zu *Gruppen* zusammenfassen [OMG 2004:3-20]. Anders als durch die CORBA IDL zugelassen, muss die Vereinbarung von Signaturen gemäß den Vorgaben des UNSCOM Metaschemas stets mit der Definition eines Moduls beginnen (vgl. Abb. 4.33). Bei der Definition eines Moduls ist ein Modulname anzugeben. Dieser erzeugt einen *Namensraum*, der zur eindeutigen Bezeichnung der Modulelemente beiträgt: so deutet Modulname::Elementname darauf hin, dass ein Element eines bestimmten Moduls gemeint ist. Als Elemente sind (neben untergeordneten Modulen) die jeweils zusammengefassten Signaturen zu definieren und in geschweiften Klammern aufzuführen. Bei der Definition dieser Signaturen kann auch Bezug auf solche Signaturen genommen werden, die von anderen Modulen zusammengefasst werden. In diesem Fall ist vor der Definition des Moduls allerdings ein *Import* der entsprechenden Module vorzunehmen. Ein Import wird durch das gleichnamige Schlüsselwort import eingeleitet [OMG 2004:3-19f.], wobei (wiederum in Abweichung vom IDL Standard) ausschließlich Module genannt werden dürfen.

In UNSCOM/G sind Signaturdefinitionen durch UML Pakete zu *Gruppen* zusammenzufassen (vgl. Abb. 4.34). Wie Module erzeugen UML Pakete dabei mit ihrem Namen jeweils einen *Namensraum*, der den Namen der zusammengefassten Signaturen vorangestellt wird und so zu deren eindeutiger Bezeichnung beiträgt [OMG 2003b:99]. Jedes Paket ist graphisch als Rechteck so darzustellen, dass es seine Signaturen umfasst [OMG 2003b:100f.]. Alternativ lassen sich die Signaturen dem Paket jedoch auch über eine spezielle Enthaltenseinsbeziehung zuordnen oder durch Voranstellen des Modulnamens vor den Elementnamen zuordnen [OMG 2003b:101]. Schließlich können andere Pakete einbezogen werden, damit bei der Definition von Signaturen auf die dort enthaltenen Signaturen Bezug genommen werden kann. Die *Einbeziehung* eines anderen Pakets ist mit Hilfe einer UML Abhängigkeitsbeziehung darzustellen, die mit dem Stereotyp «import» zu versehen ist [OMG 2003b:131, 138f.].

Als Bestandteile von Signaturgruppen können neben Schnittstellen, Ereignissen und Komponententypen vor allem anwendungsspezifische Typen, Konstanten und Ausnahmen definiert werden. *Anwendungsspezifische Typen* lassen sich in UNSCOM/T durch die Schlüsselwörter typedef, struct und enum der IDL vereinbaren (vgl. Abb. 4.33). Als Ausgangspunkt für die Definition von Typen stellt die IDL vordefinierte, elementare Datentypen zur Verfügung, wobei Ganzzahlen als long, Fließkommazahlen als double, Zeichenketten als string und Wahrheitswerte als boolean bezeichnet werden [OMG 2004:3-37f.]. Neben den hier aufgeführten bietet die IDL weitere elementare Datentypen an, die Spezialfälle der genannten Typen darstellen. Diese sind in die allgemeinen Typen zu überführen, die im

UNSCOM Metaschema enthalten sind. Das Schlüsselwort typedef ermöglicht einfache Typdefinitionen, die einen einzelnen Wert oder eine Gruppe von Werten eines Typs unter einem neuen Namen zusammenfassen [OMG 2004:3-36]. Die Schlüsselwörter struct und enum erlauben dagegen komplexe Typdefinitionen, mit denen sich strukturierte Typen bzw. Aufzählungstypen vereinbaren lassen [OMG 2004:3-40f.]. Hierzu sind neben dem Namen des neuen Typs ein Name und – bei strukturierten Typen – ein Typ für jede charakteristische Eigenschaft in geschweiften Klammern anzugeben. Bei einfachen Typdefinitionen und der Vereinbarung von strukturierten Typen ist außerdem anzugeben, wie viele Werte ggf. als Array oder Sequenz zu einer Gruppe zusammengefasst werden.

```
module Lagerorganisation {

  const long LagerzahlLimit = 5;

  typedef string ID;

  typedef long Bestand;

  enum Lagereinheit {
    Artikel,
    Europalette
  };

  struct Lagerplatztyp {
    string       name;
    long         hoehe;
    long         breite;
    long         tiefe;
    long         tragfaehigkeit;
    long         maximalbelegung;
    Lagereinheit einheit;
  };

  struct Lagerplatz {
    string  typ;
    string  gangNummer;
    string  regalNummer;
    string  ebenenNummer;
    Bestand b;
  };

  exception AenderungVerweigert {string error;};

  exception LoeschungVerweigert {string error;};

  ...
};
```

Abb. 4.33: Spezifikation der Module, Typen, Konstanten und Ausnahmen in UNSCOM/T.

In UNSCOM/G werden anwendungsspezifische Typen als UML Datentypen bzw. Typen dargestellt (vgl. Abb. 4.34). Die graphische Darstellung erfolgt dabei jeweils durch ein UML Klassenkonstrukt, das mit dem Stereotyp «data type» bzw. «type» zu versehen ist. Ein UML Datentyp ist immer dann zur Repräsentation zu verwenden, wenn seine Ausprägungen zur Laufzeit ausschließlich aus einfachen Werten bestehen [OMG 2003b:95], wäh-

rend UML Typen (zumindest partiell) eine Referenzsemantik besitzen. Als vordefinierte elementare Datentypen stellt die UML 2.0 Ganzzahlen (Integer), Fließkommazahlen (Real), Zeichenketten (String) und Wahrheitswerte (Boolean) bereit, die durch UML Klassenkonstrukte mit dem Stereotyp »primitive« graphisch darzustellen sind [OMG 2003b:537]. Unter Verwendung dieser Konstrukte lassen sich einfache Typdefinitionen in UNSCOM/G durch gerichtete UML Assoziationen darstellen. Sie verbinden je einen neu definierten Typ mit dem Typen, der als Ausgangspunkt für die Ableitung dient. Die Assoziation ist mit dem Stereotyp «implements» zu kennzeichnen und ggf. mit einer Kardinalität zu versehen, falls bei der Typdefinition mehrere Werte zu einem Feld oder einer Sequenz zusammengefasst werden (vgl. Abb. 4.34).

Bei komplexen Typdefinitionen sind die charakteristischen Eigenschaften als Attribute in die Attributregion des Klassenkonstrukts einzutragen, das den vereinbarten Typen darstellt. Wird ein strukturierter Typ definiert, sind neben dem Namen auch der Typ der jeweiligen Eigenschaft und ggf. eine Kardinalität anzugeben, falls von der Eigenschaft mehrere Werte zu einer Gruppe zusammengefasst werden. Anstelle des Eintrags in die Attributregion kann bei der Definition strukturierter Typen auch eine (gerichtete) UML Assoziation verwendet werden, die den neu zu definierenden Typ mit dem Typ der Eigenschaft verbindet. Der Name der Eigenschaft ist dann als Rolle des assoziierten Typen anzugeben. Bei der Vereinbarung von Aufzählungstypen ist das Stereotyp des Klassenkonstrukts in «enumeration» zu ändern [OMG 2003b:96f.].

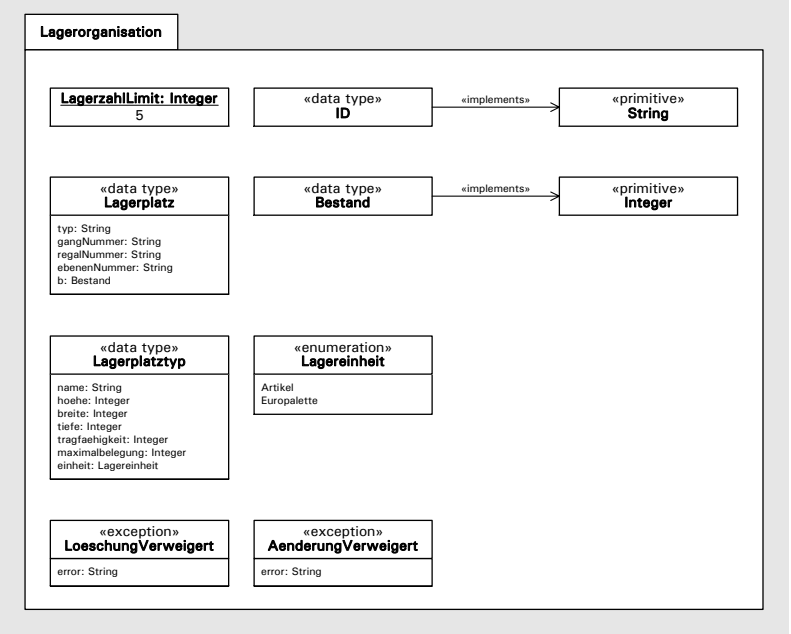


Abb. 4.34: Spezifikation der Pakete, Typen, Konstanten und Ausnahmen in UNSCOM/G.

Konstanten sind in UNSCOM/T unter Verwendung des IDL Schlüsselworts const darzustellen [OMG 2004:3-32f.]. Dabei ist neben dem Namen und dem Typen der Konstante auch die Berechnungsvorschrift anzugeben (vgl. Abb. 4.33). Sie kann entweder aus einem Wert oder einer Aussage bestehen [OMG 2004:3-34f.]. In UNSCOM/G sind Konstanten als UML Instanzen darzustellen [OMG 2003b:58f.]. Ihr Name ist mit dem Datentyp zu ergänzen, der der Konstanten zugrunde liegt (vgl. Abb. 4.34). Darüber hinaus ist die Berechnungsvorschrift anzugeben, die den konstanten Wert festgelegt.

Die zu vereinbarenden *Ausnahmen* lassen sich in UNSCOM/T schließlich durch das Schlüsselwort exception definieren [OMG 2004:3-49f.]. Analog zur Definition eines strukturierten Typs ist dabei der Name der Ausnahme festzulegen. Darüber hinaus sind die charakteristischen Eigenschaften in geschweiften Klammern anzugeben (vgl. Abb. 4.33). Für jede Eigenschaft sind wiederum ein Name und ein Typ zu vereinbaren. Ferner ist ggf. anzugeben, wie viele Werte von der Eigenschaft als Array oder Sequenz zu einer Gruppe zusammengefasst werden. In UNSCOM/G sind Ausnahmen durch Klassenkonstrukte darzustellen, die mit dem Stereotyp «exception» gekennzeichnet sind. Wie bei der Definition strukturierter Datentypen sind die charakteristischen Eigenschaften dabei als Attribute des Klassenkonstrukts zu behandeln und mit einer Kardinalität zu versehen, falls mehrere Werte von einer Eigenschaft zusammengefasst werden (vgl. Abb. 4.34).

Komponentenschnittstellen sind in UNSCOM/T mit dem Schlüsselwort interface zu vereinbaren, wobei durch Voranstellen des optionalen Schlüsselworts abstract festgelegt werden kann, dass es sich um eine abstrakte Schnittstelle handelt [OMG 2004:3-21]. Daran anschließend ist der Name der Schnittstelle anzugeben, der analog zum Modul einen Namensraum für die Schnittstellenelemente erzeugt (vgl. Abb. 4.35). Optional lässt sich hinter dem Namen der Schnittstelle eine Menge von Schnittstellen angeben, die im Rahmen von Vererbungsbeziehungen spezialisiert werden [OMG 2004:3-21f.]. In geschweiften Klammern sind schließlich die Elemente der Schnittstelle, d.h. lokale Definitionen von Typen, Konstanten und Ausnahmen sowie die Attribute und Methoden anzugeben [OMG 2004:3-22]. Zur Definition von Typen, Konstanten und Ausnahmen sind dabei die bereits genannten Schlüsselwörter zu verwenden. *Attribute* sind hingegen durch das Schlüsselwort attribute zu vereinbaren, dem das Schlüsselwort readonly voranzustellen ist, falls es sich um ein unveränderliches Attribut handelt [OMG 2004:3-53f.]. Neben dem Namen und dem Typ des Attributs können bei der Definition optional die Ausnahmen genannt werden, die beim Lesezugriff (sowie beim Schreibzugriff auf veränderliche Attribute) ggf. ausgesandt werden. Die Ausnahmen sind bei unveränderlichen Attributen hinter dem Schlüsselwort raises und bei veränderlichen Attributen hinter den Schlüsselwörtern getraises (Lesezugriff) bzw. setraises (Schreibzugriff) aufzuführen [OMG 2004:3-54]. Die *Methoden* der

Komponentenschnittstelle sind zu definieren, indem jeweils ein Typ für die Ergebniswerte, ein Methodenname, eine Liste von Parametern sowie optional eine Menge von Ausnahmen festgelegt wird (vgl. Abb. 4.35). Werden von einer Methode keine Ergebniswerte zurückgegeben, ist bei der Definition als Ergebnistyp void anzugeben [OMG 2004:3-50]. Die Liste der Parameter folgt auf den Namen der Methode und ist in runden Klammern anzugeben [OMG 2004:3-50]. Für jeden Parameter ist dabei zunächst die Art der Übergabe durch eines der Schlüsselwörter in, out bzw. inout festzulegen. Darüber hinaus sind ein Typ und ein Parametername anzugeben [OMG 2004:3-51]. Die Menge der bei der Methodenausführung ggf. erzeugten Ausnahmen ist nach der Liste der Parameter festzulegen und mit dem Schlüsselwort raises einzuleiten [OMG 2004:3-50].

```
interface IBestandsvwt {
  enum Aenderungstyp {
    Wareneingang,
    Warenausgang
  };

  struct Bestandsveraenderung {
    ID            konto;
    ID            auftrag;
    Aenderungstyp typ;
    long          menge;
  };

  exception BestandFehlt {};
  exception ReservierungUngueltig {};
  exception AuftragUngueltig {};
  exception KontoUngueltig {};

  void    setzeMindestbestand(in ID konto, in Bestand b) raises (KontoUngueltig);
  Bestand aktuellerBestand(in ID konto) raises (KontoUngueltig);
  Bestand physischerBestand(in ID konto) raises (KontoUngueltig);
  ID      reserviere(in ID konto, in Bestand b)
          raises (KontoUngueltig, BestandFehlt);
  Bestand buche(in Bestandsveraenderung b, in ID reservierung)
          raises (KontoUngueltig, ReservierungUngueltig, AuftragUngueltig);
  void    loescheReservierung(in ID reservierung) raises (ReservierungUngueltig);
};

eventtype MindestbestandUnterschritten {ID artikel;};

interface IDatenbank {
  exception VerbindungFehlgeschlagen {string error;};
  exception SQLAusdruckUngueltig {string error;};

  void oeffneVerbindung(in string datenbank) raises (VerbindungFehlgeschlagen);
  void beginneTransaktion();
  void verarbeiteSQL(in string sqlAusdruck) raises (SQLAusdruckUngueltig);
  void rollback();
  void beendeTransaktion();
  void schliesseVerbindung();
};


component Lagermanagement {
  provides  IBestandsvwt __IBestandsvwt;
  uses      IDatenbank __IDatenbank;
  publishes MindestbestandUnterschritten __MindestbestandUnterschritten;
};
```

Abb. 4.35: Spezifikation der Komponentenschnittstellen, Ereignisse und Komponententypen in UNSCOM/T.

In UNSCOM/G sind die Komponentenschnittstellen als Klassenkonstrukte darzustellen, die mit dem Stereotyp «interface» gekennzeichnet sind [OMG 2003b:115]. Optional kann die-

sem Stereotyp die Eigenschaft {abstract} hinzugefügt werden, um die Vereinbarung einer abstrakten Schnittstelle zu kennzeichnen. Vererbungsbeziehungen zu anderen Schnittstellen sind in UNSCOM/G durch UML Spezialisierungen darzustellen, die die zu definierende Schnittstelle mit der jeweiligen Generalisierung verbinden. Die *Attribute* der zu vereinbarenden Schnittstelle sind in die Attributregion des Klassenkonstrukts einzutragen und so als Schnittstellenattribute kenntlich zu machen [OMG 2003b:114]. Für jedes Attribut sind ein Name, ein Typ und ggf. eine Kardinalität anzugeben, falls mehrere Werte zu einer Gruppe zusammengefasst werden. Optional kann durch die Eigenschaft {readonly} außerdem angezeigt werden, dass es sich um ein unveränderliches Attribut handelt. Anstelle des Eintrags in die Attributregion kann grundsätzlich auch eine (gerichtete) UML Assoziation verwendet werden, die die neu zu definierende Schnittstelle mit dem Typ des Attributs verbindet. Der Name des Attributs ist dann als Rolle des assoziierten Typen anzugeben. Die *Methoden* der Schnittstelle sind dagegen in die Methodenregion des Klassenkonstrukts einzutragen (vgl. Abb. 4.36). Für jede Methode ist ein Name, eine Parameterliste in runden Klammern sowie ein optionaler Rückgabewert anzugeben [OMG 2003b:114]. Ihre Parameter bestehen aus einem Namen, einem Typ und ggf. aus einer Kardinalität, falls mehrere Werte zu einer Gruppe zusammengefasst werden [OMG 2003b:77]. Durch die Schlüsselwörter in, out und inout ist zudem die Art der Übergabe festzulegen [OMG 2003b:74].

Die mit der Schnittstelle lokal vereinbarten Typen, Konstanten und Ausnahmen sind in UNSCOM/G unter Verwendung der bereits vorgestellten Formatelemente darzustellen. Allerdings ist jede lokale Definition durch eine UML Enthaltenseinsbeziehung ausdrücklich der Schnittstelle als Bestandteil zuzuordnen. Alternativ dazu kann den Namen der lokalen Definitionen der Name der jeweiligen Schnittstelle vorangestellt werden (vgl. Abb. 4.36). Durch Schnittstellenname::Elementname wird angezeigt, dass die Vereinbarungen als Elemente der Schnittstelle zu betrachten sind. Anzumerken bleibt, dass durch den UML Standard keine Vorgehensweise für die Zuordnung von Ausnahmen zu Methoden bzw. Attributen festgelegt wird. In UNSCOM/G sind die erzeugten Ausnahmen dem jeweiligen Element als Eigenschaften hinzuzufügen. Hierzu ist zunächst das Schlüsselwort exceptions (bzw. getexceptions oder setexceptions bei veränderlichen Attributen) zu verwenden und anschließend die Liste der Ausnahmen zu nennen: {exceptions = *Name* (, *Name*)*}.

Die Definition von *Ereignissen* entspricht sowohl in UNSCOM/T als auch in UNSCOM/G der Definition von Schnittstellen. In UNSCOM/T ist deshalb lediglich das Schlüsselwort interface gegen das Schlüsselwort eventtype zu ersetzen [OMG 2004:3-57]. Anzumerken bleibt, dass es prinzipiell mit der OMG IDL möglich ist, weitere Eigenschaften für Ereignissen zu spezifizieren, die von denen einer Komponentenschnittstelle abweichen [OMG 2004:3-57]. Wegen der angestrebten Gleichbehandlung von Schnittstellen und Ereignissen

werden diese Eigenschaften vom UNSCOM Spezifikationsrahmen jedoch nicht unterstützt. In UNSCOM/G sind Ereignisse analog zu Schnittstellen durch Klassenkonstrukte darzustellen. Die Klassenkonstrukte sind allerdings mit dem Stereotyp «event» zu kennzeichnen.

Die Definition von *Komponententypen* erfolgt in UNSCOM/T mit Hilfe des Schlüsselworts component [OMG 2004:3-58]. Neben dem Namen sind dabei die Menge der Angebots-, die Menge der Nachfrageschnittstellen, die Menge der ausgesandten sowie die Menge der empfangbaren Ereignisse als Elemente der Komponente zu spezifizieren [OMG 2004:3-60]. Zur Festlegung der Komponentenelemente stehen die IDL Schlüsselwörter provides, uses, publishes bzw. consumes zur Verfügung (vgl. Abb. 4.35). Anders als das UNSCOM Metaschema lässt die IDL allerdings zu, dass Komponentenschnittstellen bzw. Ereignisse mehrmals in jeder der Mengen aufgeführt werden dürfen. Für jede Schnittstelle und jedes Ereignis ist daher neben dem Typ ein eindeutiger Name festzulegen, der einen elementaren Port für das jeweilige Element erzeugt [OMG 2004:3-61]. Nach den Vorgaben des UNSCOM Metaschemas werden Ports jedoch nicht mit eigenen Namen versehen. Der zu vergebende Name muss deshalb standardmäßig dem jeweiligen Elementtypnamen (ergänzt um ein vorangestelltes „__“) entsprechen. Zudem darf jedes Element in einer Menge höchstens einmal enthalten sein. Im Gegensatz zur IDL erlaubt der UNSCOM Spezifikationsrahmen auch keine Vererbungsbeziehungen zwischen Komponenten, weswegen zur Repräsentation nur eine Untermenge der IDL Konstrukte verwendet werden darf.

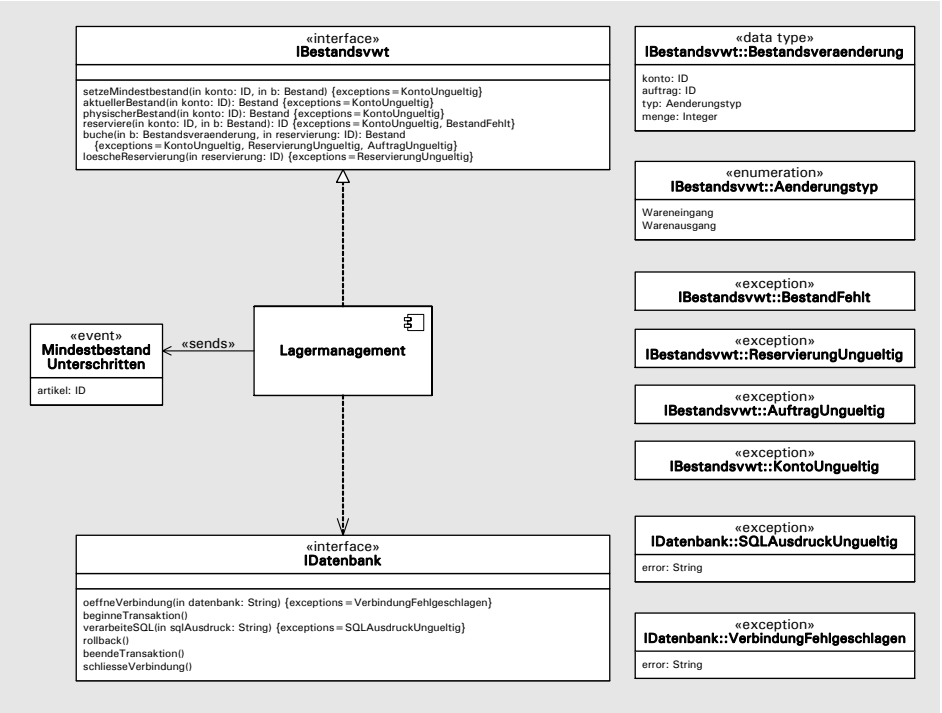


Abb. 4.36: Spezifikation der Komponentenschnittstellen, Ereignisse und Komponententypen in UNSCOM/G.

In UNSCOM/G sind Komponententypen als UML Komponenten in der sog. Blackbox-Sicht graphisch darzustellen [OMG 2003b:139]. Jede UML Komponente wird durch ein Klassenkonstrukt mit dem Stereotyp «component» repräsentiert, wobei das Stereotyp durch ein spezielles Komponentensymbol ersetzt werden kann [OMG 2003b:138]. Bei der Darstellung in UNSCOM/G sind neben dem Namen des Komponententyps die Angebots- und Nachfrageschnittstellen sowie die ausgesandten und empfangbaren Ereignisse anzugeben (vgl. Abb. 4.36). Für die Angebots- und Nachfrageschnittstellen ergeben sich dabei mehrere alternative Darstellungsmöglichkeiten. In der *verkürzten Darstellungsweise* werden sie durch spezielle Symbole repräsentiert, die an die Komponente angeheftet sind: Angebotsschnittstellen sind dabei als sog. Bälle und Nachfrageschnittstellen als sog. Sockets darzustellen [OMG 2003b:139, 147f.]. Optional lassen sich bei dieser Darstellungsweise auch die Ports explizit als Rechtecke graphisch darstellen und der Komponente hinzufügen. In der *ausführlichen Darstellungsweise* werden Schnittstellen durch Klassenkonstrukte repräsentiert und mit dem Komponententyp in Beziehung gesetzt (vgl. Abb. 4.36). Die Angebotsschnittstellen sind durch UML Implementierungsbeziehungen und die Nachfrageschnittstellen durch UML Abhängigkeitsbeziehungen mit dem Komponententyp zu verbinden [OMG 2003b:139]. Neben den genannten existiert eine weitere Darstellungsform, bei der die Schnittstellen in die Attributregion des Klassenkonstrukts eingetragen werden, das den Komponententyp repräsentiert. Dabei ist den Angebotsschnittstellen das Stereotyp «provided interfaces» und den Nachfrageschnittstellen «required interfaces» voranzustellen [OMG 2003b:139]. Ereignisse sind hingegen grundsätzlich als Klassenkonstrukte darzustellen und durch UML Assoziationen mit dem Komponententyp zu verbinden. Die Assoziationen sind mit dem Stereotyp «sends» bzw. «receives» zu kennzeichnen, um ausgesandte von empfangbaren Ereignissen zu unterscheiden.

Aufgrund der angestrebten Darstellbarkeit in CORBA IDL existieren jedoch einige Einschränkungen, die sich vor allem auf die Spezifikation mit der UML im Rahmen von UNSCOM/G auswirken. So ist es zunächst nicht möglich, denselben Namen bei unterschiedlicher *Groß- und Kleinschreibung* mehrfach zu verwenden, da die IDL nicht zwischen Groß- und Kleinschreibung unterscheidet [OMG 2004:3-67]. Darüber hinaus können grundsätzlich nur folgende UML Kardinalitäten auf IDL Konstrukte abgebildet werden: eine feste Zahl "*Zahl*" (als IDL Array), ein Bereich von Null bis zu einer festen Zahl "0..*Zahl*" (als IDL Sequenz mit Obergrenze) sowie eine beliebige Menge "0..*" bzw. "*" (als IDL Sequenz ohne Obergrenze). Zur Spezifikation in UNSCOM/G dürfen deshalb nur die eben genannten *Kardinalitäten* verwendet werden, um Werte zu Gruppen zusammenzufassen.

Zusammenfassungen von Werten zu Gruppen dürfen gemäß den Vorgaben des UNSCOM Metaschemas bei der Vereinbarung von Attributen, Parametern und Rückgabetypen, bei

einfachen Typdefinitionen sowie bei der Definition der charakteristischen Eigenschaften von strukturierten Typen und Ausnahmen erfolgen. Während die UML die Zusammenfassung von Werten mit Kardinalitäten wie gefordert unterstützt (vgl. Abb. 4.37 oben links), wird in der CORBA IDL jede Zusammenfassung als eigenständige einfache Typdefinition betrachtet. Deshalb darf bei der Vereinbarung von Attributen und Parametern grundsätzlich keine Zusammenfassung von Werten erfolgen. Bei der Vereinbarung der charakteristischen Eigenschaften von strukturierten Typen und Ausnahmen ist eine Zusammenfassung zwar erlaubt (vgl. Abb. 4.37 oben rechts). Sie wird jedoch wegen der dabei implizit vorgenommenen, zusätzlichen einfachen Typdefinition als *unerwünscht* betrachtet [OMG 2004:3-47]. Um Werte zu Gruppen zusammenzufassen, ist in UNSCOM/T deshalb prinzipiell eine eigenständige einfache Typdefinition vorzunehmen. Die in UNSCOM/G spezifizierbaren Zusammenfassungen sind in UNSCOM/T dabei nach der folgenden Regel als einfache Typdefinitionen darzustellen: für die Zusammenfassung wird ein einfacher Typ definiert und mit dem Namen des Elements versehen, das die Zusammenfassung eigentlich durchführen soll. Dieser Typ wird als behelfsmäßig gekennzeichnet, indem seinem Namen entweder __Seq, __Seq*Zahl* oder __Array*Zahl* angehängt wird (vgl. Abb. 4.37 unten).

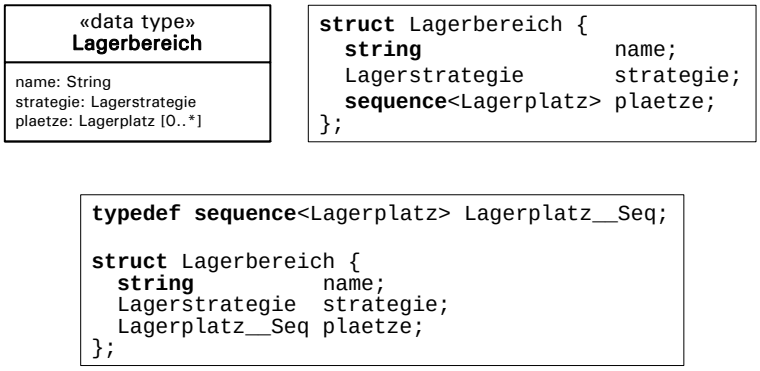


Abb. 4.37: Zusammenfassung von Werten zu Gruppen in UNSCOM/G und UNSCOM/T.

Auch die *Reihenfolge*, in der die Definitionen bei einer textbasierten Darstellung aufzuführen sind, ist aus der graphischen Darstellung in UNSCOM/G nicht ersichtlich. Die Elemente von Paketen, Schnittstellen, strukturierten Typen und Ausnahmen lassen sich deshalb mit Hilfe der Eigenschaft {order = *Zahl*} im Diagramm explizit ordnen. Durch die Zahl wird keine absolute Position, sondern nur eine Ordnung festgelegt. Verschiebungen in der Position ergeben sich bei der Darstellung in UNSCOM/T nämlich durch Definitionen, die z.B. zur Zusammenfassung von Werten behelfsweise vorzunehmen sind und in UNSCOM/G entfallen. Zu beachten ist jedoch, dass bei der Definition eines Elements mit der CORBA IDL nur auf Vereinbarungen Bezug genommen werden darf, die eine kleinere Ordnungsnummer besitzen. Alternativ kann die Ordnung durch eine *Namensliste* angegeben werden.

Abb. 4.38: Formatelemente zur Darstellung der Signaturdefinitionen in UNSCOM/T und UNSCOM/G.

Abb. 4.38 zeigt die Formatelemente, mit denen die verschiedenen Signaturdefinitionen in UNSCOM/T bzw. UNSCOM/G darzustellen sind, noch einmal im Detail. Die einander jeweils entsprechenden Formatelemente sind dabei gegenübergestellt und in Zusammenhang zu dem in Abschnitt 4.5.1.1 beschriebenen Metaschemaauszug gesetzt. Ergänzend zum Komponententyp kann schließlich optional noch ein parametrisierter Komponentenvertrag festgelegt werden. Dieser Vertrag ist durch eine proprietäre textbasierte Notation, der sog. *Component Contract Definition Language* (CCDL), zu beschreiben. Sie ist hinsichtlich der Syntax an die OCL angelehnt und sowohl im Rahmen von UNSCOM/T als auch von UNSCOM/G zu verwenden. Bei der Spezifikation ist zunächst der Komponententyp als Bezugsobjekt festzulegen. Anschließend ist für die Methoden der Angebotsschnittstellen die Liste der Methoden der Nachfrageschnittstellen anzugeben, die bei der Ausführung aufgerufen werden. Statt der Liste kann bei Vorhandensein auch ein Dienstbedarfsprotokoll genannt werden (vgl. Abb. 4.39). Den Abhängigkeiten ist jeweils das Schlüsselwort requires voranzustellen.

```
context Lagermanagement
  IBestandsvwt::setzeMindestbestand requires EinfacheDatenverarbeitung
  IBestandsvwt::aktuellerBestand    requires EinfacheDatenverarbeitung
  IBestandsvwt::physischerBestand   requires EinfacheDatenverarbeitung
  IBestandsvwt::reserviere          requires KomplexeDatenverarbeitung
  IBestandsvwt::buche               requires KomplexeDatenverarbeitung
  IBestandsvwt::loescheReservierung requires EinfacheDatenverarbeitung
```

Abb. 4.39: Spezifikation parametrisierter Komponentenverträge in UNSCOM/T und UNSCOM/G.

4.5.2.2 Zusicherungen

Auch die Spezifikation der Vor- und Nachbedingungen sowie der Invarianten erfolgt in UNSCOM/T und UNSCOM/G unter Rückgriff auf eine textbasierte Notation. Allerdings wird hierzu die standardisierte OCL [OMG 2003a] verwendet, mit der sich prädikatenlogische Aussagen formulieren lassen. Die Vorgehensweise zur Definition von Zusicherungen entspricht im Wesentlichen der Definition von parametrisierten Komponentenverträgen (vgl. Abb. 4.40). So ist für jede Zusicherung zunächst das *Bezugsobjekt* festzulegen [OMG 2003a:7], wobei sich Invarianten nach den Vorgaben des Metaschemas stets auf Schnittstellen und Vor- bzw. Nachbedingungen stets auf einzelne Methoden beziehen müssen [OMG 2003a:8f.]. Im Anschluss daran ist ein optionaler Name und eine prädikatenlogische Aussage zu spezifizieren, die möglichst um eine natürlichsprachliche Beschreibung zu ergänzen ist. Durch Voranstellen des Schlüsselworts pre, post bzw. inv ist zwischen Vor-, Nachbedingungen und Invarianten zu unterscheiden [OMG 2003a:8]. Zusicherungen, die sich auf das gleiche Objekt beziehen, können nach der Festlegung des Bezugsobjekts zudem direkt untereinander aufgeführt und so zu einer Menge zusammengefasst werden.

```
context IBestandsvwt::buche(b: Bestandsveraenderung, reservierung: ID): Bestand
  pre : b.menge > 0
  -- Der Betrag der Bestandsveraenderung muss groesser als Null sein
  pre : if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            b.menge <= IBestandsvwt::physischerBestand(b.konto)
          else
            true
        endif
  -- Bei einem Warenausgang darf der Betrag der Bestandsveraenderung nicht
     groesser als der physische Bestand des Artikels sein
  post: result = if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
                   then
                     IBestandsvwt::physischerBestand@pre(b.konto) - b.menge
                   else
                     IBestandsvwt::physischerBestand@pre(b.konto) + b.menge
                 endif
  -- Nach der Buchung der Bestandsveraenderung wurde der physische Bestand des
     Artikels um den genannten Betrag vermindert bzw. erhoeht
  post: result >= 0
  -- Nach der Buchung der Bestandsveraenderung ist der physische Bestand nicht
     kleiner als Null
```

Abb. 4.40: Spezifikation der Zusicherungen in UNSCOM/T und UNSCOM/G.

Werden bei der Spezifikation von Zusicherungen für eine Schnittstelle und ihre Methoden zusätzliche Typdefinitionen im Rahmen eines Datenmodells benötigt, lassen sich diese in UNSCOM/T als Elemente eines speziellen Moduls vereinbaren. Das Modul ist mit Hilfe des Schlüsselworts module als Bestandteil des Moduls zu definieren, das auch die entsprechende Schnittstelle enthält. Der Name des zu definierenden Moduls ist aus dem Namen der Schnittstelle abzuleiten, indem diesem die Bezeichnung „__datamodel“ angehängt wird. Als Bestandteile des Datenmodells dürfen außerdem lediglich einfache und komplexe Typen definiert werden. Die Definition von Konstanten, Ausnahmen oder Schnittstellen ist nicht zulässig. In UNSCOM/G ist das Datenmodell als UML Paket darzustellen. Das Paket ist mit einem Namen zu bezeichnen, der wie oben beschrieben aus dem der Schnittstelle abzuleiten ist. Die Datentypen des Datenmodells sind als UML Klassenkonstrukte darzustellen, wobei als Stereotyp «model type» anstelle von «data type» zu verwenden ist. Hierdurch wird die Zugehörigkeit der Datentypen zum Datenmodell deutlich gemacht.

4.5.2.3 Interaktionsprotokolle

Interaktionsprotokolle sind in UNSCOM/T schließlich unter Rückgriff auf die *Protocol Definition Language* (PDL) darzustellen, die zusammen mit dem UNSCOM Spezifikationsrahmen entwickelt wurde. Für jedes Interaktionsprotokoll (protocol) ist optional ein Name sowie ein Bezugsobjekt festzulegen. Das Bezugsobjekt darf dabei entweder eine einzelne Angebotsschnittstelle (Angebotsprotokoll) oder eine Liste von Angebotsschnittstellen (Verbindungsprotokoll) sein. Bei der Spezifikation von Dienstbedarfsprotokollen entfällt das Bezugsobjekt, da diese bei der Spezifikation des parametrisierten Komponentenvertrags in Bezug zu einzelnen Methoden gesetzt werden. Mit Hilfe des Schlüsselworts states

ist anschließend eine Menge von *Zuständen* zu definieren und mit einer Menge von *Zustandsübergängen* (transitions) zu vervollständigen (vgl. Abb. 4.41). Bei der Definition eines Zustands ist jeweils ein eindeutiger Name zu vergeben und mit Hilfe der optionalen Schlüsselwörter startstate und finalstate festzulegen, ob es sich bei dem Zustand ggf. um einen Start- bzw. Endzustand handelt.

```
protocol for IDatenbank {
  states {
    Start startstate;
    Datenbankverbindung;
    Transaktionsverarbeitung;
    Ende finalstate;
  };
  transitions {
    Start->Datenbankverbindung with oeffneVerbindung;
    Datenbankverbindung->Datenbankverbindung with verarbeiteSQL;
    Datenbankverbindung->Transaktionsverarbeitung with beginneTransaktion;
    Transaktionsverarbeitung->Transaktionsverarbeitung with verarbeiteSQL;
    Transaktionsverarbeitung->Transaktionsverarbeitung with rollback;
    Transaktionsverarbeitung->Datenbankverbindung with beendeTransaktion;
    Datenbankverbindung->Ende with schliesseVerbindung;
  };
};

protocol EinfacheDatenverarbeitung {
  states {
    Start startstate;
    Datenbankverbindung;
    Ende finalstate;
  };
  transitions {
    Start->Datenbankverbindung with IDatenbank::oeffneVerbindung;
    Datenbankverbindung->Datenbankverbindung with IDatenbank::verarbeiteSQL;
    Datenbankverbindung->Ende with IDatenbank::schliesseVerbindung;
  };
};

protocol KomplexeDatenverarbeitung {
  states {
    Start startstate;
    Datenbankverbindung;
    Transaktionsverarbeitung;
    Ende finalstate;
  };
  transitions {
    Start->Datenbankverbindung with IDatenbank::oeffneVerbindung;
    Datenbankverbindung->Transaktionsverarbeitung with IDatenbank::beginneTransaktion;
    Transaktionsverarbeitung->Transaktionsverarbeitung with IDatenbank::verarbeiteSQL;
    Transaktionsverarbeitung->Transaktionsverarbeitung with IDatenbank::rollback;
    Transaktionsverarbeitung->Datenbankverbindung with IDatenbank::beendeTransaktion;
    Datenbankverbindung->Ende with IDatenbank::schliesseVerbindung;
  };
};
```

Abb. 4.41: Spezifikation der Angebots- und Dienstbedarfsprotokolle in UNSCOM/T.

Bei der Vereinbarung von Zustandsübergängen werden die definierten Zustände verwendet, um für jeden Übergang einen Ausgangs- und einen Zielpunkt anzugeben. Darüber hinaus ist ggf. hinter dem Schlüsselwort with die Methode anzugeben, deren Benutzung den Zustandsübergang auslöst (vgl. Abb. 4.41). Standardmäßig wird dabei davon ausgegangen,

dass die Ausführung der Methode den Zustandsübergang verursacht. Erfolgt der Übergang dagegen beim Aufruf der Methode oder mit deren Abschluss, ist dies durch Anhängen der Schlüsselwörter .call bzw. .return anzuzeigen. Für die Zustandsübergange von Dienstbedarfsprotokollen lässt sich schließlich eine Reihe optionaler Werte spezifizieren, die die *Wahrscheinlichkeit* angeben, mit der der Übergang während einer Interaktion eintritt.

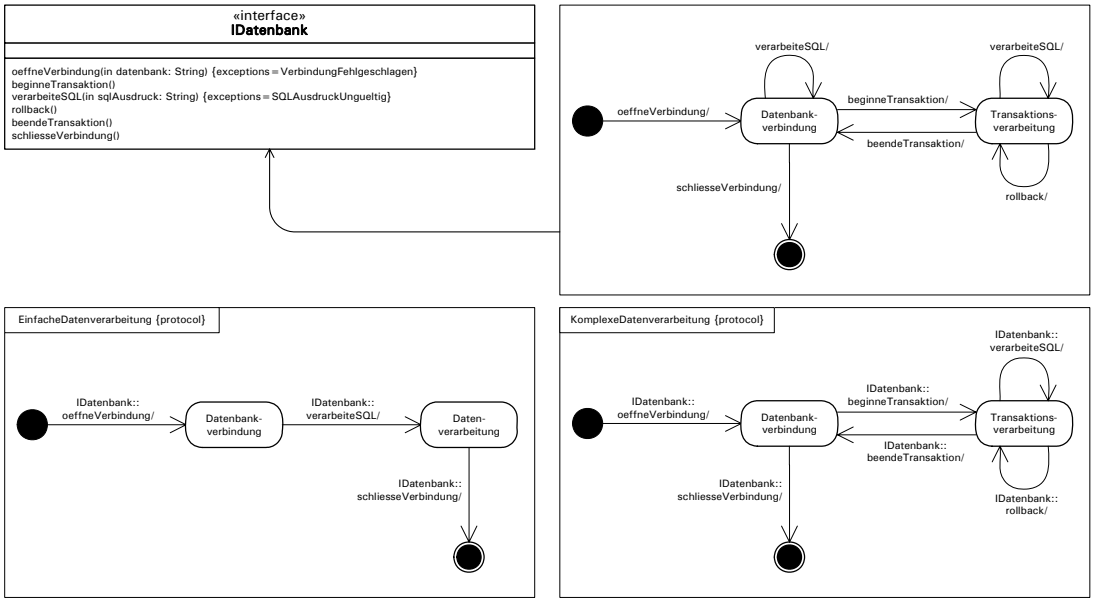


Abb. 4.42: Spezifikation der Angebots- und Dienstbedarfsprotokolle in UNSCOM/G.

Im UNSCOM/G Format sind Interaktionsprotokolle als UML Protokollzustandsautomaten darzustellen [OMG 2003b:455, 464]. Jeder Protokollzustandsautomat ist durch ein Rechteck graphisch darzustellen, das die Zustände und Übergänge enthält und in seiner linken oberen Ecke ein optionales Namensfeld besitzt [OMG 2003b:466]. Nach den Vorgaben der UML 2.0 ist der bei Bedarf zu vergebende Protokollname bei der Spezifikation mit der Eigenschaft {protocol} zu ergänzen, um zu verdeutlichen, dass es sich um einen Protokollzustandsautomaten handelt [OMG 2003b:466]. Ferner ist das Bezugsobjekt bei Bedarf durch eine UML Assoziation festlegen (vgl. Abb. 4.42). Die Zustände sind durch abgerundete Rechtecke zu kennzeichnen, wobei der Start- und die Endzustände ggf. durch spezielle graphische Symbole hervorgehoben werden [OMG 2003b:462f., 471, 482f.]. Die Übergänge werden durch Pfeile dargestellt und ggf. mit der Methode benannt, die den Übergang auslöst [OMG 2003b:469]. Standardmäßig ist dabei der Methodenname anzugeben und mit einem „/“ zu ergänzen (vgl. Abb. 4.42). Der Name ist bei Bedarf mit den Schlüsselwörtern .call bzw. .return zu ergänzen. Die *Wahrscheinlichkeitswerte* können den Übergängen eines Dienstbedarfsprotokolls schließlich als Kommentare hinzugefügt werden. Sie sind mit dem Stereotyp «probability» einzuleiten und entsprechen der Grammatik der Protocol Definition Language.

Abb. 4.43: Formatelemente zur Darstellung der Verträge und Protokolle in UNSCOM/T und UNSCOM/G.

Abb. 4.43 fasst die Formatelemente zur Repräsentation der Komponenten- und Dienstverträge sowie der Interaktionsprotokolle noch einmal zusammen. Die einander entsprechenden Elemente der UNSCOM/T und UNSCOM/G Formate sind dabei jeweils gegenübergestellt und in Bezug zu den korrespondieren Auszügen des UNSCOM Metaschemas gesetzt.

4.5.3 Geltung

Die mit den Green Pages zu beschreibenden architekturspezifischen Merkmale der Komponentenaußensicht werden zum größten Teil während des Systementwurfs bzw. Domänenentwurfs festgelegt, in dessen Rahmen die Software-Architektur des zugehörigen Systems bestimmt wird. Dieses kann bspw. ein Anwendungssystem oder ein Software-System für einen Ausschnitt aus dem Anwendungsbereich sein, das die zu entwickelnde Komponente als Strukturierungseinheit beinhaltet. Die architekturspezifischen Merkmale der Komponentenaußensicht werden dabei zusammen mit der Struktur- und Kontrollsicht des jeweiligen Systems definiert. Bei der Festlegung dieser beiden Architektursichten kann ggf. auf Schnittstellen- und Architekturstandards zurückgegriffen werden, die Signaturen bzw. Interaktionsabläufe für einzelne Funktionalitätsbereiche vorgeben.

In Abhängigkeit vom Detaillierungsgrad, mit dem der System- bzw. Domänenentwurf durchgeführt wird, liegt die Beschreibung der Komponentenarchitektur auf unterschiedliche Weise vor. Im einfachsten Fall (*Detailstufe 1: Signaturen*) beinhalten die Green Pages lediglich eine Menge von Signaturdefinitionen, durch die die Angebots- und Nachfrageschnittstellen sowie die ausgesandten und empfangbaren Ereignisse festgelegt werden. Zusätzlich kann ein parametrisierter Komponentenvertrag spezifiziert werden, der für jede Methode der Angebotsschnittstellen auflistet, welche Methoden während ihrer Ausführung von anderen Komponenten über die Nachfrageschnittstellen in Anspruch genommen werden. Wird bei der Festlegung der Systemarchitektur auch der Ablauf betrachtet, lassen sich im Rahmen der Green Pages zudem die Protokolle (*Detailstufe 2: Interaktionsprotokolle*) angeben, die den Interaktionsverlauf festlegen. Für die Zustandsübergänge der Dienstbedarfsprotokolle sind dabei auch Übergangswahrscheinlichkeiten zu schätzen, da diese während des Systementwurfs zur Qualitätsvorhersage genutzt werden können [REUSSNER, et al. 2003]. Zugleich ersetzen die Dienstbedarfsprotokolle die Auflistung der nachgefragten Methoden bei der Spezifikation eines parametrisierten Komponentenvertrags. Die vollständige Beschreibung der architekturspezifischen Merkmale (*Detailstufe 3: Dienstverträge*) umfasst schließlich auch die Vor- und Nachbedingungen, mit denen die systemtechnischen Bedingungen für die Inanspruchnahme der einzelnen Methoden formal-präzise beschrieben werden.

Nach Abschluss des System- bzw. Domänenentwurfs beschreiben die architekturspezifischen Merkmale der Komponentenaußensicht zunächst einen Soll-Zustand, der bei der Realisierung der Komponenten einzuhalten ist. Durch die Festlegung einer gemeinsamen Systemarchitektur bleibt dabei gewährleistet, dass die einzelnen Komponenten auch dann zusammenarbeiten können, wenn sie unabhängig voneinander (also bspw. von verschiedenen Teams) realisiert wurden. Nach Fertigstellung der Implementierung und der dazugehörigen Modultests beschreiben die Green Pages dann den jeweils erreichten Ist-Zustand. Durch Tests lassen sich schließlich während der Stabilisierung auch die Wahrscheinlichkeiten für die einzelnen Zustandsübergänge der Interaktionsprotokolle exakter ermitteln, die nur für die *Dienstbedarfsprotokolle* anzugeben sind. Der Ist-Zustand kann bei der Auswahl von Komponenten im Rahmen späterer Entwicklungsprojekte ausgewertet werden, um die systemtechnische Eignung einer Komponente zu überprüfen oder die Qualität des Gesamtsystems vorherzusagen [D'SOUZA und WILLS 1999:583]. Durch die explizite Dokumentation der systemtechnischen Komponentenbeschaffenheit wird zudem die Herausbildung von Architekturstandards begünstigt – etwa wenn andere Komponenten Teile der architekturspezifischen Merkmale übernehmen.

Anzumerken bleibt, dass sich die Beschreibung der (fachlichen) Komponentenfunktionalität grundsätzlich von der der (systemtechnischen) Komponentenarchitektur unterscheidet. Zwar beschreiben sowohl Blue Pages als auch Green Pages jeweils die in Abschnitt 3.1.6 dargestellten Elemente des konzeptionellen Komponentenmodells. Häufig ähnelt die Beschreibung der Komponentenarchitektur auf den ersten Blick auch der fachlichen Beschreibung. So lassen sich die definierten Typen und Methoden der Schnittstellen idealerweise aus den zuvor beschriebenen Informationsobjekten und Funktionen ableiten. Dennoch werden bei der Beschreibung verschiedene Sichtweisen eingenommen. Deshalb sind die systemtechnischen Merkmale im Allgemeinen nicht identisch mit den fachlichen Merkmalen, sondern stellen *Weiterentwicklungen* dar [D'SOUZA und WILLS 1999:242]. Systemtechnische unterscheiden sich von fachlichen Komponentenmerkmalen zum einen durch die Anwendung *programmiertechnischer Konstrukte* (wie z.B. Typdefinitionen, Parameterlisten oder Schnittstellendefinitionen). Zum anderen weicht die *Namensgebung* ab, da selbst bei der Verwendung sprechender (sprich fachlicher) Namen meist zusätzliche technische Namenskonventionen zum Einsatz kommen. Einer solchen Konvention entstammen bspw. die Regeln, den Schnittstellennamen stets ein „I“ voranzustellen oder Parameternamen klein zu schreiben. Schließlich unterscheiden sich systemtechnische von fachlichen Merkmalen üblicherweise auch in Bezug auf den *Inhalt*. So gehen systemtechnische Merkmale z.B. auf Nebenläufigkeits- und Berechenbarkeitsaspekte ein, während fachliche Aspekte vielfach nicht mehr explizit spezifiziert werden (können). Durch eine technische Vorbedingung lässt sich etwa beschreiben, dass bei der Kommissionierung ein

wechselseitiger Ausschluss von Dienstnachfragern erfolgt. Hingegen wird nur als Bestandteil der fachlichen Handlungssituation festgelegt, dass ein chaotisches Kommissionierverfahren zum Einsatz kommt.

Die Weiterentwicklungen der fachlichen Merkmale, die mit dem Wechsel der Perspektive von einer fachlichen Sicht auf eine logische Sicht vorgenommen werden, lassen sich mit dem UNSCOM Spezifikationsrahmen explizit dokumentieren. Durch eine solche Dokumentation lässt sich einerseits prüfen, ob sämtliche der für die Funktionalität relevanten Merkmale auch in die weitere Komponentenentwicklung eingegangen sind. Andererseits lässt sich durch einen Umkehrschluss vergleichsweise einfach ermitteln, welche fachliche Funktionalität die einzelnen systemtechnischen Merkmale einer Komponente realisieren. Damit lässt sich unter anderem die Durchführung von Kompatibilitätstests zwischen den Angebots- und Nachfrageschnittstellen zweier Komponenten mit dem Ziel unterstützen, Methoden mit gleicher Funktionalität und abweichender Signatur zu identifizieren.

Begriff	Art	Architekturbestandteil	Art
Lagerplatz	Infoobjekt	Lagerorganisation::Lagerplatz	Typ
Lagerbereich	Infoobjekt	Lagerorganisation::Lagerbereich	Typ
Bestandsveränderung	Infoobjekt	Lagerorganisation::IBestandsvwt::Bestandsveraenderung	Typ
Bestandsveränderung buchen	Funktion	Lagerorganisation::IBestandsvwt::buche	Methode

Abb. 4.44: Repräsentation der Realisierungsbeziehungen in UNSCOM/T und UNSCOM/G.

Wie bei Catalysis sind die durchgeführten Weiterentwicklungen durch *Realisierungsbeziehungen* zwischen den betroffenen Merkmalen zu spezifizieren [D'SOUZA und WILLS 1999:34, 511]. Die einzelnen Realisierungsbeziehungen sind in UNSCOM/T und UNSCOM/G in einer Tabelle mit der Bezeichnung Realisierungen zusammenzufassen, wobei für ein fachliches Merkmal (linke Seite) anzugeben ist, von welchen systemtechnischen Merkmalen (rechte Seite) es realisiert wird. Abb. 4.44 zeigt beispielhaft einige Realisierungsbeziehungen, die zwischen den zuvor spezifizierten Merkmalen der Komponente Lagermanagement bestehen. Grundsätzlich lassen sich folgende Zusammenhänge zwischen den verschiedenen Merkmalsarten identifizieren [ACKERMANN, et al. 2002:7]: Informationsobjekte dienen als Grundlage für Typdefinitionen, aus Funktionen lassen sich Methoden ableiten, Prozesse werden durch Interaktionsprotokolle unterstützt und fachliche Handlungssituationen gehen ggf. in die Beschreibung von Dienstverträgen ein. Die Spezifikation der Zusammenhänge zwischen fachlichen Funktionen und systemtechnischen Methoden dient dabei zugleich einem weiteren Zweck: durch sie lässt sich die Gruppierung von Methoden zu Schnittstellen auf die fachlichen Funktionen übertragen, so dass diese im

Rahmen der Blue Pages nicht noch einmal thematisch zusammengefasst werden müssen (vgl. Abschnitt 4.4.3).

4.6 Physische Qualität

Außer durch ihre Funktionalität und ihre architekturspezifischen Merkmale werden Komponenten vor allem durch ihre jeweiligen qualitativen Eigenschaften charakterisiert. Diese Eigenschaften geben an, mit welcher *Güte* eine Komponente in der Lage ist, ihre Funktionalität zu erbringen. Formal lassen sich die qualitativen Eigenschaften einer Komponente unter Rückgriff auf ein *Qualitätsmodell* beschreiben, das verschiedene Qualitätsmerkmale unterscheidet. Der UNSCOM Spezifikationsrahmen unterstützt die Beschreibung der qualitativen Eigenschaften von Komponenten und baut dabei auf dem Qualitätsmodell des *ISO 9126* Standards [ISO/IEC 2001; ISO/IEC 2003] auf. Mit diesem Modell lassen sich die Verwendbarkeit, Wartbarkeit, Portabilität, Funktionalität, Zuverlässigkeit und Effizienz von Komponenten in einer standardisierten Weise beschreiben. Diese Beschreibung deckt die in Abschnitt 3.4.2 zur Festlegung der physischen Qualität identifizierten Ebenen eines Komponentenvertrags ab. Die Vertragsebenen werden vom UNSCOM Spezifikationsrahmen dabei zu einem eigenen thematischen Bereich zusammengefasst und als *Grey Pages* bezeichnet. Die gewählte Bezeichnung weist darauf hin, dass viele der qualitativen Eigenschaften von der Realisierung der Komponente abhängen. Jede Beschreibung der Qualitätsmerkmale gestattet damit zumindest einen abstrakten Einblick in deren Implementierung, weswegen in der Literatur bisweilen auch von einer *Greybox Sicht* (als Mittelweg zwischen den beiden idealtypischen Blackbox- und Whitebox-Sichten) gesprochen wird [GRIFFEL 1998:419; SZYPERSKI, et al. 2002:40f.].

4.6.1 Inhalt

Um die Informationsverarbeitung des Anwendungsbereichs in angemessener Form zu unterstützen, müssen Anwendungssysteme neben funktionalen Anforderungen in zunehmendem Maße auch anspruchsvollen *qualitativen Anforderungen* genügen, die bei der Entwicklung in gleicher Weise zu berücksichtigen sind. In den Pflichtenheften, die zu Beginn von Anwendungsentwicklungsprojekten erstellt werden, finden sich daher nicht mehr nur Vereinbarungen bezüglich der zu realisierenden Funktionalität, sondern außerdem konkrete qualitative Vorgaben [CHEESMAN und DANIELS 2001:82]. Diese werden meist unter Verwendung von *Kennzahlen* formuliert, die jeweils für ein bestimmtes *Qualitätsmerkmal* charakteristisch sind. So lässt sich für das in der Fallstudie zu realisierende Verkaufssys-

tem bspw. fordern, dass die Bearbeitung eines Bestellvorgangs nicht länger als 3 Sekunden dauern darf (Antwortzeit) und das System außerdem in der Lage sein muss, wenigstens 10 Bestellvorgänge pro Sekunde auszuführen (Durchsatz). Die verschiedenen Qualitätskategorien und Qualitätsmerkmale werden dabei durch das jeweils zugrunde gelegte *Qualitätsmodell* bestimmt [ISO/IEC 2001:6].

Mit der Strukturierung des Anwendungssystems in kleinere Einheiten werden auch die systemspezifischen Kennzahlen zu *komponentenspezifischen Qualitätskennzahlen* zerlegt, die sich mit dem UNSCOM Spezifikationsrahmen in einer einheitlichen Weise beschreiben lassen. Als Qualitätsmodell wird im UNSCOM Metaschema dabei das Modell des *ISO 9126* Standards [ISO/IEC 2001:7] zugrunde gelegt, das eine Vielzahl von Qualitätskategorien und extern messbaren Merkmalen unterscheidet (vgl. Abb. 4.45 links). Darunter befinden sich sowohl Qualitätskategorien, die die Verwendung einer Komponente als Ganzes beschreiben, als auch Kategorien, die die Güte ihrer einzelnen angebotenen bzw. nachgefragten Dienste beschreiben. Zu den Qualitätskategorien, die die Verwendung der Komponente als Ganzes charakterisieren, gehören die Verwendbarkeit, die Wartbarkeit sowie die Portabilität. Die *Verwendbarkeit* gibt Auskunft über den Aufwand, der von einem Dritten zu betreiben ist, um eine Komponente zu verstehen und in seinem Anwendungsentwicklungsprojekt zu nutzen [ISO/IEC 2001:9]. Die *Wartbarkeit* erlaubt Rückschlüsse auf den Aufwand, der mit zukünftigen Änderungen an der Komponente verbunden ist [ISO/IEC 2001:10]. Die *Portabilität* gibt schließlich Aufschluss über den Aufwand, der bei der Übertragung einer Komponente in eine spezifische Anwendungsumgebung entsteht [ISO/IEC 2001:11].

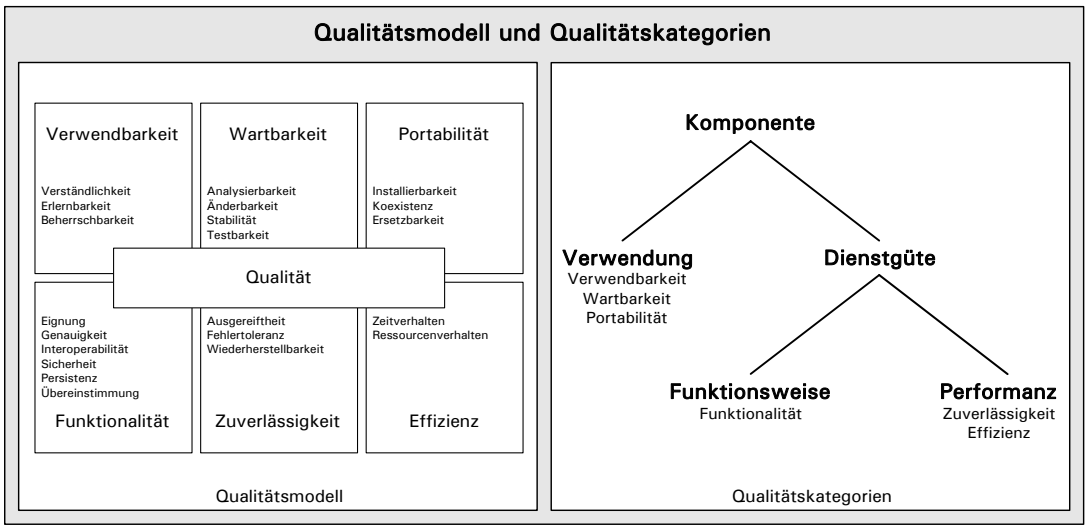


Abb. 4.45: Standardisiertes Qualitätsmodell und Einteilung in Qualitätskategorien nach ihrem jeweiligen Bezugsobjekt (in Anlehnung an [ISO/IEC 2001:7]).

Zu den Qualitätskategorien, mit denen sich die Güte der einzelnen Dienste beschreiben lässt, gehören demgegenüber die Funktionalität, die Zuverlässigkeit sowie die Effizienz. Die *Funktionalität* beschreibt Eigenschaften von Diensten, die bei der Ausführung ergänzend zur jeweils erbrachten fachlichen Funktionalität auftreten [ISO/IEC 2001:7f.]. Hierzu gehören Merkmale wie die Interoperabilität, die Persistenz oder die Sicherheit, die bei der Dienstausführung garantiert wird. Die *Zuverlässigkeit* gibt Aufschluss über die Fähigkeit einer Komponente, einen Dienst ggf. auch beim Auftreten eines Fehlers dauerhaft (also ohne Ausfall) und mit einer bestimmten Güte zu erbringen [ISO/IEC 2001:8f.]. Die *Effizienz* beschreibt schließlich die zeitbezogenen Merkmale eines Dienstes sowie die bei der Ausführung jeweils in Anspruch genommenen Ressourcen [ISO/IEC 2001:10].

Die Entwicklung standardisierter Qualitätsmodelle basiert typischerweise auf dem sog. *Factor Criteria Metrics* (FCM) Ansatz, wonach zur Konkretisierung des Qualitätsbegriffs zunächst verschiedene Qualitätskategorien zu identifizieren und im Anschluss daran die jeweiligen Qualitätsmerkmale sowie Metriken zu deren Quantifizierung festzulegen sind [BALZERT 1998:257-262]. Bei diesem anwendungsunabhängigen Vorgehen sind zwei gegensätzliche Ziele zu verfolgen: einerseits ist auf eine umfassende Zerlegung in Kategorien und Merkmale zu achten, die möglichst keine Qualitätseigenschaften unberücksichtigt lässt. Andererseits ist schon aus Gründen der Handhabbarkeit die Schaffung eines kompakten Qualitätsmodells anzustreben. Somit ist selbst bei der Verwendung eines standardisierten Qualitätsmodells nicht auszuschließen, dass einzelne Qualitätsmerkmale während der Spezifikation von qualitativen Komponenteneigenschaften neu zu definieren sein werden. Dies gilt insbesondere für Merkmale, die nur für bestimmte Anwendungsbereiche (z.B. eingebettete Systeme) von Bedeutung sind. Das UNSCOM Metaschema beinhaltet deshalb zunächst allgemeine Regeln zur Festlegung von Qualitätsmerkmalen und zur Beschreibung von Merkmalsausprägungen (Kennzahlen). Diese Regeln werden durch einen Katalog von standardisierten Qualitätsmerkmalen, die der ISO 9126 entnommen wurden, ergänzt.

Die Spezifikation von qualitativen Komponenteneigenschaften besteht also aus zwei Teilen. Zunächst werden z.B. mithilfe der *Goal Question Metric* (GQM) Methode [VAN SOLINGEN und BERGHOUT 1999] die relevanten *Qualitätskategorien* und *-merkmale* als *Qualitätstypen* bestimmt, wobei nach Möglichkeit die Vorgaben des UNSCOM Katalogs zu nutzen sind. Anschließend werden die Merkmalsausprägungen spezifiziert, die als *Kennzahlen* die eigentlichen *Qualitätseigenschaften* der Komponente darstellen. Die festgelegten Qualitätskategorien werden im UNSCOM Metaschema durch das Element `Kategorietyp` dargestellt (vgl. Abb. 4.46). Für jeden Kategorietyp sind gemäß den Vorgaben des Metaschemas ein *Name* (wie z.B. Zuverlässigkeit, Effizienz etc.) und eine *Liste* mit charakteristischen Merkmalen anzugeben [FROLUND und KOISTINEN 1998:10, 12]. Die charakteristischen

Merkmale einer Kategorie werden im Metaschema durch das Element Merkmalstyp dargestellt (vgl. Abb. 4.46). Jeder Merkmalstyp ist eindeutig zu benennen und hinsichtlich seines *Wertebereichs* zu konkretisieren. Darüber hinaus lassen sich optional eine *Maßeinheit* sowie eine partielle *Ordnung* zwischen den Elementen des Wertebereichs in Verbindung mit einer *Präferenz* angeben [FROLUND und KOISTINEN 1998:10-12].

Der Wertebereich eines Merkmalstyps kann entweder als die Menge der *reellen Zahlen* oder eine *Aufzählung von beliebigen Einzelwerten* festgelegt werden. Dabei lässt sich ggf. auch angeben, dass eine Merkmalsausprägung aus einer *Menge von aufgezählten Werten* besteht [FROLUND und KOISTINEN 1998:12f.]. So besitzen die eingangs bereits erwähnten Merkmale Antwortzeit und Durchsatz bspw. die Menge der reellen Zahlen als Wertebereich. Das Merkmal Fehlerbehandlung, das die Fähigkeit eines Dienstes zur Erkennung und Behandlung von auftretenden Fehlern charakterisiert, nimmt hingegen einen Wert aus der Menge Fehlerbehandlung = {ohne, erkennung, warnung, behandlung} an. Gleichzeitig lassen sich die Maßeinheiten für das Merkmal Antwortzeit als Sekunden und für das Merkmal Durchsatz als Aufrufe/Sekunde festlegen.

Die Festlegung des Wertebereichs impliziert ein *Skalenniveau* für die einzelnen Messwerte. Das Skalenniveau gibt an, welche Auswertungen bzw. Interpretationen für die einzelnen Qualitätsmerkmale inhaltlich sinnvoll sind. Im Allgemeinen wird dabei zwischen Nominalskalen, Ordinalskalen, Intervallskalen, Rationalskalen und Absolutskalen unterschieden [FROLUND und KOISTINEN 1998:58]. *Nominalskalen* lassen lediglich Aussagen über die Gleichheit bzw. Ungleichheit von Messwerten zu. *Ordinalskalen* erlauben zusätzlich Aussagen über die Rangordnung von Messwerten, wobei über das Ausmaß der Größenunterschiede jedoch nichts ausgesagt werden kann. Bei *Intervallskalen* ist der Abstand zwischen den Einheiten konstant gleich, so dass bspw. ein Mittelwert gebildet werden kann. Da jedoch ein Nullpunkt fehlt (oder nur beliebig festgelegt ist), lassen sich keine Quotienten aus den gemessenen Größen bilden. Ist ein absoluter Nullpunkt vorhanden, der die Bildung von Quotienten erlaubt, liegt stattdessen eine *Rationalskala* vor. *Absolutskalen* erlauben schließlich uneingeschränkte mathematische Auswertungen und eignen sich zur Darstellung dimensionsloser reeller Zahlen.

Wird bei der Definition eines Merkmalstyps die Menge der reellen Zahlen als Wertebereich angegeben, sind die einzelnen Werte automatisch *vollständig geordnet* und zumindest intervallskaliert. Bei einer Aufzählung beliebiger Einzelwerte ist eine Ordnung dagegen explizit als *partielle* (transitive) *Ordnung* zwischen den einzelnen Elementen festzulegen, wodurch sich eine Ordinalskala ergibt. Wird keine Ordnung festgelegt, gilt die Menge der Werte als ungeordnet und somit lediglich als nominalskaliert. Für die ordinalskalierten

Werte des Merkmals Fehlerbehandlung lässt sich bspw. folgende Ordnung explizit angeben: {ohne < erkennung, erkennung < warnung, warnung < behandlung}. Die zwischen den einzelnen Werten definierte Ordnung wird dabei wie folgt auf Teilmengen dieser Werte ausgedehnt: eine Teilmenge A ist kleiner als eine Teilmenge B, wenn jedes Element, das ausschließlich in A vorkommt, kleiner ist als ein Element, das ausschließlich in B vorkommt [FROLUND und KOISTINEN 1998:13]. Ist zumindest eine partielle Ordnung auf den Elementen einer Menge definiert, muss schließlich noch eine *Präferenz* angegeben werden. Durch sie wird festgelegt, ob größere (aufsteigend) oder kleinere (absteigend) Werte als qualitativ höherwertig angesehen werden. So gilt für die Antwortzeit bspw., dass kleinere Werte besser sind, während beim Durchsatz sowie bei der Fehlerbehandlung größere Werte als qualitativ höherwertig angesehen werden.

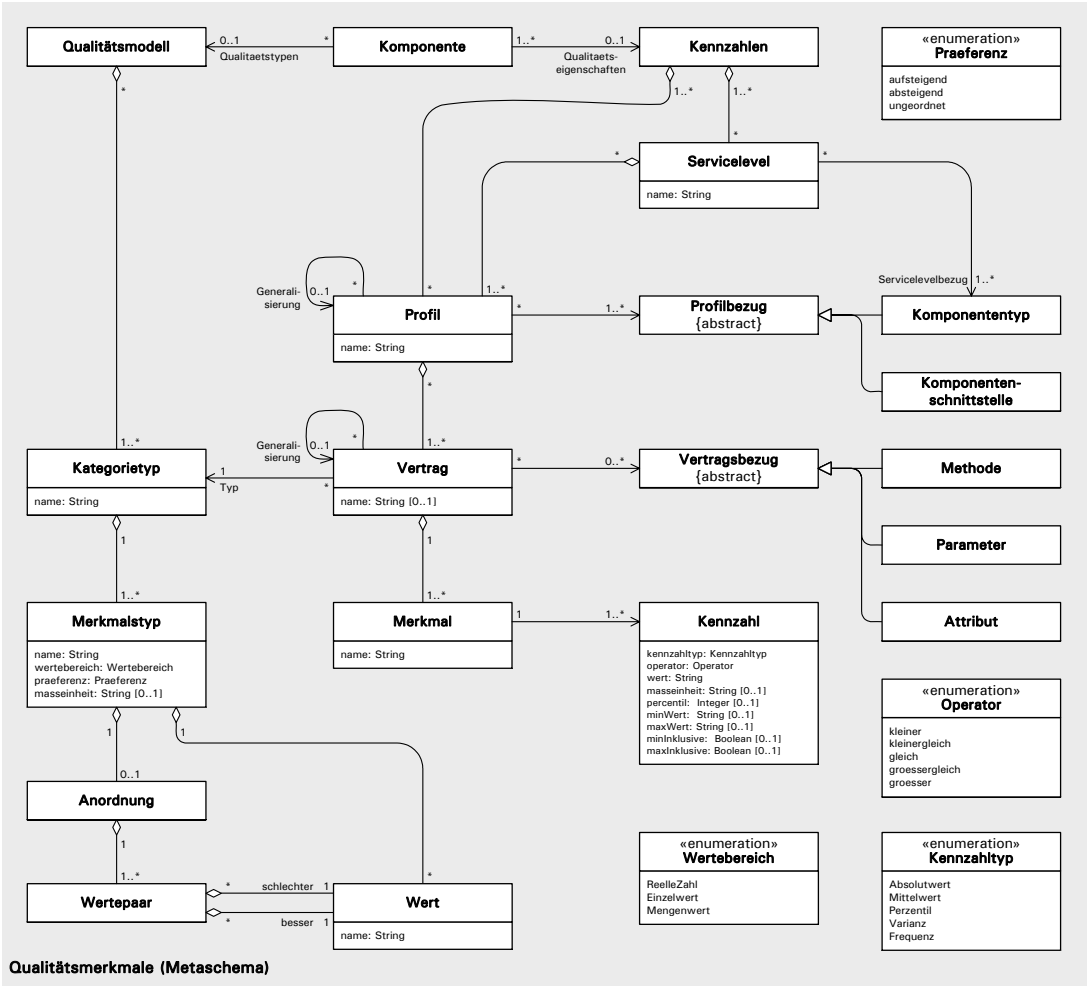


Abb. 4.46: Metaschema mit inhaltlichen Vorgaben zur Spezifikation von qualitativen Eigenschaften.

Unter Rückgriff auf die definierten Qualitätskategorien und -merkmale lassen sich anschließend die Qualitätseigenschaften der Komponente spezifizieren. Sie werden im UNS-

COM Metaschema zu *Qualitätsprofilen* zusammengefasst und durch das Schemaelement Profil dargestellt (vgl. Abb. 4.46). Für jedes Qualitätsprofil ist ein eindeutiger *Name* anzugeben und mindestens ein *Bezugsobjekt* festzulegen. Das Bezugsobjekt (Profilbezug) gibt an, ob sich die in dem Profil enthaltenen Qualitätseigenschaften auf eine *Komponente* als Ganzes oder eine ihrer *Angebots- bzw. Nachfrageschnittstellen* beziehen [FROLUND und KOISTINEN 1998:10]. Entsprechend können als Bezugsobjekte im Metaschema der jeweilige Komponententyp oder die Komponentenschnittstellen eingesetzt werden. Auf diese Weise lassen sich z.B. Qualitätseigenschaften, die sich auf die Komponentenverwendung beziehen, getrennt von jenen beschreiben, die sich auf die Güte von Diensten beziehen. In jedem Qualitätsprofil dürfen dabei Kennzahlen stehen, die sich auf verschiedene Kategorien (wie z.B. die Zuverlässigkeit oder die Effizienz) beziehen. Gleichzeitig kann bei Aussagen über Komponentenschnittstellen auf verschiedene Schnittstellenelemente, also bspw. auf einzelne Methoden, Bezug genommen werden. Die verschiedenen Kennzahlen werden deshalb weiter zu sog. *Verträgen* gruppiert, die im Metaschema durch das gleichnamige Element Vertrag dargestellt werden und optional mit einem jeweils eigenen Namen versehen werden können [FROLUND und KOISTINEN 1998:10].

Die in einem Vertrag enthaltenen *Kennzahlen* gehören allesamt zu einer Qualitätskategorie [FROLUND und KOISTINEN 1998:10]. Außerdem beziehen sie sich entweder auf die im Profil festgelegte Komponente bzw. Schnittstelle insgesamt oder auf Schnittstellenelemente, die zusammen mit dem Vertrag festzulegen sind (vgl. Vertragsbezug in Abb. 4.46). Jede Kennzahl stellt eine Ausprägung eines Merkmals dar, das zuvor als Teil der jeweiligen Qualitätskategorie definiert wurde. Die Kennzahlen sind deshalb mit dem Namen des jeweiligen Merkmalstyps zu bezeichnen, um diesen Bezug zu dokumentieren. Für jedes Merkmal darf in einem Vertrag entweder eine *absolute Aussage* oder eine Reihe von *statistischen Aussagen* angegeben werden. Bei Merkmalen mit einem nominalskalierten Wertebereich ist durch eine absolute Aussage stets eine *exakte Übereinstimmung* anzugeben: Verschlüsselung = keine. Bei Merkmalen, die zumindest einen ordinalskalierten Wertebereich mit einer festgelegten Präferenz besitzen, sind hingegen *Schranken* festzulegen [FROLUND und KOISTINEN 1998:14]. Bei aufsteigenden Werten sind dabei jeweils untere und bei absteigenden Werten jeweils obere Schranken anzugeben: Durchsatz > 10 Aufrufe/Sekunde, Antwortzeit < 3 Sekunden.

Durch statistische Aussagen lassen sich vor allem Merkmale mit schwankenden Ausprägungen besser beschreiben. So ergeben sich bei der Messung der Antwortzeit bspw. folgende Ergebnisse: 1, 1, 1, 2, 2, 2, 2, 2, 3, 3. Zur Beschreibung solcher Merkmale sind grundsätzlich Aussagen über den *Mittelwert*, die *Varianz*, die verschiedenen *Perzentilwerte* sowie die *Frequenz*, mit der bestimmte Werte auftreten, erlaubt: Antwortzeit Mittel ≤ 1.9

Sekunden, Antwortzeit Perzentil 80 ≤ 2 Sekunden. Zu beachten ist jedoch, dass nicht alle statistischen Aussagen zur Beschreibung der Ausprägungen von beliebigen Merkmalen sinnvoll einsetzbar sind. Aussagen über den Mittelwert dürfen nur für Merkmale getroffen werden, deren Wertebereich aus den reellen Zahlen besteht. Zudem gelten dieselben Konventionen wie bei absoluten Aussagen, wonach lediglich Schranken für den Mittelwert spezifiziert werden dürfen [FROLUND und KOISTINEN 1998:15]. Einschränkungen ergeben sich auch für Aussagen über die Varianz, für die prinzipiell nur obere Schranken angegeben werden dürfen [FROLUND und KOISTINEN 1998:16]. Darüber hinaus bleibt anzumerken, dass die Güte von statistischen Aussagen wesentlich durch die Form der zugrunde liegenden *Datenerhebung* beeinflusst wird. Die Datenerhebung ist bei der Beschreibung von schwankenden qualitativen Eigenschaften deshalb als wesentlicher Faktor zu berücksichtigen [FROLUND und KOISTINEN 1998:16; ACKERMANN, et al. 2002:13].

Für schwankende Qualitätseigenschaften wie die Effizienz oder die Zuverlässigkeit werden häufig verschiedene Schwankungsbreiten definiert und jeweils zu einem sog. *Service Level* zusammengefasst [PANTRY und GRIFFITHS 1997:12f.; HEINRICH 1999:434f.; KNOBLAUCH und EGARDT 2003:28]. Auch das UNSCOM Metaschema unterstützt die Vereinbarung solcher Service Level, indem es die Zusammenfassung mehrerer Qualitätsprofile unter einer eigenständigen Bezeichnung erlaubt (vgl. Abb. 4.46). Die so zu einem Service Level zusammengefassten Profile werden durch das gleichnamige Schemaelement Servicelevel dargestellt. Sie enthalten die qualitativen Eigenschaften, die mit dem jeweiligen Service Level für die Dienste der Angebotsschnittstellen garantiert und im Gegenzug ggf. von den Diensten der Nachfrageschnittstellen erwartet werden. Mit dem UNSCOM Spezifikationsrahmen lassen sich allerdings keine vollständigen Intervalle, sondern ausschließlich *qualitative Mindestanforderungen* für die einzelnen Qualitätseigenschaften festlegen. Nur das Unterschreiten dieser Mindestanforderungen wird dementsprechend als Verletzung des jeweiligen Service Level interpretiert.

Analog zur Spezialisierung von Schnittstellen erlaubt das UNSCOM Metaschema schließlich auch die *Spezialisierung von Qualitätsprofilen*. Auf diese Weise ist es möglich, im Rahmen von Vererbungsbeziehungen zwischen Schnittstellen auch Qualitätsprofile von den generelleren auf die speziellere Schnittstelle zu übertragen und so deren *Konformität* in Bezug auf die qualitativen Eigenschaften sicherzustellen [FROLUND und KOISTINEN 1998:21]. Bei der Spezialisierung eines Qualitätsprofils dürfen die dort enthaltenen Verträge spezialisiert und um weitere Verträge ergänzt werden. Die *Spezialisierung eines Vertrags* erfolgt entweder durch das *Hinzufügen* von Merkmalsausprägungen, die bislang noch nicht festgelegt waren, oder die *Redefinition* von Merkmalsausprägungen, die bereits festgelegt wurden. Bei der Redefinition dürfen Merkmalsausprägungen ausschließlich durch

qualitativ höherwertige Ausprägungen ersetzt werden [FROLUND und KOISTINEN 1998:22]. Hieraus ergibt sich, dass nominalskalierte Merkmale grundsätzlich nicht redefiniert werden können. Bei Merkmalen, deren Werte zumindest ordinalskaliert sind, wird die Gültigkeit einer Redefinition durch die jeweilige Präferenz festgelegt. So lässt sich die Aussage Antwortzeit < 3 Sekunden bspw. durch Antwortzeit < 2 Sekunden redefinieren, während die Aussage Durchsatz > 10 Aufrufe/Sekunde durch Durchsatz > 20 Aufrufe/Sekunde redefiniert werden darf. Um eine Spezialisierung anzuzeigen, sind die spezialisierten Verträge bzw. Profile im Metaschema jeweils mit ihrer Generalisierung in Bezug zu setzen. Anzumerken bleibt, dass sich bei der Spezialisierung im Rahmen einer Schnittstellenvererbung aufgrund der vorgenommenen funktionalen Erweiterungen häufig *Verschlechterungen* einzelner qualitativer Eigenschaften ergeben. Bei Schnittstellenvererbungen können deshalb nicht automatisch auch alle Qualitätsprofile vererbt werden, obwohl dies im Sinne der Konformität grundsätzlich zu wünschen wäre.

Ergänzend zu den genannten Vorgaben, die sich auf die Spezifikation qualitativer Komponenteneigenschaften beziehen, stellt der UNSCOM Spezifikationsrahmen einen Katalog mit standardisierten Qualitätskategorien und Merkmalen zu Verfügung. Diese Kategorien und Merkmale sind bei der Spezifikation qualitativer Komponenteneigenschaften möglichst zu nutzen, um die Einheitlichkeit und Vergleichbarkeit von Komponentenspezifikationen zu verbessern. Der Katalog baut auf den Vorgaben des *ISO 9126* Standards auf, der in einem zweiten Teil (dem sog. ISO 9126-2 Dokument) eine Reihe von extern messbaren Qualitätsmerkmalen aufführt [ISO/IEC 2003]. Diese lassen sich nutzen, um die *Verwendung von Komponenten*, *die Funktionsweise von Diensten* sowie die *Performanz von Diensten* durch absolute oder statistische Aussagen zu beschreiben. Da sich der ISO Standard jedoch auf die Software-Qualität im Allgemeinen bezieht, sind einige der Qualitätsmerkmale für die Nutzung im Rahmen der komponentenorientierten Anwendungsentwicklung umzudeuten bzw. anzupassen [BERTOA und VALLECILLO 2002:3f.]. Dadurch ergeben sich die nachfolgend aufgeführten Merkmalstypen als Elemente des UNSCOM Merkmalskatalogs.

4.6.1.1 Verwendung

Die Verwendung von Komponenten als Ganzes lässt sich unter Rückgriff auf die Merkmale beschreiben, die als Bestandteile der Qualitätskategorien Verwendbarkeit, Wartbarkeit und Portabilität mit dem ISO 9126 Qualitätsmodell eingeführt werden. Jede dieser Kategorien besteht zunächst aus weiteren Unterkategorien, die schließlich eine Reihe von charakteristischen Merkmalstypen enthalten. Als geeignete Unterkategorien zur Beschreibung der *Verwendbarkeit* werden dabei zunächst die Verständlichkeit, Erlernbarkeit und die Beherrschbarkeit betrachtet [ISO/IEC 2001:9f.]. Die im ISO Qualitätsmodell ebenfalls aufgeführte Attraktivität bezieht sich dagegen ausschließlich auf die Eignung von Software-Pro-

dukten für Endanwender und wurde aus diesem Grund nicht einbezogen. Die Attraktivität (Eignung) von Komponenten ist im Rahmen eines Entwicklungsprojekts vielmehr anhand ihrer jeweiligen Spezifikation zu überprüfen und kann abhängig von den jeweils bestehenden Anforderungen durchaus unterschiedlich ausfallen.

Verständlichkeit		
Methoden		
Inhalt: Anteil der Methoden, die nach Durchsicht der Schnittstellen verstanden werden.		
Messung: Zählen der verstandenen Methoden (m_v) und Vergleich mit der Gesamtanzahl (m_g).		
Formel: $V_{Methoden} = m_v / m_g$		
Maßeinheit: -	**Skalentyp:** absolut ($V \in [0,1]$)	**Präferenz:** aufsteigend
Datenformate		
Inhalt: Anteil der Datenformate, die nach Durchsicht der Schnittstellen verstanden werden.		
Messung: Zählen der verstandenen Datenformate (d_v) und Vergleich mit der Gesamtanzahl (d_g).		
Formel: $V_{Datenformate} = d_v / d_g$		
Maßeinheit: -	**Skalentyp:** absolut ($V \in [0,1]$)	**Präferenz:** aufsteigend
Parameter		
Inhalt: Anteil der Parameter zur Anpassung, die nach Durchsicht der Schnittstellen verstanden werden.		
Messung: Zählen der verstandenen Parameter (p_v) und Vergleich mit der Gesamtanzahl (p_g).		
Formel: $V_{Parameter} = p_v / p_g$		
Maßeinheit: -	**Skalentyp:** absolut ($V \in [0,1]$)	**Präferenz:** aufsteigend
Dokumentation		
Inhalt: Anteil der Dienste, die nach Einsicht der Dokumentation verstanden werden.		
Messung: Zählen der verstandenen Dienste (d_v) und Vergleich mit der Gesamtanzahl (d_g).		
Formel: $V_{Beschreibung} = d_v / d_g$; Dienst umfasst Methode, Datenformate und Parameter zur Anpassung		
Maßeinheit: -	**Skalentyp:** absolut ($V \in [0,1]$)	**Präferenz:** aufsteigend
Erlernbarkeit		
Mittlere Nutzungszeit		
Inhalt: Zeit, die im Mittel bis zur korrekten Nutzung eines Dienstes benötigt wird.		
Messung: Messen der Zeit T_d bis zur korrekten Nutzung eines Dienstes d und Bilden des Mittels.		
Formel: $T_{Nutzung} = Sum(T_d) / d_g$; $d_g :=$ Gesamtanzahl der Dienste		
Maßeinheit: Minuten (m)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** absteigend
Mittlere Anpassungszeit		
Inhalt: Zeit, die im Mittel bis zur korrekten Anpassung eines Dienstes benötigt wird.		
Messung: Messen der Zeit T_d bis zur korrekten Anpassung eines Dienstes d und Bilden des Mittels.		
Formel: $T_{Anpassung} = Sum(T_d) / d_g$; $d_g :=$ Gesamtanzahl der anpassbaren Dienste		
Maßeinheit: Minuten (m)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** absteigend
Beherrschbarkeit		
Anpassbarkeit		
Inhalt: Anteil der durch Parameter anpassbaren Methoden.		
Messung: Zählen der anpassbaren Methoden (m_a) und Vergleich mit der Gesamtanzahl (m_g).		
Formel: $A = m_a / m_g$		
Maßeinheit: -	**Skalentyp:** absolut ($A \in [0,1]$)	**Präferenz:** aufsteigend

Abb. 4.47: Standardisierte Qualitätsmerkmale zur Beschreibung der Verwendbarkeit von Komponenten (in Anlehnung an [ISO/IEC 2003:27-40]).

Die *Verständlichkeit* gibt Aufschluss über den Aufwand, der mit der Einschätzung der Eignung einer Komponente und ihrer Anwendbarkeit im Rahmen eines Entwicklungsprojekts verbunden ist [ISO/IEC 2001:9]. Die *Erlernbarkeit* lässt Rückschlüsse auf den Aufwand zu, der von einem Entwickler zu betreiben ist, um eine Komponente einzusetzen [ISO/IEC 2001:9]. Die *Beherrschbarkeit* beschreibt die Möglichkeiten der Kontrolle, die beim Einsatz einer Komponente in einem Entwicklungsprojekt bestehen [ISO/IEC 2001:9]. Im Einzelnen lassen sich zur Spezifikation die in Abb. 4.47 dargestellten standardisierten Merkmale verwenden, die an den ISO 9126-2 Standard angelehnt sind [BERTOA und VALLECILLO 2002:8; ISO/IEC 2003:25-40]. Als Grundlage für die Messung der einzelnen Merkmalsausprägungen dienen dabei vor allem sog. *Nutzungstests*, die mit einer Reihe von Anwendern exemplarisch durchzuführen sind [ISO/IEC 2003:25].

Als geeignete Unterkategorien zur Beschreibung der *Wartbarkeit* von Komponenten ergeben sich die mit dem ISO Qualitätsmodell eingeführte Analysierbarkeit, die Änderbarkeit, die Stabilität sowie die Testbarkeit [ISO/IEC 2001:10f.]. Grundsätzlich sind dabei jedoch alle Merkmalstypen zu ignorieren, die sich auf die Durchführbarkeit eigener Wartungsarbeiten seitens des Anwendungsentwicklers beziehen [BERTOA und VALLECILLO 2002:3]. Diese ist bei der Verwendung von Komponenten schon wegen des zugrunde liegenden Blackbox-Prinzips ausgeschlossen. Somit bleiben zur Beschreibung der Wartbarkeit die in Abb. 4.48 dargestellten Merkmalstypen übrig, die dem ISO 9126-2 Standard entnommen wurden [ISO/IEC 2003:52-59]. Die *Analysierbarkeit* gibt Aufschluss über die für den Anwendungsentwickler bestehenden Möglichkeiten, auftretende Fehler jeweils den Methoden zuzuordnen, die sie verursachen [ISO/IEC 2001:10]. Die *Änderbarkeit* beschreibt vor allem den Zeitraum, der bis zum Eintreffen einer korrigierten Version vergeht [ISO/IEC 2001:10]. Die *Stabilität* gibt an, wie häufig mit Fehlern beim Betrieb der Komponente zu rechnen ist [ISO/IEC 2001:10]. Die *Testbarkeit* stellt schließlich dar, wie viele Methoden durch eine eingebaute Testfunktionalität überprüft werden können [ISO/IEC 2001:11]. Als Grundlage für die Messung der Merkmalsausprägungen lassen sich vor allem Beobachtungen aus der Pilot- bzw. Nutzungsphase verwenden. Diese Beobachtungen können ggf. durch Nutzungstests mit Anwendungsentwicklern ergänzt werden, die nach Abschluss der Komponentenentwicklung durchzuführen sind.

Die *Portabilität* einer Komponente lässt sich durch die Unterkategorien Installierbarkeit, Koexistenz und Ersetzbarkeit des ISO Qualitätsmodells beschreiben [ISO/IEC 2001:11]. Die im ISO 9126 Standard ebenfalls aufgeführte Anpassbarkeit lässt sich dagegen nicht sinnvoll anwenden, da sich die dort aufgeführten Merkmale auf eine Whitebox-Anpassung beziehen, die bei Komponenten grundsätzlich zu vermeiden ist (vgl. Abschnitt 3.1.3 zur Diskussion). Die *Installierbarkeit* gibt Auskunft über die Wahrscheinlichkeit, mit der ein

Anwendungsentwickler eine erfolgreiche Installation erwarten kann [ISO/IEC 2001:11]. Die *Koexistenz* beschreibt die Wahrscheinlichkeit, mit der während des Betriebs der Komponente Unverträglichkeiten zu anderen Komponenten aus der Software-Umgebung auftreten [ISO/IEC 2001:11]. Die *Ersetzbarkeit* erlaubt schließlich Rückschlüsse auf den Aufwand, der beim Ersetzen vorangehender Komponentenversionen zu erwarten ist [ISO/IEC 2001:11]. Insgesamt lassen sich zur Spezifikation die in Abb. 4.49 zusammengestellten Merkmalstypen des ISO 9126-2 Standards verwenden [ISO/IEC 2003:60-67].

Analysierbarkeit			
	Fehlerzuordnung		
		Inhalt: Anteil der Fehler, denen eine verursachende Methode zugeordnet werden kann.	
		Messung: Zählen der nicht zugeordneten Fehler (f_n) und Vergleich mit der Gesamtanzahl (f_g).	
		Formel: $F_{Zuordnung} = 1 - (f_n / f_g)$	
		Maßeinheit: -	**Skalentyp:** absolut ($F \in [0,1]$)	**Präferenz:** aufsteigend
	Mittlere Fehlerzuordnungszeit		
		Inhalt: Zeit, die im Mittel bis zur Zuordnung eines Fehlers zu einer verursachenden Methode benötigt wird.	
		Messung: Messen der Zeit T_f bis zur Zuordnung eines Fehlers f und Bilden des Mittels.	
		Formel: $T_{Zuordnung} = Sum(T_f) / f_g$; $f_g :=$ Gesamtanzahl der zugeordneten Fehler	
		Maßeinheit: Minuten (m)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** absteigend
	Überwachungsdaten		
		Inhalt: Anteil der Fehler, bei denen Überwachungsdaten für die Zuordnung zur Verfügung stehen.	
		Messung: Zählen der Fehler ohne Überwachungsdaten (f_f) und Vergleich mit der Gesamtanzahl (f_g).	
		Formel: $F_{Überwachungsdaten} = 1 - (f_f / f_g)$	
		Maßeinheit: -	**Skalentyp:** absolut ($F \in [0,1]$)	**Präferenz:** aufsteigend
Änderbarkeit			
	Fehlerbehebungszeit		
		Inhalt: Zeit, die nach der Fehlermeldung bis zur Behebung vergeht.	
		Messung: Messen der Zeit T_f von der Meldung bis zur Behebung eines Fehlers f.	
		Formel: $T_{Behebung} = T_f$; $T_f :=$ $Zeitpunkt_{Behebung}$ - $Zeitpunkt_{Meldung}$	
		Maßeinheit: Tage (d)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** absteigend
Stabilität			
	Fehlerhäufigkeit		
		Inhalt: Häufigkeit der bei der Ausführung von Methoden auftretenden Fehler pro Zeiteinheit.	
		Messung: Zählen der auftretenden Fehler (f_a) und Vergleich mit dem Ausführungszeitraum (T_a).	
		Formel: $F_{Häufigkeit} = f_a / T_a$	
		Maßeinheit: Fehler/Tag (faults/d)	**Skalentyp:** rational ($0 \leq F$)	**Präferenz:** absteigend
Testbarkeit			
	Testfunktion		
		Inhalt: Anteil der Methoden, für die Testfälle und -funktionen zur Verfügung stehen.	
		Messung: Zählen der Methoden mit Testfunktionen (m_t) und Vergleich mit der Gesamtanzahl (m_g).	
		Formel: $M_{Testfunktion} = m_t / m_g$	
		Maßeinheit: Tage (d)	**Skalentyp:** absolut ($M \in [0,1]$)	**Präferenz:** aufsteigend

Abb. 4.48: Standardisierte Qualitätsmerkmale zur Beschreibung der Wartbarkeit von Komponenten (in Anlehnung an [ISO/IEC 2003:53-59]).

Installierbarkeit			
Installation			
Inhalt: Anteil der Installationsvorgänge, die erfolgreich durchgeführt werden können.			
Messung: Zählen der nicht erfolgreichen Vorgänge (i_f) und Vergleich mit der Gesamtanzahl (i_g).			
Formel: $I = 1 - (i_f / i_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($I \in [0,1]$)	**Präferenz:** aufsteigend	
Koexistenz			
Unverträglichkeitshäufigkeit			
Inhalt: Häufigkeit der Unverträglichkeiten mit anderen Komponenten pro Zeiteinheit.			
Messung: Zählen der Unverträglichkeiten (u_f) und Vergleich mit dem Ausführungszeitraum (T_a).			
Formel: $U_{Häufigkeit} = u_f / T_a$			
Maßeinheit: Konflikt/Tag (conflict/d)	**Skalentyp:** rational ($0 \leq U$)	**Präferenz:** absteigend	
Ersetzbarkeit			
Methodenkompatibilität			
Inhalt: Anteil der Methoden der Vorgängerversion, die mit denen dieser Version kompatibel sind.			
Messung: Zählen der inkompatiblen Methoden (m_i) und Vergleich mit der Gesamtanzahl (m_g).			
Formel: $K_{Methoden} = 1 - (m_i / m_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($K \in [0,1]$)	**Präferenz:** aufsteigend	
Datenkompatibilität			
Inhalt: Anteil der Datenformate der Vorgängerversion, die mit denen dieser Version kompatibel sind.			
Messung: Zählen der inkompatiblen Datenformate (d_i) und Vergleich mit der Gesamtanzahl (d_g).			
Formel: $K_{Datenformate} = 1 - (d_i / d_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($K \in [0,1]$)	**Präferenz:** aufsteigend	
Parameterkompatibilität			
Inhalt: Anteil der Parameter der Vorgängerversion, die mit denen dieser Version kompatibel sind.			
Messung: Zählen der inkompatiblen Parameter (p_i) und Vergleich mit der Gesamtanzahl (p_g).			
Formel: $K_{Parameter} = 1 - (p_i / p_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($K \in [0,1]$)	**Präferenz:** aufsteigend	

Abb. 4.49: Standardisierte Qualitätsmerkmale zur Beschreibung der Portabilität von Komponenten (in Anlehnung an [ISO/IEC 2003:61-67]).

4.6.1.2 Funktionsweise

Die Funktionsweise der von einer Komponente angebotenen bzw. nachgefragten Dienste lässt sich durch die Merkmale beschreiben, die als Bestandteile der Qualitätskategorie *Funktionalität* mit dem ISO 9126 Qualitätsmodell eingeführt werden. Die Beschreibung der Funktionsweise vervollständigt dabei die Spezifikation der fachlichen Funktionalität, die Bestandteil der Blue Pages ist. Hier werden zusätzliche Aspekte spezifiziert, die sich in die Unterkategorien Eignung, Genauigkeit, Interoperabilität, Sicherheit, Persistenz und Übereinstimmung zerlegen lassen. Bis auf die Persistenz, die bei der Erstellung des UNSCOM Merkmalskatalogs ergänzt wurde, entstammen sämtliche Unterkategorien dem ISO 9126 Standard [ISO/IEC 2001:7f.]. Neben der Sicherheit und Persistenz bilden sich mit der Entwicklung der aspektorientierten Programmierung allerdings weitere Aspekte heraus, die dem Merkmalskatalog schrittweise hinzuzufügen sein werden.

Eignung			
Angemessenheit			
Inhalt: Anteil der Methoden, deren Realisierung zur Erfüllung der fachlichen Funktionen angemessen ist.			
Messung: Zählen der nicht angemessenen Methoden (m_f) und Vergleich mit der Gesamtanzahl (m_g).			
Formel: $E_{Angemessenheit} = 1 - (m_f / m_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($E \in [0,1]$)	**Präferenz:** aufsteigend	
Vollständigkeit			
Inhalt: Anteil der spezifizierten fachlichen Funktionen, die durch Methoden abgedeckt werden.			
Messung: Zählen der nicht abgedeckten Funktionen (a_f) und Vergleich mit der Gesamtanzahl (a_g).			
Formel: $E_{Vollständigkeit} = 1 - (a_f / a_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($E \in [0,1]$)	**Präferenz:** aufsteigend	
Stabilität			
Inhalt: Anteil der Methoden, deren Spezifikation bis zum Abschluss der Testphase stabil geblieben ist.			
Messung: Zählen der modifizierten Methoden (m_m) und Vergleich mit der Gesamtanzahl (m_g).			
Formel: $E_{Stabilität} = 1 - (m_m / m_g)$			
Maßeinheit: -	**Skalentyp:** absolut ($E \in [0,1]$)	**Präferenz:** aufsteigend	
Genauigkeit			
Präzision			
Inhalt: Häufigkeit des Auftretens von Ergebnissen mit unzureichender Präzision pro Zeiteinheit.			
Messung: Zählen der unpräzisen Ergebnisse (e_u) und Vergleich mit dem Ausführungszeitraum (T_a).			
Formel: $G_{Präzision} = e_u / T_a$			
Maßeinheit: Ergebnis/Tag (result/d)	**Skalentyp:** rational ($0 \leq G$)	**Präferenz:** absteigend	
Berechnungsgenauigkeit			
Inhalt: Häufigkeit des Auftretens von Ergebnissen mit unzureichender Genauigkeit pro Zeiteinheit.			
Messung: Zählen der ungenauen Ergebnisse (e_u) und Vergleich mit dem Ausführungszeitraum (T_a).			
Formel: $G_{Berechnungsgenauigkeit} = e_u / T_a$			
Maßeinheit: Ergebnis/Tag (result/d)	**Skalentyp:** rational ($0 \leq G$)	**Präferenz:** absteigend	
Interoperabilität			
Datenstandard			
Inhalt: Gibt an, ob die bei Benutzung einer Methode ausgetauschten Daten einem Standard entsprechen.			
Messung: Ermitteln, ob sämtliche ausgetauschten Daten einer Methode einem Standard entsprechen.			
Formel: $I_{Datenstandard}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal ($I \in$ {nein, ja})	**Präferenz:** aufsteigend	
Funktionsstandard			
Inhalt: Gibt an, ob eine Methode einem Standard entspricht.			
Messung: Ermitteln, ob die Funktion einem Standard entspricht.			
Formel: $I_{Funktionsstandard}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal ($I \in$ {nein, ja})	**Präferenz:** aufsteigend	
Schnittstellenstandard			
Inhalt: Gibt an, ob eine Schnittstelle einem Standard entspricht.			
Messung: Ermitteln, ob die Schnittstelle einem Standard entspricht.			
Formel: $I_{Schnittstellenstandard}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal ($I \in$ {nein, ja})	**Präferenz:** aufsteigend	

Abb. 4.50: Standardisierte Qualitätsmerkmale zur Beschreibung der Funktionalität von Komponentendiensten (in Anlehnung an [BERTOA und VALLECILLO 2002:6f.; ISO/IEC 2003:7-10]).

Die *Eignung* gibt Aufschluss über die Angemessenheit der realisierten Funktionalität [ISO/IEC 2001:8]. Die *Genauigkeit* beschreibt die Güte der von einem Dienst zurück gelieferten Ergebnisse [ISO/IEC 2001:8]. Die *Interoperabilität* gibt Hinweise auf die Fähigkeit zur Zusammenarbeit (Interaktion) mit anderen Komponenten [ISO/IEC 2001:8]. Die *Sicherheit* beschreibt Maßnahmen zur Sicherung von Daten und zum Zugriffsschutz [ISO/IEC 2001:8]. Die *Persistenz* spezifiziert die Strategie, die zur Speicherung von Ergebniswerten eingeschlagen wird [ISO/IEC 2001:8]. Die *Übereinstimmung* gibt an, ob ein Dienst einem fachlichen Standard entspricht bzw. zertifiziert ist [ISO/IEC 2001:8].

Sicherheit			
Verschlüsselung			
Inhalt: Gibt an, ob die bei Benutzung einer Methode ausgetauschten Daten verschlüsselt werden.			
Messung: Ermitteln, ob sämtliche ausgetauschten Daten einer Methode verschlüsselt werden.			
Formel: $S_{Verschlüsselung}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {nein, ja})	**Präferenz:** aufsteigend	
Zugriffsschutz			
Inhalt: Gibt an, ob eine Methode über einen Zugriffsschutz verfügt.			
Messung: Ermitteln, ob die Funktion einen Zugriffsschutz besitzt.			
Formel: $S_{Zugriffsschutz}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {nein, ja})	**Präferenz:** aufsteigend	
Überwachungsprotokoll			
Inhalt: Gibt an, ob Zugriffe auf eine Methode protokolliert werden.			
Messung: Ermittle, ob die Zugriffe auf eine Methode protokolliert werden.			
Formel: $S_{Überwachungsprotokoll}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {nein, ja})	**Präferenz:** aufsteigend	
Persistenz			
Persistenzmanagement			
Inhalt: Gibt die Strategie zur dauerhaften Speicherung der Ergebnisdaten eines Dienstes an.			
Messung: Ermitteln der Strategie zur dauerhaften Datenspeicherung.			
Formel: P = ohne | container | komponente; ohne < container, ohne < komponente			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {ohne, container, komponente})	**Präferenz:** aufsteigend	
Übereinstimmung			
Funktionalstandard			
Inhalt: Gibt an, ob die Funktionalität der Methode einem Standard bzw. einer Vorschrift entspricht.			
Messung: Ermitteln, ob die Funktionalität der Methode einem Standard bzw. einer Vorschrift entspricht.			
Formel: $S_{Funktionalstandard}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {nein, ja})	**Präferenz:** aufsteigend	
Zertifizierung			
Inhalt: Gibt an, ob die Funktionalität der Methode zertifiziert ist.			
Messung: Ermitteln, ob die spezifizierte Funktionalität der Methode durch ein Zertifikat bescheinigt wird.			
Formel: $S_{Zertifizierung}$ = nein | ja; nein < ja			
Maßeinheit: -	**Skalentyp:** ordinal (S ∈ {nein, ja})	**Präferenz:** aufsteigend	

Abb. 4.51: Standardisierte Qualitätsmerkmale zur Beschreibung der Funktionalität von Komponentendiensten (in Anlehnung an [BERTOA und VALLECILLO 2002:5-7; ISO/IEC 2003:11-13]).

Im Einzelnen lassen sich zur Spezifikation der Funktionalität die in Abb. 4.50 und Abb. 4.51 dargestellten Merkmalstypen verwenden, die dem ISO 9126-2 Standard entnommen bzw. in Anlehnung an diesen konzipiert wurden [BERTOA und VALLECILLO 2002:5-7; ISO/IEC 2003:5-13]. Als Grundlage zur Messung der verschiedenen Merkmalsausprägungen dienen neben den Ergebnissen der Komponentenimplementierung dabei vor allem die Ergebnisse der *Testphase*, die im Rahmen der Stabilisierung durchgeführt wird.

4.6.1.3 Performanz

Die Performanz der von einer Komponente angebotenen bzw. nachgefragten Dienste lässt sich unter Rückgriff auf die Merkmale beschreiben, die als Bestandteile der Qualitätskategorien Zuverlässigkeit und Effizienz mit dem ISO 9126 Qualitätsmodell eingeführt werden. Verglichen mit der Literatur, in der der Begriff „Performanz" häufig synonym zu „Effizienz" verwendet wird, nutzt der UNSCOM Merkmalskatalog damit ein breiter gefasstes Begriffsverständnis. Die Zuverlässigkeit und Effizienz werden vom ISO Standard in Unterkategorien unterteilt, die die charakteristischen Merkmalstypen enthalten. Zur Beschreibung der *Zuverlässigkeit* eines Dienstes eignen sich dabei die Ausgereiftheit, die Fehlertoleranz und die Wiederherstellbarkeit [ISO/IEC 2001:8f.]. Die *Ausgereiftheit* gibt Aufschluss über die zu erwartenden Ausfälle bzw. die Fehlerzustände, die beim Betrieb auftreten und ggf. zu Ausfällen führen können [ISO/IEC 2001:8]. Die *Fehlertoleranz* beschreibt das Verhalten einer Komponente beim Auftreten von Fehlern und ihre Fähigkeit, in einem solchen Fall den Ausfall des entsprechenden Dienstes zu vermeiden [ISO/IEC 2001:9]. Die *Wiederherstellbarkeit* beinhaltet schließlich Angaben zur Effizienz, mit der sich die Funktionalität eines Dienstes nach einem Ausfall wiederherstellen lässt [ISO/IEC 2001:9].

Zur Beschreibung lassen sich die in Abb. 4.52 und Abb. 4.53 dargestellten Merkmalstypen verwenden, die entweder dem ISO 9126-2 Standard entnommen oder in Anlehnung an diesen konzipiert wurden [BERTOA und VALLECILLO 2002:5, 7; ISO/IEC 2003:14-24]. Zur Spezifikation der Merkmalsausprägungen lassen sich neben den Ergebnissen der Komponentenimplementierung vor allem die Resultate der *Testphase* nutzen, die während der Stabilisierung durchgeführt wird. Zu beachten ist dabei, dass während den späten Testphasen eine *geringere* Menge erkannter Fehler bzw. Ausfälle als qualitativ besser empfunden wird [ISO/IEC 2003:15]. Grundlage für diese Präferenz ist die Tatsache, dass ein häufiges Auftreten von Fehlern bzw. Ausfällen während der späten Testphasen im Allgemeinen erwarten lässt, dass außer diesen noch weitere bislang verborgene Fehler bzw. Ausfälle auftreten [BERTOA und VALLECILLO 2002:7]. Diese treten unter Umständen erst nach der Behebung der im Test erkannten Probleme auf. Allerdings gilt die genannte Präferenz nur bei einem genügend hohen Grad der Testabdeckung. Erst in diesem Fall besitzt eine Aussage über die erkannten Fehler nämlich auch die notwendige Aussagekraft.

Ausgereiftheit			
Ausfalldichte			
	Inhalt: Anteil der Ausfälle, die bei den abschließenden Testdurchläufen auftraten.		
	Messung: Zählen der auftretenden Ausfälle (a_a) und Vergleich mit der Zahl der untersuchten Testfälle (t_g).		
	Formel: $A_{Ausfalldichte} = a_a / t_g$		
	Maßeinheit: -	**Skalentyp:** absolut ($0 \leq A$)	**Präferenz:** absteigend
Ausfallbeseitigung			
	Inhalt: Anteil der Ausfälle, die nach Korrekturmaßnahmen nicht mehr aufgetreten sind.		
	Messung: Zählen der behobenen Ausfälle (a_b) und Vergleich mit der Gesamtanzahl erkannter Ausfälle (a_g).		
	Formel: $A_{Ausfallbeseitigung} = a_b / a_g$		
	Maßeinheit: -	**Skalentyp:** absolut ($A \in [0,1]$)	**Präferenz:** aufsteigend
Fehlerdichte			
	Inhalt: Anteil der Fehler, die bei den abschließenden Testdurchläufen auftraten.		
	Messung: Zählen der auftretenden Fehler (f_a) und Vergleich mit der Zahl der untersuchten Testfälle (t_g).		
	Formel: $A_{Fehlerdichte} = f_a / t_g$		
	Maßeinheit: -	**Skalentyp:** absolut ($0 \leq A$)	**Präferenz:** absteigend
Fehlerbeseitigung			
	Inhalt: Anteil der Fehler, die nach Korrekturmaßnahmen nicht mehr aufgetreten sind.		
	Messung: Zählen der behobenen Fehler (f_b) und Vergleich mit der Gesamtanzahl erkannter Fehler (f_g).		
	Formel: $A_{Fehlerbeseitigung} = f_b / f_g$		
	Maßeinheit: -	**Skalentyp:** absolut ($A \in [0,1]$)	**Präferenz:** aufsteigend
Testabdeckung			
	Inhalt: Anteil der durchgeführten Tests im Vergleich zur optimalen/geforderten Testabdeckung.		
	Messung: Zählen der durchgeführten Tests (t_d) und Vergleich mit der optimalen/geforderten Testzahl (t_g).		
	Formel: $A_{Testabdeckung} = t_d / t_g$		
	Maßeinheit: -	**Skalentyp:** absolut ($A \in [0,1]$)	**Präferenz:** aufsteigend
Zeit zwischen Ausfällen			
	Inhalt: Laufzeit, die zwischen zwei Ausfällen vergeht.		
	Messung: Messen der Laufzeit T_L und Vergleich mit den auftretenden Ausfällen a_g.		
	Formel: $T_{Ausfallzeit} = T_L / a_g$		
	Maßeinheit: Tage (d)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** aufsteigend
Zeit zwischen Versionen			
	Inhalt: Zeit, die zwischen dem Erscheinen zweier Versionen vergeht.		
	Messung: Messen der Zeit T_{VxVy} zwischen zwei Versionen V_x und V_y.		
	Formel: $T_{Versionszeit} = T_{VxVy}$		
	Maßeinheit: Tage (d)	**Skalentyp:** rational ($0 < T$)	**Präferenz:** aufsteigend

Abb. 4.52: Standardisierte Qualitätsmerkmale zur Beschreibung der Zuverlässigkeit von Komponentendiensten (in Anlehnung an [BERTOA und VALLECILLO 2002:7; ISO/IEC 2003:15-18]).

Die *Effizienz* der von einer Komponente angebotenen bzw. nachgefragten Dienste lässt sich sowohl hinsichtlich des Zeitverhaltens als auch hinsichtlich des Ressourcenverhaltens beschreiben [ISO/IEC 2001:10]. Das *Zeitverhalten* lässt dabei vor allem Rückschlüsse auf die sich bei der Ausführung zur Laufzeit ergebenden Antwortzeiten und den Durchsatz zu [ISO/IEC 2001:10]. Das *Ressourcenverhalten* enthält Informationen zum Haupt- bzw. Festspeicherbedarf, der bei der Ausführung eines Dienstes entsteht [ISO/IEC 2001:10]. Im

Einzelnen können zur Spezifikation der Effizienz die in Abb. 4.54 zusammenfassend dargestellten Merkmalstypen verwendet werden, die dem ISO 9126-2 Standard entnommen wurden [ISO/IEC 2003:41-51].

Fehlertoleranz		
Fehlerbehandlung		
Inhalt: Strategie, die zur Fehlerbehandlung verwendet wird.		
Messung: Ermitteln der Strategie, die beim Auftreten eines Fehlers verfolgt wird.		
Formel: F_{Fb} = keine | erkennung | warnung | behandlung; keine < erkennung < warnung < behandlung		
Maßeinheit: -	**Skalentyp:** ordinal (F ∈ {keine, erkennung, warnung, behandlung})	**Präferenz:** aufsteigend
Ausfallvermeidung		
Inhalt: Anteil der Fehler, die nicht zum Ausfall führen.		
Messung: Zählen der Ausfälle (a_g) und Vergleich mit der Gesamtanzahl der Fehler (f_g).		
Formel: $F_{Av} = 1 - (a_g / f_g)$		
Maßeinheit: -	**Skalentyp:** absolut (F ∈ [0,1])	**Präferenz:** aufsteigend
Wiederherstellbarkeit		
Verfügbarkeit		
Inhalt: Anteil der Aufrufe, bei denen die Funktionalität einer Methode verfügbar ist.		
Messung: Zählen der erfolgreichen Aufrufe (a_e) und Vergleich mit der Gesamtanzahl (a_g).		
Formel: $W_{Verfügbarkeit} = a_e / a_g$		
Maßeinheit: -	**Skalentyp:** absolut (W ∈ [0,1])	**Präferenz:** aufsteigend
Ausfallzeit		
Inhalt: Zeit, für die eine Methode ausfällt.		
Messung: Messen der Ausfallzeit T_a der Methode.		
Formel: $T_{Ausfallzeit} = T_a$		
Maßeinheit: Sekunden (s)	**Skalentyp:** rational (0 < T)	**Präferenz:** absteigend
Wiederherstellungszeit		
Inhalt: Zeit, die bis zur Wiederherstellung der Funktionalität einer Methode vergeht.		
Messung: Messen der Wiederherstellungszeit T_w der Methode.		
Formel: $T_{Wiederherstellungszeit} = T_w$		
Maßeinheit: Sekunden (s)	**Skalentyp:** rational (0 < T)	**Präferenz:** absteigend
Häufigkeit der automatischen Wiederherstellung		
Inhalt: Häufigkeit der erfolgreichen automatischen Wiederherstellung der Funktionalität.		
Messung: Zählen der Wiederherstellungen (w_e) und Vergleich mit der Zahl der provozierten Ausfälle (a_g).		
Formel: $W_{Häufigkeit} = w_e / a_g$		
Maßeinheit: -	**Skalentyp:** absolut (W ∈ [0,1])	**Präferenz:** aufsteigend

Abb. 4.53: Standardisierte Qualitätsmerkmale zur Beschreibung der Zuverlässigkeit von Komponentendiensten (in Anlehnung an [BERTOA und VALLECILLO 2002:7; ISO/IEC 2003:19-24]).

Als Grundlage für die Messung der Effizienzmerkmale dienen vor allem *Last- und Konfigurationstests*, die während der Stabilisierungsphase durchgeführt werden. Prinzipiell ist bei der Messung jedoch zu beachten, dass die Performanz einer Komponente von externen Faktoren beeinflusst wird. Zu diesen Faktoren zählt das *Benutzungsprofil*, das die Performanz der Komponente nachhaltig verändern kann. So kann bspw. die Beschaffenheit der übergebenen Parameter ein unterschiedliches Laufzeitverhalten bewirken. Daneben wirkt

sich auch die Lastsituation, die sich aus der jeweiligen Art der Nutzung ergibt, auf die Performanz aus. Bei einer logischen Wiederverwendung wird die Performanz außerdem von der jeweils unterschiedlichen *Umwelt* und der *Hardware-Konfiguration* beeinflusst. Zum einen wirkt sich nämlich die Performanz der anderen Komponenten des Anwendungssystems, die während der Informationsverarbeitung von der zu beschreibenden Komponente aufgerufen werden, auf die Performanz der zu beschreibenden Komponente aus. Zum anderen zeigt sich auch in Abhängigkeit von der jeweils eingesetzten Hardware ein unterschiedliches Laufzeitverhalten. Da eine formelbasierte Spezifikation der Performanz, in der alle diese Abhängigkeiten exakt beschrieben werden könnten, derzeit nicht zur Verfügung steht, bleiben als Alternativen nur die Verwendung von (standardisierten) *Referenzumgebungen und Referenzszenarien* oder die Generierung einer hinreichend *umfassenden Probe*, in die möglichst viele Faktorgrößen eingehen [ACKERMANN, et al. 2002:13]. Bei der Spezifikation von Qualitätseigenschaften mit dem UNSCOM Spezifikationsrahmens wird letztere Vorgehensweise empfohlen, da das Aufstellen von Referenzumgebungen sehr aufwändig ist und die festgelegten Umgebungen zudem rasch veralten. Daher wäre die Aussagekraft der Messergebnisse bei einem solchen Vorgehen grundsätzlich in Frage zu stellen [ACKERMANN, et al. 2002:13].

Zeitverhalten			
Antwortzeit			
Inhalt: Zeit vom Aufruf einer Methode bis zum Abschluss ihrer Ausführung.			
Messung: Messen der Zeitpunkte, zu denen die Methode aufgerufen (t_a) und abgeschlossen (t_e) wurde.			
Formel: $T_{Antwortzeit} = t_e - t_a$			
Maßeinheit: Sekunden (s)	**Skalentyp:** rational (0 < T)	**Präferenz:** absteigend	
Durchsatz			
Inhalt: Anzahl der pro Zeiteinheit gleichzeitig ausführbaren Methodenaufrufe.			
Messung: Zählen der ausgeführten Aufrufe (a_a) und Vergleich mit dem Zeitraum (T).			
Formel: $Z_{Durchsatz} = a_a / T$			
Maßeinheit: Aufrufe/Sekunde (calls/s)	**Skalentyp:** rational (0 < Z)	**Präferenz:** aufsteigend	
Ressourcenverhalten			
Speicherverbrauch			
Inhalt: Speicher, der zur Ausführung einer Methode benötigt wird.			
Messung: Messen des maximalen Speicherbedarfs (m_{max}) während der Ausführung.			
Formel: $R_{Speicherverbrauch} = m_{max}$			
Maßeinheit: bytes (b)	**Skalentyp:** absolut (0 < R)	**Präferenz:** absteigend	
Plattenverbrauch			
Inhalt: Platz auf der Festplatte, der zur Ausführung einer Methode benötigt wird.			
Messung: Messen des maximalen Platzbedarfs (p_{max}) während der Ausführung.			
Formel: $R_{Plattenverbrauch} = p_{max}$			
Maßeinheit: bytes (b)	**Skalentyp:** absolut (0 < R)	**Präferenz:** absteigend	

Abb. 4.54: Standardisierte Qualitätsmerkmale zur Beschreibung der Effizienz von Komponentendiensten (in Anlehnung an [ISO/IEC 2003:42-51]).

Bei Bedarf lassen sich dem ISO 9126-2 Standard weitere, jedoch weniger allgemein verwendbare Merkmalstypen, entnehmen und in die Spezifikation einbinden. Weitere Vorschläge für Qualitätsmerkmale finden sich in [ISO/IEC 1994; ISO/IEC 1999] und zahlreichen anderen Arbeiten, die bislang noch nicht standardisiert wurden. Diese können als herstellerspezifische Merkmalstypen im Metaschema definiert und dann ebenfalls im Rahmen der Grey Pages genutzt werden. Anzumerken bleibt abschließend, dass sich die zur Beschreibung einer Komponente heranzuziehenden Qualitätsmerkmale je nach Art der Wiederverwendung unterscheiden können. So kann bei einer physischen Wiederverwendung bspw. auf die Beschreibung des Ressourcenverhaltens und der Installierbarkeit verzichtet werden.

4.6.2 Repräsentation

Zur Darstellung von Qualitätsmerkmalen und Merkmalsausprägungen wird sowohl im UNSCOM/T als auch im UNSCOM/G Format die sog. *Quality of Service Modeling Language* (QML [FROLUND und KOISTINEN 1998]) verwendet. Diese textbasierte Notation unterstützt die Vereinbarung von Qualitätskategorien und Merkmalstypen ebenso wie die Beschreibung von Merkmalsausprägungen und deren Zuordnung zu Komponenten bzw. Schnittstellen (-elementen) [FROLUND und KOISTINEN 1998:4]. Über ein eigens definiertes UML Profil lassen sich die in QML dargestellten Merkmalsausprägungen außerdem als Bestandteile in UML Komponentendiagramme einbinden [FROLUND und KOISTINEN 1998:27f.]. Die QML unterstützt damit gleichzeitig die Anreicherung von Komponentendiagrammen mit Qualitätseigenschaften und ist in der Summe ihrer Fähigkeiten deshalb anderen Notationen wie der QDL (QoS Definition Language [DANIEL, et al. 1999]), QuO (QoS for Objects [ZINKY, et al. 1997; LOYALL, et al. 1998]) oder der QIDL (Quality IDL [BECKER und GEIHS 1999; BECKER und GEIHS 2000]) überlegen (zur ausführlichen Analyse und Gegenüberstellung vgl. [AAGEDAL 2001:25-36]). In UNSCOM/X werden die spezifizierten Qualitätseigenschaften durch ein XML Dokument repräsentiert, das nach den Regeln eines eigens definierten XML Schemas zu strukturieren ist. Dieses Schema wird in Anhang B näher dargestellt. In diesem Abschnitt wird dagegen ausschließlich die Repräsentation von Qualitätseigenschaften in UNSCOM/T und UNSCOM/G betrachtet.

Mit den Sprachkonzepten der QML lassen sich Qualitätskategorien und Merkmalstypen (wie z.B. die des Katalogs) sowie die dazugehörigen Ausprägungen auf eine einheitliche Weise darstellen [FROLUND und KOISTINEN 1998:3f.]. Die Vereinbarung einer *Qualitätskategorie* wird durch das Schlüsselwort `type` eingeleitet (vgl. Abb. 4.55). Daran anschließend ist ein *Name* für die zu vereinbarende Qualitätskategorie zu vergeben und eine Reihe von

Merkmalstypen zu definieren, die Bestandteil der Kategorie werden. Die Merkmalstypen sind in geschweiften Klammern zusammenzufassen, mit dem Schlüsselwort contract auszuzeichnen und der Kategorie zuzuweisen [FROLUND und KOISTINEN 1998:16]. Die vereinbarten Qualitätskategorien stellen zugleich *Vertragstypen* dar, denen die bei der Spezifikation der Merkmalsausprägungen definierten Verträge entsprechen müssen.

```
type Usability = contract {
   methodUnderstandability : increasing numeric;
   dataformatUnderstandability : increasing numeric;
   paramterUnderstandability : increasing numeric;
   documentationUnderstandability : increasing numeric;
   meanTimeToUse : decreasing numeric m;
   meanTimeToAdapt : decreasing numeric m;
   adaptability : increasing numeric;
};

type Reliability = contract {
   ...
   failureDensity : decreasing numeric;
   faultDensity : decreasing numeric;
   timeBetweenFailures : increasing numeric d;
   faultHandling : increasing enum {none, detection, warning, handling} with order
                   {none < detection, detection < warning, warning < handling};
   availability : increasing numeric;
   downTime : decreasing numeric s;
   timeToRecover : decreasing numeric s;
   ...
};

type Efficiency = contract {
   responseTime : decreasing numeric s;
   throughput : increasing numeric calls/s;
   memoryUtilization : decreasing numeric b;
   discUtilization : decreasing numeric b;
};
```

Abb. 4.55: Spezifikation der Qualitätskategorien und Merkmalstypen in UNSCOM/T und UNSCOM/G.

Die Definition eines *Merkmalstyps* beginnt mit der Vergabe eines *Namens* [FROLUND und KOISTINEN 1998:11]. Darüber hinaus ist ein *Wertebereich* festzulegen, der entweder aus den reellen Zahlen oder einer Aufzählung von Werten bestehen kann [FROLUND und KOISTINEN 1998:11]. Um die Menge der reellen Zahlen als Wertebereich festzulegen, ist das Schlüsselwort numeric zu verwenden (vgl. Abb. 4.55). Eine Aufzählung von Werten ist hingegen mit dem Schlüsselwort enum (Ausprägungen sind Einzelwerte) oder set (Ausprägungen sind Mengen) einzuleiten. Danach sind die jeweiligen Werte in geschweiften Klammern aufzuführen. Im Anschluss an die Aufzählung der Einzelwerte kann optional eine (transitive) partielle Ordnung definiert werden, die mit dem Schlüsselwort with order einzuleiten ist. Nach dem Schlüsselwort ist eine Menge von geordneten Wertepaaren in geschweiften Klammern anzugeben, die jeweils der Vorschrift $Wert_1 < Wert_2$ entsprechen müssen [FROLUND und KOISTINEN 1998:13]. Falls eine Ordnung definiert oder die Menge der reellen Zahlen als Wertebereich festgelegt wurde, ist außerdem eine *Präferenz* anzugeben. Die Präferenz legt fest, ob größere (aufsteigend) oder kleinere (absteigend) Werte als qualitativ besser angesehen werden, und ist durch Voranstellen des Schlüsselworts increasing bzw. decreasing vor den Wertebereich anzugeben [FROLUND und

KOISTINEN 1998:14]. Schließlich kann für jeden Merkmalstyp optional noch eine Maßeinheit festgelegt werden, die an den Schluss der Definition zu stellen ist (vgl. Abb. 4.55).

Qualitätsprofile lassen sich in QML mit dem Schlüsselwort profile vereinbaren [FROLUND und KOISTINEN 1998:20]. Bei der Definition eines Profils sind zunächst ein *Name* und die *Bezugsobjekte* zu spezifizieren, wobei letztere auf das Schlüsselwort for folgen (vgl. Abb. 4.56). Als Bezugsobjekte dürfen nach den Vorgaben des UNSCOM Metaschemas entweder Komponenten oder Schnittstellen genannt werden. Damit werden die Vorgaben der QML erweitert, die zunächst ausschließlich die Vereinbarung von Profilen für Schnittstellen vorsieht [FROLUND und KOISTINEN 1998:20]. Jedes Profil besteht aus einer Menge von *Verträgen*, die jeweils Merkmalsausprägungen einer Qualitätskategorie enthalten. Die Menge der Verträge ist hinter dem Schlüsselwort profile in geschweiften Klammern anzugeben (vgl. Abb. 4.56). Für jeden Vertrag kann außerdem das *Bezugsobjekt* weiter eingeschränkt werden.

```
LagermanagementVerwendbarkeit for Lagermanagement = profile {
   require Usability contract {
      documentationUnderstandability > 0.9;
      meanTimeToUse {mean < 60 m;};
   };
};

BestandsvwtNormallast for IBestandsvwt = profile {
   from buche require Efficiency contract {
      responseTime {mean < 2 s;};
      throughput {mean > 15 calls/s; percentile 80 > 10 calls/s;};
   };
};

DatenbankNormallast for IDatenbank = profile {
   from verarbeiteSQL require Efficiency contract {
      responseTime {mean < 0.5 s;};
      throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
   };
};

BestandsvwtHochlast for IBestandsvwt = profile {
   from buche require Efficiency contract {
      responseTime {mean < 1 s;};
      throughput {mean > 20 calls/s; percentile 80 > 15 calls/s};
   };
};

DatenbankHochlast for IDatenbank = profile {
   from verarbeiteSQL require Efficiency contract {
      responseTime {mean < 0.25 s;};
      throughput {mean > 200 calls/s; percentile 80 > 180 calls/s;};
   };
};
```

Abb. 4.56: Spezifikation der Qualitätsprofile und Merkmalsausprägungen in UNSCOM/T und UNSCOM/G.

Innerhalb eines Profils erfolgt die Vereinbarung eines Vertrags mit den Schlüsselworten require contract, zwischen denen der Name der zugrunde gelegten Qualitätskategorie zu nennen ist [FROLUND und KOISTINEN 1998:21]. Durch Voranstellen des Schlüsselworts from lassen sich bei der Vereinbarung die Bezugsobjekte weiter eingrenzen (vgl. Abb. 4.56). Ist für einen Vertrag kein Bezugsobjekt genannt, so gilt er je nach dem Bezugsobjekt

des Profils entweder für die Komponente als Ganzes oder für alle Elemente der Schnittstelle als *Standardvertrag* [FROLUND und KOISTINEN 1998:21]. Die in einem Standardvertrag festgelegten Qualitätseigenschaften lassen sich anschließend durch speziellere Verträge für einzelne Schnittstellenelemente redefinieren. Sie stellen jedoch Mindestwerte dar, die bei der Redefinition nicht unterschritten werden dürfen. Die Merkmalsausprägungen des jeweiligen Vertrags sind schließlich nach dem Schlüsselwort contract in geschweiften Klammern anzugeben. Ein Vertrag kann dabei grundsätzlich Ausprägungen für eine beliebige Zahl von Merkmalstypen beinhalten, so dass nicht für jeden Merkmalstyp der zu beschreibenden Qualitätskategorie eine Ausprägung angegeben werden muss.

Um eine *Merkmalsausprägung* zu beschreiben, ist zunächst der Name des Merkmalstyps zu nennen. Im Anschluss daran ist entweder eine absolute Aussage oder eine Reihe von statistischen Aussagen in geschweiften Klammern anzugeben (vgl. Abb. 4.56). Als statistische Aussagen dürfen Angaben zum Mittelwert (mean), zur Varianz (variance), den verschiedenen Perzentilen (percentil x) sowie der Frequenz (frequency) auftreten [FROLUND und KOISTINEN 1998:12]. Zu beachten ist, dass bei absoluten Aussagen über nominalskalierte Merkmale stets eine exakte Übereinstimmung zu formulieren ist (vgl. Abb. 4.56). Bei Merkmalen, die zumindest ordinalskaliert sind, ist bei einer absoluten Aussage oder der Angabe des Mittelwerts gemäß den Konventionen des UNSCOM Metaschemas hingegen eine Schranke festzulegen. Wurde mit dem Merkmalstyp eine Maßeinheit festgelegt, muss der Wert der Ausprägung außerdem dieser Maßeinheit entsprechen. Die Maßeinheit ist in diesem Fall hinter dem Wert zu wiederholen [FROLUND und KOISTINEN 1998:12].

```
LagermanagementNormallast for Lagermanagement = servicelevel {
   require BestandsvwtNormallast, DatenbankNormallast;
};

LagermanagementHochlast for Lagermanagement = servicelevel {
   require BestandsvwtHochlast, DatenbankHochlast;
};
```

Abb. 4.57: Spezifikation der Service Level in UNSCOM/T und UNSCOM/G.

Nicht unterstützt wird durch QML dagegen die im UNSCOM Metaschema vorgesehene Vereinbarung eines oder mehrerer *Service Level*. Deshalb ist eine Erweiterung der QML vorzunehmen, durch die die Vereinbarung von Service Levels möglich wird (vgl. Abb. 4.57). Die Definition eines Service Levels erfolgt mit dem der QML Grammatik hinzugefügten Schlüsselwort servicelevel und ist mit der Vergabe eines Namens einzuleiten. Jedes Service Level ist mit dem Schlüsselwort for mindestens einer Komponente zuzuweisen und fasst eine Menge von Qualitätsprofilen in geschweiften Klammern zusammen. Die Zusammenfassung von Qualitätsprofilen ist mit dem Schlüsselwort require einzuleiten.

In UNSCOM/G lassen sich definierte Qualitätsprofile und Service Level durch eine speziell definierte *UML Abbildung* auch in die Komponentendiagramme integrieren und mit einzelnen Elementen graphisch in Bezug setzen [FROLUND und KOISTINEN 1998:27f.]. Qualitätsprofile dürfen dabei mit Komponenten oder Beziehungen assoziiert werden, die zwischen einer Komponente und einer Angebots- bzw. Nachfrageschnittstelle bestehen (vgl. Abb. 4.58). Letzteres ist sowohl in der ausführlichen als auch der verkürzten Darstellungsform möglich (zur Erläuterung dieser Darstellungsformen vgl. Abschnitt 4.5.2.1). Service Level dürfen dagegen ausschließlich mit einer Komponente assoziiert werden. Um die Assoziation herzustellen, ist das Qualitätsprofil bzw. der Service Level innerhalb eines gestrichelten Rechtecks zu nennen. Das Rechteck ist mit der Komponente bzw. der Beziehung zwischen der Komponente und einer Schnittstelle zu verbinden (vgl. Abb. 4.58).

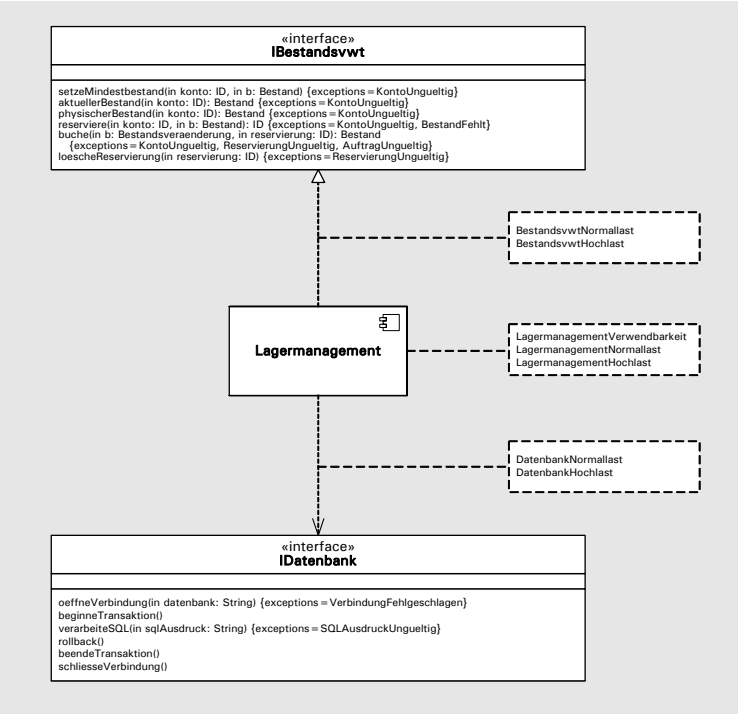


Abb. 4.58: Spezifikation der Verweise auf Qualitätsprofile und Service Level in UNSCOM/G.

Anzumerken bleibt abschließend, dass ein Vertrag auch *unabhängig* von einem Profil definiert werden darf, um ihn in mehreren Profilen verwenden zu können [FROLUND und KOISTINEN 1998:16]. In diesem Fall ist zunächst ein *Name* für den Vertrag zu vergeben. Anschließend ist eine Qualitätskategorie und, hinter dem Schlüsselwort `contract`, eine Reihe von Merkmalsausprägungen in geschweiften Klammern zuzuweisen. In einem Profil kann dann auf den definierten Vertrag verwiesen werden, indem hinter dem Schlüsselwort `require` der Vertragsname genannt wird. Die profilunabhängige Definition von Verträgen ist in Abb. 4.59 zusammen mit der *Verfeinerung* von Profilen überblicksartig dargestellt.

Abb. 4.59: Formatelemente zur Darstellung der Qualitätseigenschaften in UNSCOM/T und UNSCOM/G.

4.6.3 Geltung

Die mit den Grey Pages zu beschreibenden qualitativen Komponenteneigenschaften werden erstmals während des Systementwurfs festgelegt, in dessen Rahmen die Komponente und ihre äußere Architektur definiert werden. Mit der Definition der Systemarchitektur werden die qualitativen Vorgaben für das Gesamtsystem aus dem Pflichtenheft in Vorgaben für die einzelnen Komponenten transformiert. Durch eine (analytische oder simulationsbasierte) *Architekturevaluation* lässt sich dann überprüfen, ob die qualitativen Vorgaben für die einzelnen Komponenten ausreichen, um die Vorgaben für das Gesamtsystem noch zu erfüllen [CLEMENTS, et al. 2003:12; BECKER, et al. 2004:170f.]. In Abhängigkeit von den bei der Entwicklung zu verfolgenden Qualitätszielen werden im Systementwurf dabei ggf. unterschiedliche Komponentenmerkmale untersucht und beschrieben.

Mit Fertigstellung des Systementwurfs stellen die beschriebenen Qualitätseigenschaften zunächst Vorgaben dar, die einen bei der Realisierung der Komponente einzuhaltenden Soll-Zustand definieren. Nach Vollendung der Implementierung und während der Stabilisierungsphase ist deshalb zu untersuchen, ob die jeweils realisierte Komponente den festgelegten Qualitätseigenschaften auch tatsächlich entspricht. Die Einhaltung der Qualitätsvorgaben ist bereits während der Realisierung durch Maßnahmen der *Qualitätssicherung* sicherzustellen und dann durch (Modul- und Integrations-) *Tests* zu überprüfen, die im Anschluss an die Implementierung durchzuführen sind. Die Durchführung dieser Tests unterscheidet sich abhängig davon, ob eine logische oder physische Wiederverwendung der Komponente angestrebt wird. Während Tests bei einer physischen Wiederverwendung bereits auf einer Umgebung durchgeführt werden können, die dem späteren Laufzeitsystem entspricht, sind bei einer logischen Wiederverwendung zahlreiche verschiedene Laufzeitsysteme zu untersuchen. Nur auf diese Weise lässt sich sicherstellen, dass eine logisch wieder zu verwendende Komponente als Kompositionseinheit auch in möglichst vielen Anwendungskontexten eingesetzt werden kann.

Die bei den verschiedenen Tests ermittelten Qualitätseigenschaften gehen in die Dokumentation der Komponente ein und beschreiben den Ist-Zustand, der bei der Implementierung erreicht wurde. Der dokumentierte Ist-Zustand wird bspw. bei der Wiederverwendung der Komponente im Rahmen späterer Entwicklungsprojekte ausgewertet, um die qualitative Eignung der Komponente zu ermitteln. Die qualitative Eignung stellt neben der fachlichen Eignung eines der wichtigsten Auswahlkriterien für eine wieder zu verwendende Komponente dar und wird aus diesem Grund auch im Rahmen von (teil-) *automatisierten Auswahlverfahren* ausführlich untersucht [MILI, et al. 1995:552; KONTIO 1996].

4.7 Entwicklungsprozess

Die verschiedenen Sichten auf eine Komponente, die zuvor als Bestandteile des UNSCOM Spezifikationsrahmens eingeführt wurden, werden beim Durchlaufen des Entwicklungsprozesses im Rahmen unterschiedlicher Entwicklungsphasen sukzessive beschrieben und ausgewertet. Um die Entstehung und Verwendung der einzelnen Teile von Komponentenspezifikationen während des Entwicklungsprozesses im Gesamtzusammenhang darzustellen, wird im Folgenden näher auf die *Entwicklung des Verkaufssystems* eingegangen, die im Rahmen einer Fallstudie (vgl. Abschnitt 2.4) exemplarisch durchgeführt wurde. Als Vorgehensmodell für die Entwicklung wurde dabei auf das zuvor beschriebene *vereinheitlichte Vorgehensmodell* der komponentenorientierten Anwendungsentwicklung zurückgegriffen, das sich auf andere Vorgehensmodelle übertragen lässt (vgl. Abschnitt 2.2).

Mit dem Vorgehensmodell lassen sich die verschiedenen Vorgehensweisen bei der Entwicklung komponentenorientierter *Anwendungssysteme* gleichermaßen unterstützen. Eine idealtypische *Top-Down Vorgehensweise*, in der ausgehend von den Anforderungen und zunächst ohne Beachtung wieder verwendbarer Komponenten eine systematische Strukturierung durchgeführt wird, ist dabei vor allem für Projekte typisch, bei denen die optimale Modularisierung des Anwendungssystems im Mittelpunkt steht [CHEESMAN und DANIELS 2001:2]. Eine idealtypische *Bottom-Up Vorgehensweise*, bei der zugunsten der Wiederverwendung ggf. sogar auf die Realisierung von Anforderungen verzichtet wird, wird dagegen in Projekten angewendet, die eine möglichst umfangreiche Wiederverwendung von existierenden Komponenten anstreben [SAMETINGER 1997:186]. In der Praxis wird häufig eine *wiederverwendungsorientierte Top-Down Vorgehensweise* eingeschlagen, bei der existierende Komponenten während der Strukturierung berücksichtigt werden. Eine Kompromittierung von Anforderungen zugunsten der Wiederverwendung wird jedoch üblicherweise nicht oder nur in eingeschränktem Maße zugelassen [SAMETINGER 1997:186].

Auch bei der Entwicklung des Verkaufssystems wurde ein solcher Mittelweg eingeschlagen. Im Folgenden wird jedoch auf die beiden idealtypischen Vorgehensweisen an den Stellen näher eingegangen, an denen sich Unterschiede ergeben. Schließlich lässt sich mit dem vereinheitlichten Vorgehensmodell neben der Anwendungsentwicklung auch die *Komponentenentwicklung* für den anonymen Markt unterstützen, bei der entweder die Realisierung einer einzelnen Komponente oder die Schaffung eines Komponentensystems angestrebt wird, mit dem sich ein ganzer Funktionsbereich abdecken lässt. Prinzipiell ähnelt die Vorgehensweise bei der Komponentenentwicklung der der Anwendungsentwicklung, wobei eine idealtypische Top-Down Strategie zum Einsatz kommt (vgl. Abschnitt 2.2.2).

Die Beschreibung des Entwicklungsprozesses lässt sich deshalb auch auf ein Komponentenentwicklungsprojekt beziehen, dessen Ziel die Bereitstellung geeigneter Komponenten für die Bestellabwicklung ist.

4.7.1 Voruntersuchung

Im Rahmen der Voruntersuchung werden zunächst die *Anforderungen* zusammengetragen, die an das zu entwickelnde Anwendungs- bzw. Komponentensystem zu stellen sind (vgl. Abschnitt 2.2.2). Dazu gehören neben der Systemvision, die das allgemeine Entwicklungsziel formuliert, vor allem die grundlegenden funktionalen und qualitativen Anforderungen, die vom System zu erfüllen sind. Im Falle des zu entwickelnden Anwendungssystems zur Abwicklung von Bestellungen, die fachlich korrekt als Kunden- bzw. Abnehmeraufträge bezeichnet werden, lautet die *Systemvision* bspw. wie folgt:

> Es wird ein Anwendungssystem zur Abwicklung von Kundenaufträgen benötigt, das die Eingabe neuer Aufträge, die Prüfung und Stornierung von Aufträgen sowie deren logistische und buchhalterische Abwicklung unterstützt. Mit dem System soll es Kunden möglich sein, die Verfügbarkeit von Verkaufsartikeln abzufragen und den aktuellen Status einer Auftragsabwicklung einzusehen. Kundenaufträge sollen von jedem autorisierten Kunden eingegeben werden können. Das System soll über das World Wide Web zu benutzen sein und bei Bedarf in die Anwendungssysteme der Kunden integriert werden können. Die Zielzeit zur Eingabe eines Kundenauftrags soll dabei zwei Minuten nicht überschreiten. Um dieses Ziel zu unterstützen, soll das System die Speicherung der zur Abwicklung notwendigen Kundendaten erlauben und diese dem Kunden zur Verfügung stellen. Bei der Abwicklung eines Kundenauftrags soll das System außerdem die notwendigen Unterlagen (Lieferschein, Rechnung etc.) erstellen und den zuständigen Mitarbeitern des Unternehmens zur Verfügung stellen.

Die Systemvision lautet ähnlich, wenn ein Komponentensystem für den anonymen Markt herzustellen ist, das die Abwicklung von Kundenaufträgen unterstützt: »Es werden Komponenten zur Abwicklung von Kundenaufträgen benötigt, die die Eingabe neuer Aufträge … unterstützen«. Üblicherweise werden neben dem allgemeinen Entwicklungsziel weitere Bestandteile der Systemvision festgelegt, auf die an dieser Stelle jedoch nicht im Detail eingegangen wird. So kann bspw. durch *Storyboarding* beschrieben werden, wie das System zu nutzen sein wird und sich seine Einführung auf die Unternehmensorganisation auswirkt. Zum Vorgehen bei der Formulierung der Systemvision vgl. [HODGSON 1999].

Im Anschluss an die Systemvision werden die grundlegenden *funktionalen und qualitativen Anforderungen* an das System ermittelt und formuliert [CHEESMAN und DANIELS 2001:77-82]. Abb. 4.60 zeigt dabei exemplarisch einige der Anforderungen, die für die Dienste zur Verwaltung von Kundendaten und Kundenaufträgen festgelegt wurden.

Dienst	Kundenkonto erstellen
Initiator	Kunde
Ablauf	1. Kunde eröffnet ein Kundenkonto
	2. Kunde gibt seine persönlichen Anmeldedaten ein
	3. System speichert alle Daten
Qualität	Antwortzeit < 2 Sekunden
	Durchsatz > 5 Aufrufe/Sekunde
Dienst	Kundenkonto Daten abfragen
Initiator	Kunde
Ablauf	1. Kunde fragt seine Daten ab
	2. System stellt die für den Kunden einsehbaren Kundendaten zur Verfügung
Qualität	Antwortzeit < 1.5 Sekunden
	Durchsatz > 20 Aufrufe/Sekunde
Dienst	Kundenkonto Änderungen speichern
Initiator	Kunde
Ablauf	1. Kunde ändert Daten
	2. System speichert die geänderten Daten
Qualität	Antwortzeit < 2 Sekunden
	Durchsatz > 5 Aufrufe/Sekunde
Dienst	Kundenkonto löschen
Initiator	Kunde
Ablauf	1. Kunde initiiert Löschung des Kontos
	2. System prüft die Löschbarkeit und führt sie ggf. durch
Qualität	Antwortzeit < 3 Sekunden
	Durchsatz > 5 Aufrufe/Sekunde
Dienst	Artikel Verfügbarkeit prüfen
Initiator	Kunde
Ablauf	1. Kunde fragt Verfügbarkeit eines Artikels ab
	2. System erstellt eine Ja/Nein Antwort
Extras	2a. System gibt die Menge der Artikel zurück, falls ein bestimmter Bestand unterschritten wird
Qualität	Antwortzeit < 1 Sekunde
	Durchsatz > 50 Aufrufe/Sekunde
Dienst	Kundenauftrag erstellen
Initiator	Kunde
Ablauf	1. Kunde gibt Artikel und Bestellmenge an
	2. Kunde gibt Liefer- und Rechnungsanschrift an - System stellt Kundendaten zur Verfügung
	3. Kunde gibt Zahlweise an - System stellt Kundendaten zur Verfügung
	4. System prüft und speichert den Auftrag
Qualität	Antwortzeit < 3 Sekunden
	Durchsatz > 10 Aufrufe/Sekunde
Dienst	Kundenauftrag Daten abfragen
Initiator	Kunde
Ablauf	1. Kunde fragt Daten eines Auftrags ab
	2. System stellt alle Auftragsdaten zur Verfügung
Qualität	Antwortzeit < 1.5 Sekunden
	Durchsatz > 20 Aufrufe/Sekunde
Dienst	Kundenauftrag Status abfragen
Initiator	Kunde
Ablauf	1. Kunde fragt Status eines Auftrags ab
	2. System erstellt eine Statusinformation
Qualität	Antwortzeit < 1.5 Sekunden
	Durchsatz > 30 Aufrufe/Sekunde
Dienst	Kundenauftrag stornieren
Initiator	Kunde
Ablauf	1. Kunde storniert einen Auftrag
	2. System prüft die Stornierbarkeit und führt sie ggf. durch
Qualität	Antwortzeit < 3 Sekunden
	Durchsatz > 10 Aufrufe/Sekunde

Abb. 4.60: Anforderungen an die Dienste zur Verwaltung von Kundendaten und Kundenaufträgen.

Neben den eben dargestellten Anforderungen sind noch weitere Anforderungen festzulegen, von denen bspw. die Struktur der Dokumente beschrieben wird, die bei der Abwicklung eines Kundenauftrags zu generieren sind. Auf eine über die dargestellten Ergebnisse hinausgehende detaillierte Beschreibung der Voruntersuchung und der zur Durchführung einzusetzenden Methoden wird wegen der angestrebten Konzentration auf den Spezifikationsprozess jedoch verzichtet. Statt dessen wird auf die umfangreiche Standardliteratur zum *Requirements Engineering* verwiesen, in der die Durchführung der Voruntersuchung ausführlich beschrieben wird [DAVIS 1990; SOMMERVILLE 1992; PRESSMAN 1997].

Die ermittelten Anforderungen stellen die Basis für die Durchführung von Machbarkeits- und Wirtschaftlichkeitsstudien dar und gehen zusammen mit einem groben Projektplan in das *Pflichtenheft* ein. Das Pflichtenheft bildet im Rahmen der durchgeführten Fallstudie die Grundlage für die Durchführung des Fachentwurfs. Alternativ wäre es auf Basis des Pflichtenhefts jedoch auch möglich, eine *Fremdentwicklung* durch Dritte durchführen zu lassen (vgl. Abschnitt 2.2.2).

4.7.2 Fachentwurf

Im Fachentwurf wird die fachliche Funktionalität des zu entwickelnden Anwendungs- bzw. Komponentensystems im Detail bestimmt. Hierzu wird das der Informationsverarbeitung zugrunde liegende fachliche Handeln im Anwendungsbereich rekonstruiert und als *Begriffssystem* (bzw. Business Concept Model [CHEESMAN und DANIELS 2001:68-70] oder Domain Model [D'SOUZA und WILLS 1999:528]) spezifiziert. Anders als bei einem strukturierten Entwurf, bei dem die Betrachtung und Zerlegung von Funktionen im Mittelpunkt steht [WIRTH 1971:226], und bei einem objekt- bzw. modulorientierten Entwurf, bei dem primär eine Modellierung der Informationsobjekte durchgeführt wird [MEYER 1997:116, 684], werden die *Informationsobjekte*, *Funktionen* und *Prozesse* des Anwendungsbereichs dabei gleichermaßen in die Betrachtung einbezogen. Zwar werden die Informationsobjekte, Funktionen und Prozesse auch bei der Modellierung des fachlichen Handels im Fachentwurf zunächst getrennt voneinander betrachtet. Da dies jedoch vor allem aus Gründen der Komplexitätsreduktion geschieht und sie darüber hinaus als gleichberechtigte Spezifikationsobjekte angesehen werden, wird beim Fachentwurf noch kein spezielles Entwurfsparadigma zugrunde gelegt [BECKER und SCHÜTTE 2004:84f.]. Der Fachentwurf wird in der Literatur deshalb auch als *methodenneutral* bezeichnet [ORTNER 1997:20f.].

Bei der Rekonstruktion des Begriffssystems sind neben Entwicklern üblicherweise auch Anwender und Experten aus dem jeweiligen Anwendungsbereich im Rahmen eines Joint

Application Design beteiligt [D'SOUZA und WILLS 1999:287]. Zur Rekonstruktion der Informationsobjekte, Funktionen oder Prozesse können verschiedene Methoden des *Detailed Requirements Engineering* zum Einsatz kommen (vgl. [DAVIS 1990; DAVIS 1993; JACKSON 1995]). Zu nennen sind dabei insbesondere *Befragungen*, *Fachliteratur*, *Beobachtungen*, *Dokumentenanalyse* und *Tutorials* [DAVIS 1993:192ff.; ORTNER 1997:56f.]. Vielfach geht die Rekonstruktion des Begriffssystems dabei über das Erfassen des Ist-Zustands hinaus, so dass mit der Entwicklung und Einführung des Anwendungs- bzw. Komponentensystems auch das fachliche Handeln in einem Unternehmen (z.B. im Sinne einer Geschäftsprozessreorganisation) effizienter gestaltet werden kann.

Das wesentliche Ergebnis des Fachentwurfs stellt das spezifizierte Begriffssystem dar, das aus einem *Lexikon* und einer *Sammlung von Aussagen* besteht. Die Aussagensammlung beschreibt dabei die bestehenden Beziehungen und Zusammenhänge zwischen den einzelnen Begriffen, während im Lexikon die Bedeutung der einzelnen Begriffe festgelegt ist. Die Bedeutung kann durch eine Definition oder anhand von Beispielen konkretisiert werden und dient als *Erkennungsregel*, mit der sich eine Ausprägung (z.B. eine bestimmte Person „Dominik Plonner“) als Instanz eines Begriffs (z.B. Kunde) verstehen lässt. Für das zu entwickelnde Auftragsabwicklungssystem sind im Rahmen des Fachentwurfs verschiedene Organisationsbereiche zu untersuchen, die den *betrieblichen Distributionsprozess* beeinflussen. Dabei stehen vor allem die Analyse der *Kunden-, Artikel- und Lagerorganisation* sowie des *Verkaufs*, der *Fakturierung* und der *Debitorenbuchhaltung* im Mittelpunkt der Entwurfsarbeiten [BECKER und SCHÜTTE 2004:396f.].

Aufgrund der Komplexität des zu untersuchenden Anwendungsbereichs entsteht während des Fachentwurfs ein umfangreiches Begriffssystem, das an dieser Stelle schon aus Gründen der Anschaulichkeit nur unvollständig (und zum Teil erheblich vereinfacht) vorgestellt werden kann. Einige Auszüge aus dem *Lexikon*, das während des Fachentwurfs erstellt wurde, sind jedoch in den folgenden Abbildungen beispielhaft dargestellt. In Abb. 4.61 wird dabei zunächst die Bedeutung von *Informationsobjekten* beschrieben, die während des Fachentwurfs als selbstständige *Begriffe erster Ordnung* eingestuft wurden. Im Lexikon ist zusätzlich auch die Bedeutung der Informationsobjekte zu beschreiben, die als *Begriffe zweiter Ordnung* anderen Begriffen in Form von Attributen zukommen. Zwei dieser Informationsobjekte sind im unteren Teil der Abbildung exemplarisch aufgeführt. Neben der Bedeutung von Informationsobjekten beschreibt das Lexikon auch die Bedeutungen der *Funktionen* und *Prozesse*, die der Informationsverarbeitung zugrunde liegen. Einige der im Fachentwurf untersuchten Funktionen und Prozesse sind deshalb in Abb. 4.62 zur weiteren Vervollständigung der Darstellung aufgeführt.

Terminologie (Typ: Lexikon)
Begriff (Typ: Informationsobjekt)
Begriffswort: Kunde
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kunde ist eine natürliche oder juristische Person, die Güter nachfragt.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kundenauftrag
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kundenauftrag ist eine juristisch bindende Bestellung von Gütern durch einen Kunden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Artikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Artikel ist ein wirtschaftliches Gut.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lagerartikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Lagerartikel ist ein Artikel, der vom Handelsunternehmen gelagert wird.
Begriff (Typ: Informationsobjekt)
Begriffswort: Verkaufsartikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Verkaufsartikel ist ein Artikel, der vom Handelsunternehmen verkauft wird.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kundenauftragsposition
Genus: (Typ: Femininum)
Kurzdefinition: Eine Kundenauftragsposition legt die Menge eines bestellten Artikels fest.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lager
Genus: (Typ: Neutrum)
Kurzdefinition: Ein Lager ist ein Ort, an dem Artikel aufbewahrt werden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lagerbereich
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Lagerbereich ist ein Ort, an dem Artikel zu einem bestimmten Zweck aufbewahrt werden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kommissionierbereich
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kommissionierbereich ist ein Lagerbereich, an dem Artikel auftragsspezifisch zusammengestellt werden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Preis
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Preis ist ein Geldbetrag, der für den Erhalt eines Artikels (zuzüglich Umsatzsteuer) zu entrichten ist.
Begriff (Typ: Informationsobjekt)
Begriffswort: Haltbarkeitsdatum
Genus: (Typ: Neutrum)
Kurzdefinition: Ein Haltbarkeitsdatum gibt an, wie lange ein Artikel von einem Kunden verwendet werden kann.

Abb. 4.61: Beschreibung der Bedeutung von Informationsobjekten (Lexikon).

Terminologie (Typ: Lexikon)
Begriff (Typ: Funktion)
Begriffswort: Bestand reservieren
Kurzdefinition: Durch das Reservieren eines Bestands wird eine Menge von Lagerartikeln einem Kundenauftrag zugeordnet.
Begriff (Typ: Funktion)
Begriffswort: Reservierung löschen
Kurzdefinition: Durch das Löschen einer Reservierung wird eine Menge von Lagerartikeln freigegeben.
Begriff (Typ: Funktion)
Begriffswort: Kommissionierplan erstellen
Kurzdefinition: Durch das Erstellen eines Kommissionierplans wird die Zusammenstellung von Lagerartikeln vorgegeben.
Begriff (Typ: Funktion)
Begriffswort: Lagerartikel kommissionieren
Kurzdefinition: Durch das Kommissionieren von Lagerartikeln werden diese aus dem Lager zusammengestellt.
Begriff (Typ: Funktion)
Begriffswort: Lagerartikel erfassen
Kurzdefinition: Durch das Erfassen von Lagerartikeln wird ein Warenausgang ermittelt.
Begriff (Typ: Funktion)
Begriffswort: Bestandsveränderung ermitteln
Kurzdefinition: Durch das Ermitteln der Bestandsveränderung wird die Veränderung im Bestand eines Artikels quantifiziert.
Begriff (Typ: Funktion)
Begriffswort: Bestandsveränderung buchen
Kurzdefinition: Durch das Buchen einer Bestandsveränderung wird eine Veränderung im Bestand eines Lagerartikels erfasst.
Begriff (Typ: Funktion)
Begriffswort: Nachschubauftrag veranlassen
Kurzdefinition: Durch das Veranlassen eines Nachschubauftrags wird das Auffüllen eines Artikelbestands eingeleitet.
Begriff (Typ: Funktion)
Begriffswort: PhysischenBestand prüfen
Kurzdefinition: Durch das Prüfen eines physischen Bestands wird die vorhandene Menge eines Lagerartikels ermittelt.
Begriff (Typ: Funktion)
Begriffswort: Bestand prüfen
Kurzdefinition: Durch das Prüfen eines Bestands wird die verfügbare Menge eines Lagerartikels ermittelt.
Begriff (Typ: Prozess)
Begriffswort: Kundenauftrag abwickeln
Kurzdefinition: Das Abwickeln eines Kundenauftrags umfasst alle Tätigkeiten zur Bearbeitung eines Kundenauftrags.
Begriff (Typ: Prozess)
Begriffswort: Kundenauftrag prüfen
Kurzdefinition: Das Prüfen eines Kundenauftrags umfasst alle Tätigkeiten zur Freigabe eines neu erstellten Auftrags.
Begriff (Typ: Prozess)
Begriffswort: Lagerartikel auslagern
Kurzdefinition: Das Auslagern von Lagerartikeln umfasst alle logistischen Aktivitäten zur Bearbeitung eines Kundenauftrags.
Begriff (Typ: Prozess)
Begriffswort: Warenausgang buchen
Kurzdefinition: Das Buchen eines Warenausgangs umfasst alle Aktivitäten zur Verarbeitung einer Bestandsveränderung.

Abb. 4.62: Beschreibung der Bedeutung von Funktionen und Prozessen (Lexikon).

Ergänzend zum Lexikon sind in den folgenden Abbildungen einige Teile der Aussagensammlung aufgeführt, die im Rahmen der Fallstudie rekonstruiert wurde.

Ein Lieferantenauftrag ist ein Auftrag.
Ein Kundenauftrag ist ein Auftrag.
Ein Auftrag ist ein Kundenauftrag oder ein Lieferantenauftrag.
Ein Kundenauftrag besteht aus ein bis beliebig vielen Kundenauftragsposition.
Ein Kundenauftrag hat eine Auftragsnummer und ein Auftragsdatum und eine Lieferanschrift und eine Rechnungsanschrift und einen Auftragsstatus.
Eine Auftragsnummer ist eine Zeichenkette.
Ein Auftragsdatum ist ein Datum.
Eine Lieferanschrift ist zusammengesetzt aus einem Name und einer Straße und einer Hausnummer und einer Postleitzahl und einem Ort.
Ein Name ist eine Zeichenkette.
Eine Straße ist eine Zeichenkette.
Eine Hausnummer ist eine Zeichenkette.
Eine Postleitzahl ist eine Zeichenkette.
Ein Ort ist eine Zeichenkette.
Eine Rechnungsanschrift ist zusammengesetzt aus einem Name und einer Straße und einer Hausnummer und einer Postleitzahl und einem Ort.
Eine Kundenauftragsposition verknüpft einen Kunde und einen Verkaufsartikel.
Eine Kundenauftragsposition hat eine Artikelmenge.
Eine Artikelmenge ist eine Ganzzahl.
Ein Auftragsstatus ist aktiv oder geprüft oder kommissioniert oder versandt oder storniert.

Ein Kunde hat eine Kundennummer und einen Ansprechpartner und eine Adresse und 0 bis 1 Bankverbindung und 0 bis beliebig viele Kreditkarte.
Eine Kundennummer ist eine Zeichenkette.
Ein Ansprechpartner ist zusammengesetzt aus 0 bis 1 Titel und 0 bis 1 Vorname und einem Nachname und einer Telefonnummer und einer Emailadresse.
Ein Titel ist Doktor oder Professor Doktor.
Ein Vorname ist eine Zeichenkette.
Ein Nachname ist eine Zeichenkette.
Eine Telefonnummer ist eine Zeichenkette.
Eine Emailadresse ist eine Zeichenkette.
Eine Adresse ist zusammengesetzt aus einer Straße und einer Hausnummer und einer Postleitzahl und einem Ort.
Eine Bankverbindung ist zusammengesetzt aus einer Kontonummer und einer Bankleitzahl.
Eine Kontonummer ist eine Zeichenkette.
Eine Bankleitzahl ist eine Zeichenkette.
Eine Kreditkarte ist zusammengesetzt aus einem Kartentyp und einer Kartennummer und einem Gültigkeitsmonat und einem Gültigkeitsjahr.
Ein Kartentyp ist VISA oder Mastercard oder American Express.
Eine Kartennummer ist eine Zeichenkette.
Ein Gültigkeitsmonat ist eine Zeichenkette.
Ein Gültigkeitsjahr ist eine Zeichenkette.

Ein Lagerartikel ist ein Artikel.
Ein Verkaufsartikel ist ein Artikel.
Ein Artikel ist ein Lagerartikel oder ein Verkaufsartikel.
Ein Artikel hat 1 bis beliebig viele Artikelnummer und einen Name und eine Warengruppe und einen Anlagezeitpunkt und einen Änderungszeitpunkt und eine Steuerklasse und 0 bis 1 Ursprungsland und 0 bis 1 Ursprungsregion.
Eine Warengruppe ist eine Zeichenkette.
Eine Artikelnummer ist eine Zeichenkette.
Ein Anlagezeitpunkt ist ein Datum.
Ein Änderungszeitpunkt ist ein Datum.
Eine Steuerklasse ist voll oder ermäßigt.
Ein Ursprungsland ist eine Zeichenkette.
Eine Ursprungsregion ist eine Zeichenkette.
Ein Lagerartikel hat eine Länge und eine Breite und eine Höhe und 0 bis 1 Volumen und 0 bis 1 Bruttogewicht und 0 bis 1 Nettogewicht und eine Lagereinheit und 1 bis beliebig viele Lagerplatzort.
Eine Länge ist eine Ganzzahl und wird gemessen in Zentimeter.
Ein Volumen ist eine Ganzzahl und wird gemessen in Milliliter.
Ein Bruttogewicht ist eine Ganzzahl und wird gemessen in Gramm.
Ein Nettogewicht ist eine Ganzzahl und wird gemessen in Gramm.
Eine Lagereinheit ist Europalette oder Einzelartikel.

Abb. 4.63: Beschreibung der Zusammenhänge zwischen Informationsobjekten (Aussagensammlung).

Ein Bestandskonto hat eine Artikelnummer und eine Kontonummer und einen Bestand und einen PhysischerBestand und einen Mindestbestand.
Eine Kontonummer ist eine Ganzzahl.
Ein Bestand ist eine Ganzzahl.
Ein Mindestbestand ist eine Ganzzahl.
Ein PhysischerBestand ist eine Ganzzahl.
Ein Bestandskonto besteht aus 0 bis beliebig vielen Bestandsveränderung und 0 bis beliebig vielen Reservierung.
Eine Bestandsveränderung hat einen Änderungstyp und einen Auftrag und eine Artikelmenge.
Ein Änderungstyp ist Wareneingang oder Warenausgang.
Eine Reservierung hat eine Artikelmenge.

Ein Verkaufsartikel hat einen Verkaufsbeginn und ein Verkaufsende und eine Verfügbarkeit und 1 bis beliebig viele Verkaufsbedingung und einen Bontext und 0 bis 1 Regaltext.
Ein Verkaufsbeginn ist ein Datum.
Ein Verkaufsende ist ein Datum.
Eine Verfügbarkeit ist lieferbar oder vergriffen.
Eine Verkaufsbedingung ist zusammengesetzt aus einem Preis und einer Vertriebsschiene und einer Verkaufseinheit.
Ein Preis ist eine Fließkommazahl und wird gemessen in Euro.
Eine Vertriebsschiene ist Einzelartikel oder Artikelgruppe.
Eine Verkaufseinheit ist eine Ganzzahl.
Ein Bontext ist eine Zeichenkette.
Ein Regaltext ist eine Zeichenkette.

Ein Lager besteht aus 0 bis beliebig vielen Lagerbereich.
Ein Lager hat einen Name.
Ein Kommissionierbereich ist ein Lagerbereich.
Ein Reservebereich ist ein Lagerbereich.
Ein Wareneingangsbereich ist ein Lagerbereich.
Ein Warenausgangsbereich ist ein Lagerbereich.
Ein Lagerbereich ist ein Kommissionierbereich oder ein Reservebereich oder ein Wareneingangsbereich oder ein Warenausgangsbereich.
Ein Lagerbereich hat einen Name und eine Lagerstrategie.
Ein Lagerbereich besteht aus 0 bis beliebig vielen Lagerplatz.
Ein Name ist eine Zeichenkette.
Eine Lagerstrategie ist festplatzorientiert oder chaotisch.
Ein Lagerplatz hat einen Lagerplatzort und einen Lagerplatztyp und einen Lagerplatzbestand.
Ein Lagerplatzort ist zusammengesetzt aus einer Gangnummer und einer Regalnummer und einer Ebenennummer.
Ein Lagerplatztyp ist zusammengesetzt aus einer Dimension und einer Tragfähigkeit und einer Lagereinheit und einer Maximalbelegung.
Eine Gangnummer ist eine Zeichenkette.
Eine Regalnummer ist eine Zeichenkette.
Eine Ebenennummer ist eine Zeichenkette.
Eine Dimension ist zusammengesetzt aus einer Breite und einer Höhe und einer Tiefe.
Eine Breite ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Höhe ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Tiefe ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Tragfähigkeit ist eine Ganzzahl und wird gemessen in Gramm.
Eine Maximalbelegung ist eine Ganzzahl.
Ein Lagerplatzbestand ist zusammengesetzt aus 0 bis beliebig vielen Artikelbestand.
Ein Artikelbestand ist zusammengesetzt aus einer Artikelnummer und einem Bestand.

Ein Einlagerungsplan besteht aus 0 bis beliebig vielen Planposition.
Ein Kommissionierplan besteht aus 0 bis beliebig vielen Planposition.
Eine Planposition verknüpft einen Auftrag und einen Lagerartikel und einen Lagerplatz.
Eine Planposition hat eine Artikelmenge.
Ein Einlagerungsplan hat ein Erstellungsdatum.
Ein Kommissionierplan hat ein Erstellungsdatum.
Ein Erstellungsdatum ist ein Datum.

Ein Wareneingang hat ein Ausführungsdatum und 0 bis beliebig viele Lagerposition.
Ein Warenausgang hat ein Ausführungsdatum 0 bis beliebig viele Lagerposition.
Eine Lagerposition ist zusammengesetzt aus einer Artikelnummer und einer Artikelmenge.
Ein Ausführungsdatum ist ein Datum.

Abb. 4.64: Beschreibung der Zusammenhänge zwischen Informationsobjekten (Aussagensammlung, fortgesetzt).

Kundenauftrag abwickeln besteht aus Kundenauftrag erteilen und Bestand reservieren und Kundenauftrag prüfen und Lagerartikel auslagern und Kundenauftrag fakturieren und Kundenzahlung überwachen und Kundenauftrag stornieren und Reservierung löschen.

Ein Kunde tut einen Kundenauftrag erteilen, falls für alle Verkaufsartikel des Kundenauftrag gilt, dass der Lieferstatus lieferbar ist.
Ein Kunde tut einen Kundenauftrag stornieren.
Ein Sachbearbeiter tut einen Kundenauftrag prüfen.
Ein Sachbearbeiter tut einen Bestand reservieren mit einem Kundenauftrag zu einer Reservierung.
Ein Sachbearbeiter tut eine Reservierung löschen.
Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel auslagern mit einem Kundenauftrag.
Ein Sachbearbeiter tut einen Kundenauftrag fakturieren zu einer Rechnung.
Ein Sachbearbeiter tut einen Kundenauftrag überwachen.

Kundenauftrag prüfen besteht aus Bonität prüfen und Auftragsposition prüfen.
Ein Sachbearbeiter tut eine Bonität prüfen mit einem Kundenauftrag.
Ein Sachbearbeiter tut einen bis beliebig viele Auftragsposition prüfen mit einem Kundenauftrag.
Kundenauftrag prüfen beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Bonität prüfen bei Bonität prüfen ist durchzuführen.
Nach Bonität prüfen folgt Auftragsposition prüfen bei Bonität prüfen ist abgeschlossen.
Nach Bonität prüfen folgt Ablauf beenden mit Kundenauftrag prüfen ist abgeschlossen.

Lagerartikel auslagern besteht aus Kommissionierplan erstellen und Lagerartikel kommissionieren und Lagerartikel erfassen und Warenausgang buchen.
Ein Lagerverwalter tut einen Kommissionierplan erstellen mit einem Kundenauftrag.
Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel kommissionieren mit einem Kommissionierplan.
Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel erfassen mit einem Kommissionierplan zu einem Warenausgang.
Ein Lagerverwalter tut einen Warenausgang buchen.
Lagerartikel auslagern beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Kommissionierplan erstellen bei Kommissionierplan erstellen ist durchzuführen.
Nach Kommissionierplan erstellen folgt Lagerartikel kommissionieren bei Kommissionierplan erstellen ist abgeschlossen.
Nach Lagerartikel kommissionieren folgt Lagerartikel erfassen bei Lagerartikel kommissionieren ist abgeschlossen.
Nach Lagerartikel erfassen folgt Warenausgang buchen bei Lagerartikel erfassen ist abgeschlossen.
Nach Warenausgang buchen erfolgt Ablauf beenden mit Lagerartikel auslagern ist abgeschlossen.

Warenausgang buchen besteht aus Bestandsveränderung ermitteln und Bestandsveränderung buchen und Nachschubauftrag erstellen.
Ein Lagerverwalter tut 1 bis beliebig viele Bestandsveränderung ermitteln mit einem Warenausgang.
Ein Lagerverwalter tut eine Bestandsveränderung buchen mit einer Reservierung zu einem Bestand.
Ein Lagerverwalter tut einen Nachschubauftrag veranlassen.
Warenausgang buchen beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Bestandsveränderung ermitteln bei Bestandsveränderung ermitteln ist durchzuführen.
Nach Bestandsveränderung ermitteln folgt Bestandsveränderung buchen bei Bestandsveränderung ermitteln ist abgeschlossen.
Nach Bestandsveränderung buchen folgt Alternativablauf starten.
Nach Alternativablauf starten folgt entweder Nachschubauftrag veranlassen bei Bestand ist kleiner oder gleich Mindestbestand oder Nichts tun bei Bestand ist größer als Mindestbestand.
Nach Nachschubauftrag veranlassen oder Nichts tun folgt Alternativablauf beenden.
Nach Alternativablauf beenden folgt Ablauf beenden mit Warenausgang buchen ist abgeschlossen.

Kundenauftrag abwickeln beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Kundenauftrag erteilen bei Kundenauftrag erteilen ist durchzuführen.
Nach Kundenauftrag erteilen folgt Kundenauftrag prüfen bei Kundenauftrag erteilen ist abgeschlossen.
Nach Kundenauftrag prüfen erfolgt Bestand reservieren bei Kundenauftrag prüfen ist abgeschlossen.
Nach Bestand reservieren folgt Alternativablauf starten.
Nach Alternativablauf starten folgt entweder Kundenauftrag stornieren bei Kundenauftrag stornieren ist durchzuführen oder Lagerartikel auslagern bei Bestand reservieren ist abgeschlossen.
Nach Kundenauftrag stornieren erfolgt Reservierung löschen bei Kundenauftrag stornieren ist abgeschlossen.
Nach Lagerartikel auslagern folgt Kundenauftrag fakturieren bei Lagerartikel auslagern ist abgeschlossen.
Nach Kundenauftrag fakturieren folgt Kundenzahlung überwachen bei Kundenauftrag fakturieren ist abgeschlossen.
Nach Reservierung löschen oder Kundenzahlung überwachen folgt Alternativablauf beenden.
Nach Alternativablauf beenden folgt Ablauf beenden mit Kundenauftrag abwickeln ist abgeschlossen.

Abb. 4.65: Beschreibung der Zusammenhänge zwischen Funktionen und Prozessen (Aussagensammlung).

Die in den Abbildungen dargestellten Aussagen beschreiben die festgehaltenen Zusammenhänge zwischen den Informationsobjekten (Abb. 4.63 und Abb. 4.64) sowie zwischen den Funktionen und Prozessen des Anwendungsbereichs (Abb. 4.65) auszugsweise. Zur Darstellung der Aussagensammlung wird dabei auf die textbasierte *Normsprache* zurückgegriffen, die in Abschnitt 4.4.2 vorgestellt wurde. Diese Form der textbasierten Darstellung ist besonders für die am Fachentwurf mitarbeitenden Anwender und Fachexperten leichter verständlich als die Diagrammsprachen der UML bzw. Referenzmodellierung. Unter Verwendung der definierten Übersetzungsmöglichkeit von UNSCOM/T nach UNSCOM/G lässt sich die textbasierte Darstellung bei Bedarf jedoch auch in ein äquivalentes graphisches Diagramm überführen. Eine graphische Repräsentation wird vor allem von den ebenfalls am Fachentwurf zu beteiligenden Systemanalytikern bevorzugt. Unter Rückgriff auf die graphische Darstellung lässt sich außerdem die Abstimmung mit ggf. vorhandenen fachlichen *Referenzmodellen* erleichtern, die in den Fachentwurf einfließen können. So wurde bei der Entwicklung des Auftragsabwicklungssystems bspw. auf das *Handels-H-Modell* [BECKER und SCHÜTTE 2004] zurückgegriffen, das umfangreiche fachliche Vorarbeiten für die Entwicklung von Handelsinformationssystemen beinhaltet.

Das spezifizierte Begriffssystem geht zusammen mit den zu erarbeitenden fachlichen Testfällen in das *Fachkonzept* ein, das die Grundlage für die weiteren Entwicklungsarbeiten bildet. Das Fachkonzept beschreibt die zu realisierende fachliche Funktionalität im Detail und ergänzt somit das Pflichtenheft um weitere Vorgaben, die bei der Entwicklung einzuhalten sind. Auf der Basis des Fachentwurfs lässt sich untersuchen, ob die spezifizierten Anforderungen von einer *Standard-Software* erfüllt werden können. Hierzu wären im Rahmen der Fallstudie bspw. entsprechende Produkte von SAP oder aus der Reihe der Microsoft Business Solutions zu untersuchen. Soll keine Standard-Software genutzt werden, ist der Entwicklungsprozess mit dem Systementwurf (bzw. Domänenentwurf) fortzusetzen.

4.7.3 Systementwurf (Domänenentwurf)

Während des Systementwurfs (bzw. Domänenentwurfs) wird die technische Gestaltung des zu entwickelnden Anwendungs- bzw. Komponentensystems festgelegt. Hierzu gehört die Entscheidung zugunsten einer *Technologie* für die Realisierung ebenso wie die *Strukturierung* des Systems in Komponenten (Software-Architektur) und die Festlegung der technischen *Infrastruktur* (Hardware-Architektur). Im Rahmen der Fallstudie wurde die Microsoft .NET Plattform zur Realisierung des Verkaufssystems gewählt, da diese die Bereitstellung der kundenbezogenen Dienste über das World Wide Web (bspw. über eine Weboberfläche oder einen XML Web-Service) speziell unterstützt und deshalb der eingangs formu-

lierten Systemvision besonders entspricht. Im Anschluss an die Entscheidung zugunsten der Implementierungsplattform ist die Software-Architektur des Anwendungs- bzw. Komponentensystems festzulegen. Dabei erfolgt eine *Strukturierung des Systems* in Komponenten [CHEESMAN und DANIELS 2001:84]. Darüber hinaus sind die für das Gesamtsystem relevanten *Eigenschaften der Komponentenaußensichten* festzulegen und zu spezifizieren [CHEESMAN und DANIELS 2001:84].

4.7.3.1 Strukturierung in Komponenten

Zur Strukturierung des Systems wird üblicherweise ein *schichtenorientiertes Vorgehen* gewählt, das die drei Schichten Benutzungsoberfläche, Anwendungs-Software und System-Software unterscheidet [CHEESMAN und DANIELS 2001:84f.; SIEDERSLEBEN 2004:88-90]. Da die Komponenten einer Schicht dabei lediglich mit Komponenten derselben Schicht oder denen der untergeordneten Schichten interagieren dürfen, kann die Strukturierung des Anwendungs- bzw. Komponentensystems weitgehend schichtspezifisch und somit *schrittweise* vorgenommen werden [CHEESMAN und DANIELS 2001:85]. Während der Strukturierung sind zunächst die einzelnen Komponenten des Systems zu identifizieren [CHEESMAN und DANIELS 2001:83-101].

Entscheidend für die Güte einer Software-Architektur ist, dass bei der Strukturierung solche Komponenten gefunden werden, die dem Ideal der *minimalen Abhängigkeiten bei gleichzeitig maximaler Kohäsion* entsprechen (vgl. Abschnitt 2.1.4). Die Minimierung der Abhängigkeiten trägt zur Eigenständigkeit der Komponenten bei, während die Maximierung der Kohäsion (also das Zusammenfassen von fachlich zusammengehörigen Funktionen) für die angestrebte Abgeschlossenheit einer Komponente unter fachlichen Gesichtspunkten sorgt. Der Prozess der Komponentenfindung wird durch verschiedene Methoden unterstützt, die auf die rekonstruierten Informationsobjekte, Funktionen und Prozesse des Fachentwurfs zurückgreifen. So lassen sich durch eine gegenüberstellende Betrachtung von Funktionen und bearbeiteten Informationsobjekten bspw. *Funktionscluster* identifizieren, die zur Komponentenfindung genutzt werden können [HERZUM und SIMS 2000:446; ALBANI, et al. 2003]. Durch eine ergänzende Analyse der Prozesse lassen sich eigenständige Anwendungsteile identifizieren, die jeweils einen *Teilablauf* im Anwendungsbereich vollständig unterstützen [HERZUM und SIMS 2000:446f.].

Auf eine über diese Anmerkungen hinausgehende Beschreibung des Vorgehens bei der Identifikation von Komponenten wird wegen der hier angestrebten spezifikationsorientierten Betrachtung verzichtet und stattdessen wiederum auf die zitierte Literatur verwiesen. Ausdrücklich anzumerken bleibt jedoch sowohl für die Anwendungsentwicklung als auch für die Komponentenentwicklung, dass während der Strukturierung häufig solche Kompo-

nenten zu berücksichtigen sind, die bereits existieren. Bei einer Anwendungsentwicklung kann es sich dabei bspw. um *Legacy-Komponenten* handeln, die bereits produktiv eingesetzt werden, während bei der Komponentenentwicklung unter anderem solche Komponenten zu berücksichtigen sind, die vom Hersteller bereits *zuvor* auf den Markt gebracht wurden. Auch im Rahmen der Fallstudie ist mit der Komponente Artikelmanagement eine existierende Komponente zu integrieren, die die Artikelorganisation des Unternehmens realisiert. Sie verwaltet neben den für den Verkauf relevanten Artikelstammdaten (Verkaufsartikel) auch die für die Lagerhaltung benötigten Daten (Lagerartikel) und spielt bei der Abwicklung von Kundenaufträgen somit eine zentrale Rolle.

Im Rahmen einer *wiederverwendungsorientierten Anwendungsentwicklung* ist während der Strukturierung außerdem nach weiteren Komponenten zu suchen, die Teile der bereitzustellenden Funktionalität abdecken. Zur *Suche* nach entsprechenden Komponenten können unternehmensinterne Repositorien und die Kataloge globaler Komponentenmarktplätze verwendet werden. Dabei wird in einem ersten Schritt zunächst eine noch unbewertete Liste mit existierenden Komponenten zusammengetragen, wobei üblicherweise anhand einiger Stichworte wie bspw. „Microsoft .NET“ und „Lagermanagement“ in den verschiedenen Katalogen gesucht wird. Anschließend ist die Eignung der gefundenen Komponenten anhand von funktionalen, qualitativen sowie anderen (z.B. kaufmännischen) Kriterien zu untersuchen und eine Reihung der Kandidaten vorzunehmen. Dieser Prozess lässt sich durch Methoden der rationalen *Entscheidungsunterstützung* (zur Übersicht vgl. [RUHE 2002]) sowie durch automatisierte *Kompatibilitätstests* verbessern, die idealerweise auf typtheoretischen Kalkülen basieren (vgl. Abschnitt 3.2).

Bei der Entwicklung des UNSCOM Spezifikationsrahmens wurde darauf geachtet, dass mit den Yellow Pages sowie den White, Blue, Green und Grey Pages jeweils *adäquate Informationen* bereitgestellt werden, die zur Stichwortsuche sowie zur anschließenden statischen Eignungsprüfung verwendet werden können. Insbesondere lassen sich für die einzelnen Aspekte der Blue, Green und Grey Pages ontologie- [NOY 2004a; NOY 2004b], signatur- [YELLIN und STROM 1994], protokoll- [NIERSTRASZ 1995; YELLIN und STROM 1997] und qualitätsmerkmalsbasierte [FROLUND und KOISTINEN 1998:38-54] Kalküle im Rahmen statischer Kompatibilitätstests verwenden. Eine ausführliche Diskussion der Auswirkungen, die sich bei der Entwicklung durch die Einbeziehung von wieder zu verwendenden Komponenten ergeben, findet sich bei [WALLNAU, et al. 2002].

Ausgehend von den identifizierten und den ggf. einzubindenden Komponenten ist die Software-Architektur mit den verschiedenen Sichten auf das zu entwickelnde System zu beschreiben. Abb. 4.66 zeigt als Auszug aus der Struktursicht den *Bauplan*, der für den

anwendungsspezifischen Teil des Auftragsabwicklungssystems erstellt wurde. Als weitere Bestandteile der Software-Architektur sind zudem der Ablauf (Kontrollsicht), die Verteilung der Komponenten auf Rechnerknoten (Verteilungssicht) sowie die nach außen sichtbaren Eigenschaften der Komponenten zu beschreiben. Diese Eigenschaften liegen bei den existierenden Komponenten bereits fest und sind folglich unter Rückgriff auf die jeweilige Komponentendokumentation in die Software-Architektur zu *integrieren*. Die Integration gestaltet sich dabei umso schwieriger, je mehr der zur Dokumentation eingesetzte Spezifikationsrahmen von dem des Entwicklungsprojekts abweicht. Zudem können bei der Integration *Heterogenitäten* (Verschiedenheiten) zwischen den Komponenten auftreten, die durch einen (ebenfalls im Systementwurf zu spezifizierenden) *Adapter* zu beseitigen sind.

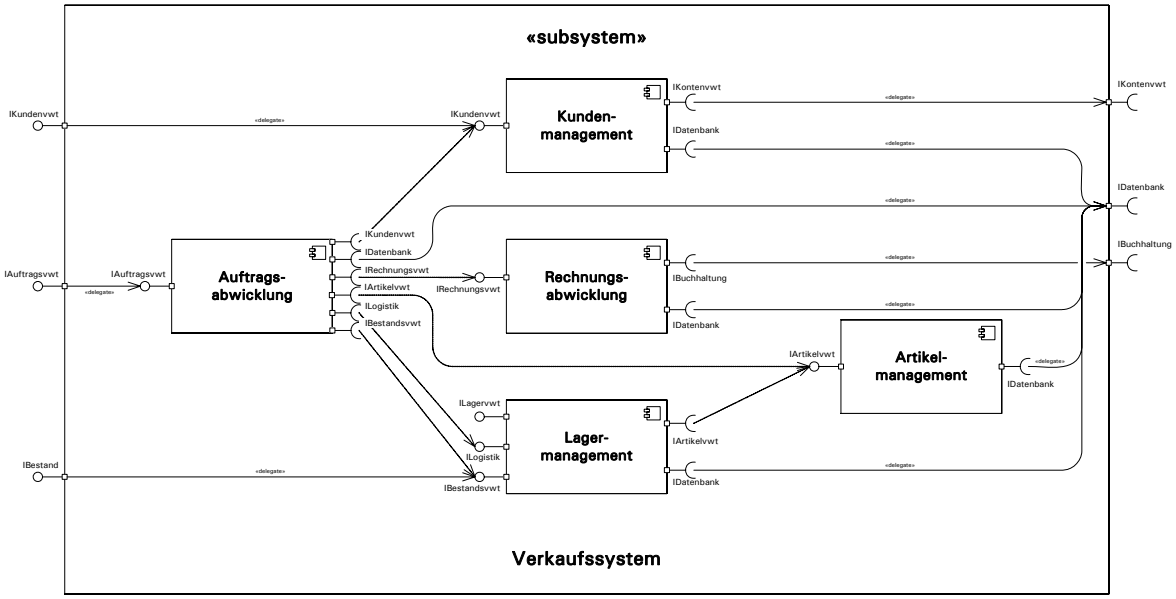


Abb. 4.66: Bauplan für den anwendungsspezifischen Teil des Auftragsabwicklungssystems.

4.7.3.2 Spezifikation der Komponentenaußensicht

Die nach außen sichtbaren Eigenschaften der neu zu entwickelnden Komponenten sind während des Systementwurfs explizit festzulegen. Die Festlegung der Komponenteneigenschaften erfolgt in drei Schritten. In einem ersten Schritt wird der Ausschnitt aus dem Begriffssystem des Fachkonzepts ermittelt, der die von der Komponente zu erbringende *Funktionalität* beschreibt [CHEESMAN und DANIELS 2001:88-90]. Dabei wird zwischen funktionalen Aspekten unterschieden, die nach außen sichtbar sind, und Aspekten, die sich auf die komponenteninterne Realisierung beziehen. Während die nach außen sichtbaren Aspekte beim Systementwurf als Komponentenfunktionalität im Rahmen der Blue Pages in die Spezifikation aufgenommen werden, werden die komponenteninternen Aspekte des Begriffssystems erst beim Komponentenentwurf verwendet. In einem zweiten Schritt werden die *architekturspezifischen Komponenteneigenschaften* festgelegt und als Green Pages in

die Spezifikation aufgenommen. Dabei werden ausgehend von der zu realisierenden Funktionalität zunächst die Schnittstellen der Komponente bestimmt und danach die zugehörigen Methoden, Zusicherungen und Protokolle beschrieben [CHEESMAN und DANIELS 2001:86-96]. Im dritten Schritt werden schließlich die *qualitativen Eigenschaften* der Komponente festgelegt und als Grey Pages in die Spezifikation aufgenommen.

Für die neu zu entwickelnde Komponente Lagermanagement, die im Folgenden beispielhaft betrachtet wird, sind vor allem die beiden Prozesse »Lagerartikel auslagern« und »Warenausgang buchen« sowie die mit diesen Prozessen in Zusammenhang stehenden Funktionen und Informationsobjekte als relevante, nach außen sichtbare funktionale Aspekte zu identifizieren und in die Blue Pages zu übernehmen (vgl. Anhang A). Als sog. Unterstützungsprozess ist zudem der (hier nicht beschriebene) Prozess »Lagerstammdaten verwalten« in die Blue Pages aufzunehmen. Auf der Basis der identifizierten Funktionalität können anschließend die drei Angebotsschnittstellen ILogistik, IBestandsvwt und ILagervwt identifiziert werden, die eine jeweils eigenständige Funktionalität realisieren (vgl. Abb. 4.66). Die Schnittstelle ILogistik bietet die benötigten lagerlogistischen Dienste zur Auslagerung von Lagerartikeln an, die Schnittstelle IBestandsvwt stellt die Dienste zur Bestandsführung bereit und die Schnittstelle ILagervwt beinhaltet schließlich die Dienste, die zur unterstützenden Lagerorganisation benötigt werden. Um die gewünschte Funktionalität zu erbringen, benötigt die Komponente Lagermanagement zudem die Dienste einer Datenbank sowie der Artikelorganisation, die jeweils von anderen Komponenten erbracht werden. Ergänzend zu den Angebotsschnittstellen sind der Komponente deshalb die beiden Nachfrageschnittstellen IDatenbank und IArtikelvwt hinzuzufügen (vgl. Abb. 4.66).

Im Anschluss an die Identifikation der Schnittstellen sind deren jeweilige architekturspezifische Eigenschaften (Signaturen, Zusicherungen sowie Protokolle) festzulegen und im Rahmen der Green Pages zu dokumentieren. Um die Komponente Lagermanagement möglichst wieder verwendbar zu realisieren, sind dabei bereits im Fachentwurf auch Dienste zur Einlagerung von Lagerartikeln zu berücksichtigen. Diese sind im Folgenden auszugsweise dargestellt, werden hier jedoch nicht im Detail betrachtet. Explizit hinzuweisen ist weiterhin auf die Tatsache, dass nicht für alle im Fachentwurf rekonstruierten Funktionen auch unterstützende Methoden an den Komponentenschnittstellen bereitzustellen sind. So sind die Funktionen »Lagerartikel kommissionieren«, »Bestandsveränderung ermitteln« und »Nachschubauftrag veranlassen« vom Lagerverwalter ohne software-seitige Unterstützung (also in einem nicht vollautomatisierten Lagerbetrieb) auszuführen. Zur Veranlassung eines Nachschubauftrags wird durch die Aussendung des Ereignisses MindestbestandUnterschritten jedoch zumindest beigetragen. Abb. 4.67 und Abb. 4.68 zeigen die *Signaturen*, die für die Komponente Lagermanagement im Rahmen eines UML

Diagramms (gemäß den Vorgaben des UNSCOM Metaschemas) festgelegt und in die Green Pages aufgenommen wurden. Auf eine explizite Ordnung der Elemente im Diagramm wird dabei wegen der besseren Übersichtlichkeit verzichtet. Die Ordnung der Elemente wird stattdessen in einer separaten *Liste* angegeben, in der die Elemente nacheinander mit ihren vollständigen Namen aufgeführt werden (vgl. Abschnitt 4.5.1.1).

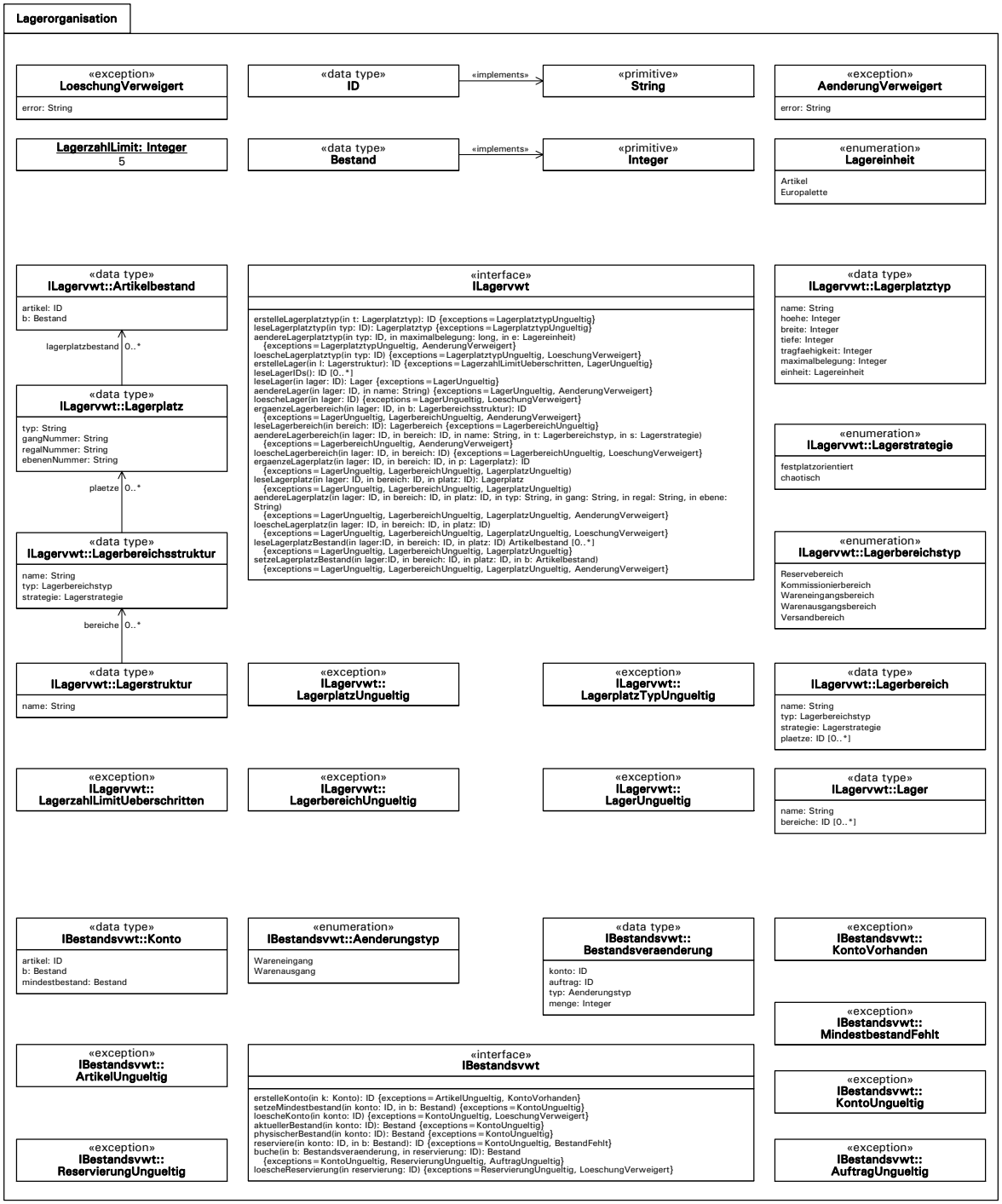


Abb. 4.67: Signaturdefinitionen für die Komponente Lagermanagement.

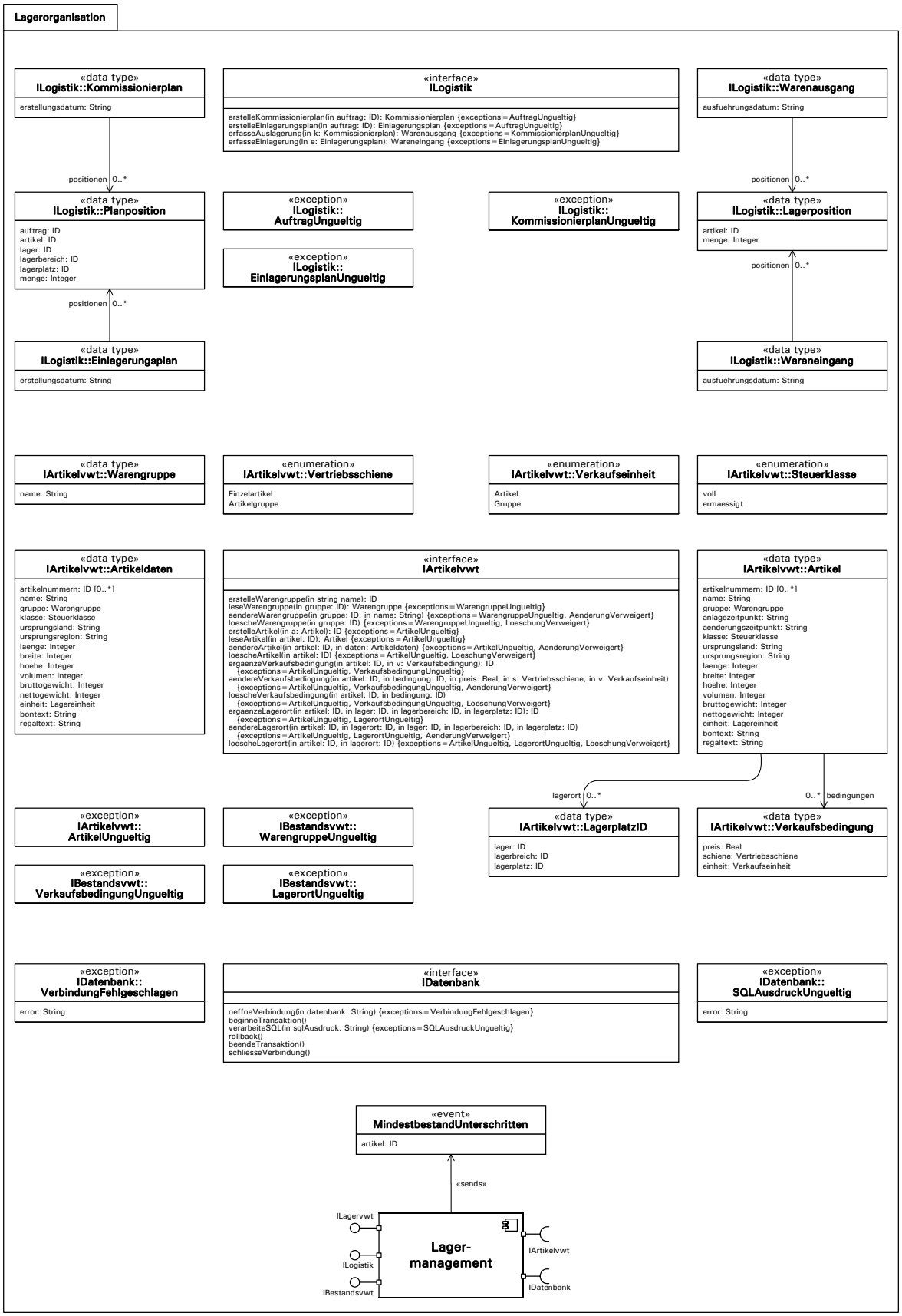


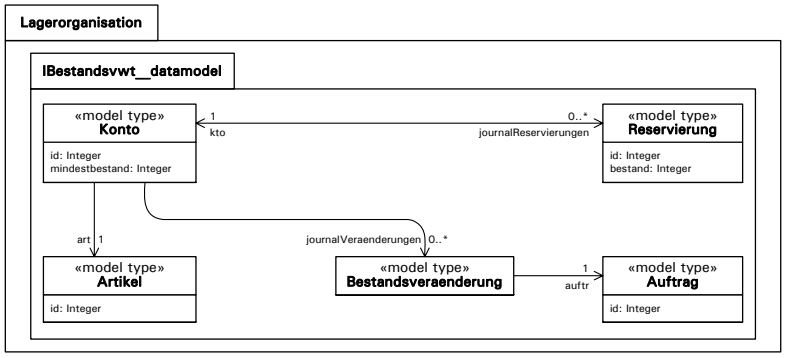
Abb. 4.68: Signaturdefinitionen für die Komponente Lagermanagement (fortgesetzt).

Die Komponentenschnittstellen wurden so entworfen, dass zwischen Komponenten nur einfache Datentypen und keine Objektreferenzen übertragen werden [CHEESMAN und DANIELS 2001:57f.; SIEDERSLEBEN 2004:184-186]. Durch das umgesetzte Prinzip der *wertbasierten Schnittstellensemantik* wird zum einen verhindert, dass das zur Realisierung gewählte *Entwicklungsparadigma* an den Komponentenschnittstellen sichtbar wird. Dies ist notwendig, damit Komponenten auch dann noch miteinander interagieren können, wenn sie unter Rückgriff auf verschiedene Entwicklungsparadigmen realisiert wurden (zum Vergleich von Entwicklungsparadigmen vgl. Abschnitt 2.1). Zum anderen besitzen Objekte schon aufgrund ihrer feinen Granularität üblicherweise eine Vielzahl von *Abhängigkeiten* auf andere Objekte (vgl. Abschnitt 2.1.3). Diese Abhängigkeiten würden bei der Übergabe auf die jeweils andere Komponente übertragen und müssten zumindest durch ein Klassendiagramm explizit modelliert werden, um eine vollständige Beschreibung der Schnittstellen zu erhalten. Trotz der genannten Nachteile kann es in einem rein objektorientierten Anwendungskontext jedoch vorteilhaft sein, Komponentenschnittstellen mit einer *referenzbasierten Semantik* zu entwerfen. Der UNSCOM Spezifikationsrahmen unterstützt aus diesem Grund beide Semantiken in gleicher Weise.

Im Anschluss an die Signaturen sind die *Dienstverträge* für die Methoden der Angebots- und Nachfrageschnittstellen festzulegen und formal zu beschreiben. Die Festlegung der Dienstverträge erfolgt mit der Definition von Zusicherungen (Vor- und Nachbedingungen), die sich unter Bezugnahme auf die zuvor spezifizierten Signaturelemente formulieren lassen. Darüber hinaus kann es ggf. notwendig sein, auch auf weitere Elemente Bezug zu nehmen, die dann im Rahmen eines *Datenmodells* zusammenzufassen und zu spezifizieren sind (vgl. Abschnitt 4.5.1.2). Für jede Schnittstelle ist dabei ein eigenes Datenmodell zu erstellen, das jeweils nicht mehr Informationen enthalten soll, als zur Definition der Zusicherungen benötigt werden [CHEESMAN und DANIELS 2001:123]. Abb. 4.69 und Abb. 4.70 zeigen die Vor- und Nachbedingungen, die im Rahmen der Fallstudie für die Angebotsschnittstelle IBestandsvwt zu definieren sind.

Um die von einigen Methoden vorausgesetzte bzw. garantierte (Nicht-) Existenz von bestimmten Konten, Artikeln, Aufträgen und Reservierungen formulieren zu können, ist zunächst das in Abb. 4.69 gezeigte Datenmodell zu definieren. Es zeigt *fiktive* komponenteninterne Realisierungen der eben genannten Informationsobjekte, deren Ausprägungen sich zur Laufzeit über ein (Schlüssel-) Attribut id identifizieren lassen. Die Attribute der Realisierungen sind dabei möglichst zu reduzieren, um den Einblick in die Komponenteninnensicht zu minimieren. Unter Rückgriff auf das Datenmodell lassen sich im Rahmen von Zusicherungen dann Aussagen über die Existenz von bestimmten Ausprägungen der spezifizierten Informationsobjekte treffen. Die entsprechenden Vor- und Nachbedingungen kön-

nen zur Laufzeit jedoch schon wegen ihres Bezugs auf fiktive Objekte nicht mehr überprüft werden und haben somit lediglich einen rein beschreibenden Charakter.




```
package Lagerorganisation

context IBestandsvwt::erstelleKonto(k: Konto): ID
  pre : k.artikel <> '' and k.b >= 0 and k.mindestbestand >= 0
  -- Die Kontodaten muessen valide sein
  pre : not IBestandsvwt__datamodel::Konto.allInstances()->exists(konto |
          konto.art.id = k.artikel)
  -- Es darf noch kein Konto fuer den angegebenen Artikel existieren
  post: IBestandsvwt__datamodel::Konto.allInstances()->exists(konto |
          konto.id = result and konto.art.id = k.artikel and
          konto.mindestbestand = k.mindestbestand) and
        IBestandsvwt::physischerBestand(result) = k.b
  -- Ein Konto mit der zurueckgelieferten ID und den angegebenen
     Daten wurde angelegt
  post: IBestandsvwt::aktuellerBestand(result) =
        IBestandsvwt::pysischerBestand(result)
  -- Der frei verfuegbare Bestand entspricht dem physischen Bestand

context IBestandsvwt::setzeMindestbestand(konto: ID, b: Bestand)
  pre : b >= 0
  -- Der Betrag des Mindestbestands darf nicht kleiner als Null sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: IBestandsvwt__datamodel::Konto.allInstances()->exists(k |
          k.id = konto and k.mindestbestand = b)
  -- Der Mindestbestand wurde eingetragen

context IBestandsvwt::loescheKonto(konto: ID)
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: not IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das Konto wurde geloescht

context IBestandsvwt::aktuellerBestand(konto: ID): Bestand
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: result >= 0
  -- Der frei verfuegbare Bestand eines Artikels ist nicht kleiner als Null

context IBestandsvwt::physischerBestand(konto: ID): Bestand
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: result >= 0
  -- Der physische Bestand eines Artikels ist nicht kleiner als Null

context IBestandsvwt::reserviere(konto: ID, b: Bestand): ID
  pre : b > 0
  -- Der Betrag der Reservierung muss groesser als Null sein
  pre : b <= IBestandsvwt::aktuellerBestand(konto)
  -- Der Betrag der Reservierung darf nicht groesser als der frei verfuegbare
     Bestand sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
```

Abb. 4.69: Definition von Vor- und Nachbedingungen für die Schnittstelle IBestandsvwt.

```
context IBestandsvwt::reserviere(konto: ID, b: Bestand): ID
  post: IBestandsvwt::aktuellerBestand(konto) =
        IBestandsvwt::aktuellerBestand@pre(konto) - b
  -- Nach der Eintragung der Reservierung wurde der frei verfuegbare Bestand
     des Artikels um den genannten Betrag vermindert
  post: IBestandsvwt::aktuellerBestand(konto) >= 0
  -- Nach der Eintragung der Reservierung ist der frei verfuegbare Bestand
     nicht kleiner als Null
  post: IBestandsvwt__datamodel::Reservierung.allInstances()->exists(reservierung |
          reservierung.id = result and reservierung.bestand = b)
  -- Eine Reservierung mit der zurueckgelieferten ID und der angegebenen
     Menge wurde angelegt
  post: IBestandsvwt::physischerBestand(konto) =
        IBestandsvwt::physischerBestand@pre(konto)
  -- Der physische Bestand des Artikels bleibt unveraendert

context IBestandsvwt::buche(b: Bestandsveraenderung, reservierung: ID): Bestand
  pre : b.menge > 0
  -- Der Betrag der Bestandsveraenderung muss groesser als Null sein
  pre : if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            b.menge <= IBestandsvwt::physischerBestand(b.konto)
          else
            true
        endif
  -- Bei einem Warenausgang darf der Betrag der Bestandsveraenderung nicht
     groesser als der physische Bestand des Artikels sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = b.konto)
  -- Das angegebene Konto muss existieren
  pre : if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
            r.id = reservierung)
          else
          reservierung = ''
        endif
  -- Bei einem Warenausgang muss die angegebene Reservierung existieren
     und bei einem Wareneingang darf keine Reservierung angegeben werden
  post: result = if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
                   then
                     IBestandsvwt::physischerBestand@pre(b.konto) - b.menge
                   else
                     IBestandsvwt::physischerBestand@pre(b.konto) + b.menge
                 endif
  -- Nach der Buchung der Bestandsveraenderung wurde der physische Bestand des
     Artikels um den genannten Betrag vermindert bzw. erhoeht
  post: result >= 0
  -- Nach der Buchung der Bestandsveraenderung ist der physische Bestand nicht
     kleiner als Null
  post: if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            IBestandsvwt::aktuellerBestand(b.konto) =
            IBestandsvwt::aktuellerBestand@pre(b.konto)
          else
            IBestandsvwt::aktuellerBestand(b.konto) =
            IBestandsvwt::aktuellerBestand@pre(b.konto) + b.menge
        endif
  -- Der frei verfuegbare Bestand des Artikels bleibt bei einem Warenausgang
     unveraendert und wird bei einem Wareneingang um den genannten Betrag erhoeht

context IBestandsvwt::loescheReservierung(reservierung: ID)
  def : konto : IBestandsvwt::ID =
        (IBestandsvwt__datamodel::Reservierung.allInstances()->any(r |
          r.id = reservierung)).kto.id
  def : menge : IBestandsvwt::Bestand =
        (IBestandsvwt__datamodel::Reservierung.allInstances()->any(r |
          r.id = reservierung)).bestand
  pre : IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
        r.id = reservierung)
  -- Die angegebene Reservierung muss existieren
  post: not IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
        r.id = reservierung)
  -- Die Reservierung wurde geloescht
  post: aktuellerBestand(konto) = aktuellerBestand@pre(konto) + menge
  -- Der reservierte Bestand wurde dem frei verfuegbaren Bestand hinzugefuegt

endpackage
```

Abb. 4.70: Definition von Vor- und Nachbedingungen für die Schnittstelle IBestandsvwt (fortgesetzt).

Die im Rahmen der Zusicherungen formulierten Existenzbedingungen beschreiben bereits einen Teil der *Reihenfolgebeziehungen*, die zwischen den verschiedenen Methoden existieren. So besagt die Vorbedingung »Das bei einer Bestandsabfrage angegebene Konto muss existieren« der Methode aktuellerBestand bspw., dass das abgefragte Konto zuvor (unter Verwendung der Methode erstelleKonto) bereits angelegt worden sein muss. Da sich Reihenfolgebeziehungen dieser Art grundsätzlich nicht mit den zur Protokollspezifikation verwendeten deterministischen endlichen Automaten beschreiben lassen, sind sie bei der Festlegung der *Interaktionsprotokolle* stattdessen als Bestandteil der Dienstverträge zu formulieren (vgl. Abschnitt 4.5.1.3). Dadurch vereinfacht sich die Beschreibung derjenigen Protokollteile, die durch Zustandsautomaten darzustellen sind.

Abb. 4.71 zeigt beispielhaft die *Angebotsprotokolle* der beiden Angebotsschnittstellen ILogistik und IBestandsvwt sowie die Dienstbedarfsprotokolle, die für die dort angebotenen Dienste festzulegen sind. Für die Dienstbedarfsprotokolle sind dabei jeweils *Übergangswahrscheinlichkeiten* zu schätzen, falls von einem Zustand mehrere Übergänge ausgehen. Die Übergangswahrscheinlichkeiten geben die Wahrscheinlichkeit an, mit der ein Übergang bei einer Interaktion ausgeführt wird (vgl. Abschnitt 4.5.1.3). Sie werden vor allem zur Vorhersage der qualitativen Systemeigenschaften genutzt, die im Rahmen der später durchzuführenden Architekturevaluation ermittelt werden.

Spätestens bei der Modellierung der Dienstbedarfsprotokolle wird auch ersichtlich, dass die Komponente Lagermanagement lediglich den Dienst leseArtikel der Komponente Artikelmanagement nachfragt. Zudem werden nicht sämtliche der zurück gelieferten Daten, sondern lediglich die lagerbezogenen Artikelstammdaten und die Lagerplatzzuordnung benötigt. Um die Wiederverwendbarkeit der Komponente Lagermanagement zu erhöhen, empfiehlt sich daher eine entsprechende Reduktion der Nachfrageschnittstelle IArtikelvwt. Die daraus resultierende Nachfrageschnittstelle IArtikelvwt' lässt sich jedoch nicht mehr direkt mit der Schnittstelle IArtikelvwt verbinden, die von der Komponente Artikelmanagement angeboten wird. Vielmehr liegt eine signaturspezifische Abweichung (ein sog. *Signature Mismatch* [ZAREMSKI und WING 1995]) vor, der durch einen Adapter zu beheben ist. Die durch die Reduktion entstehende Schnittstelle IArtikelvwt' ist in Anhang A dargestellt.

Unter Bezugnahme auf die modellierten Dienstbedarfsprotokolle lässt sich schließlich auch der *parametrisierte Komponentenvertrag* für die Komponente Lagermanagement angeben, der in Abb. 4.72 dargestellt ist. Durch den Vertrag werden die Dienstbedarfsprotokolle in Beziehung zu den angebotenen Diensten der Komponente gesetzt, so dass die existierenden Abhängigkeiten dienstspezifisch beschrieben und später ermittelt werden können.

Zu den Auswertungsmöglichkeiten von Komponentenverträgen vgl. Abschnitt 3.3.2 und 3.3.3.

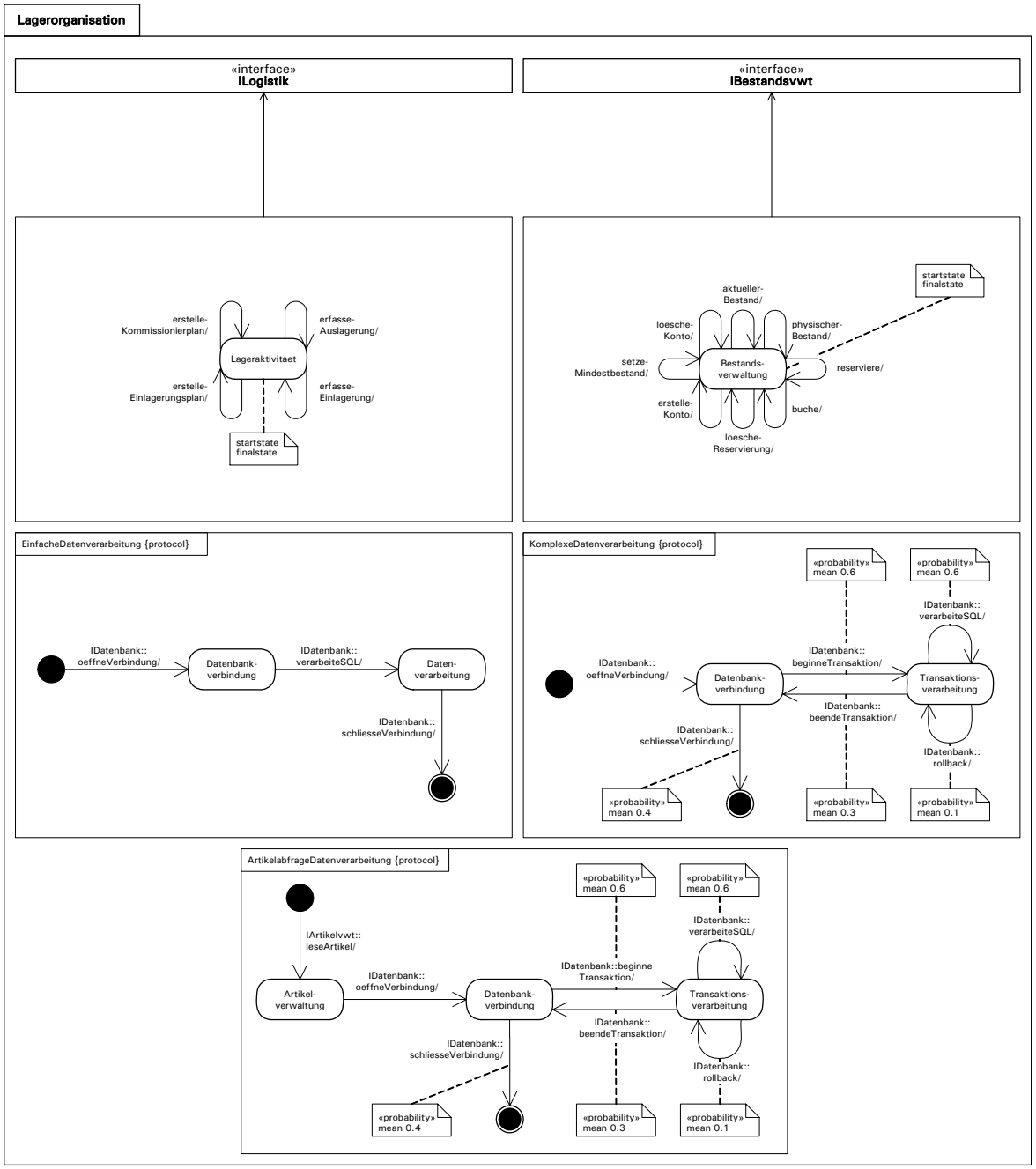


Abb. 4.71: Definition von Angebots- und Dienstbedarfsprotokollen für die Komponente Lagermanagement.

Die Beschreibung des Komponentenvertrags vervollständigt die Spezifikation der architekturspezifischen Komponenteneigenschaften und geht – analog zu den anderen Spezifikationsteilen – in die Green Pages der Komponentenspezifikation ein. Die festgelegten Eigenschaften bilden dabei Entwurfsvorgaben, die im Rahmen des Komponentenentwurfs und

der anschließenden Implementierung einzuhalten sind. Sie schränken somit den Spielraum des Komponentenentwicklers ein.

```
package Lagerorganisation

context Lagermanagement
  ILagervwt::erstelleLagerplatztyp     requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerplatztyp         requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerplatztyp      requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerplatztyp      requires KomplexeDatenverarbeitung
  ILagervwt::erstelleLager             requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerIDs              requires EinfacheDatenverarbeitung
  ILagervwt::leseLager                 requires EinfacheDatenverarbeitung
  ILagervwt::aendereLager              requires KomplexeDatenverarbeitung
  ILagervwt::loescheLager              requires KomplexeDatenverarbeitung
  ILagervwt::ergaenzeLagerbereich      requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerbereich          requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerbereich       requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerbereich       requires KomplexeDatenverarbeitung
  ILagervwt::ergaenzeLagerplatz        requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerplatz            requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerplatz         requires KomplexeDatenverarbeitung
  ILagervwt::leseLagerplatzbestand     requires EinfacheDatenverarbeitung
  ILagervwt::setzeLagerplatzbestand    requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerplatz         requires KomplexeDatenverarbeitung
  ILogistik::erstelleKommissionierplan requires KomplexeDatenverarbeitung
  ILogistik::erstelleEinlagerungsplan  requires ArtikelabfrageDatenverarbeitung
  ILogistik::erfasseAuslagerung        requires KomplexeDatenverarbeitung
  ILogistik::erfasseEinlagerung        requires KomplexeDatenverarbeitung
  IBestandsvwt::erstelleKonto          requires KomplexeDatenverarbeitung
  IBestandsvwt::setzeMindestbestand    requires EinfacheDatenverarbeitung
  IBestandsvwt::loescheKonto           requires KomplexeDatenverarbeitung
  IBestandsvwt::aktuellerBestand       requires EinfacheDatenverarbeitung
  IBestandsvwt::physischerBestand      requires EinfacheDatenverarbeitung
  IBestandsvwt::reserviere             requires KomplexeDatenverarbeitung
  IBestandsvwt::buche                  requires KomplexeDatenverarbeitung
  IBestandsvwt::loescheReservierung    requires EinfacheDatenverarbeitung

endpackage
```

Abb. 4.72: Definition des parametrisierten Komponentenvertrags für die Komponente Lagermanagement.

Nachdem die architekturspezifischen Komponenteneigenschaften spezifiziert sind, ist zu überprüfen, ob die zuvor in den Blue Pages beschriebene Komponentenfunktionalität durch entsprechende Methoden abgedeckt wird. Bei der Überprüfung sind die Begriffe des Lexikons in Beziehung zu den korrespondierenden Architekturbestandteilen zu setzen, so dass ggf. vorhandene Abweichungen offenbar werden. Dabei ist für diejenigen Begriffe, für die keine entsprechenden Architekturbestandteile identifiziert werden konnten, zu untersuchen, ob die Abweichung durch eine bewusste Entwurfsentscheidung oder einen fehlerhaften Entwurf verursacht wurde. In Abb. 4.73 ist der *Entwurfszusammenhang* zwischen Begriffen und Architekturbestandteilen für die Komponente Lagermanagement dargestellt. Die dort auftretenden Abweichungen entstammen bewussten Entwurfsentscheidungen, die zuvor erläutert wurden. Die spezifizierte Komponentenarchitektur ist somit dazu geeignet, die in den Blue Pages beschriebene Komponentenfunktionalität abzudecken, und kann in einem letzten Entwurfsschritt um Qualitätseigenschaften ergänzt werden.

Begriff	Art	Architekturbestandteil	Art
Artikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Lagerartikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Verkaufsartikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Bestandskonto	Infoobjekt	Lagerorganisation::IBestandsvwt::Konto	Typ
Bestandsveränderung	Infoobjekt	Lagerorganisation::IBestandsvwt::Bestandsveraenderung	Typ
Lager	Infoobjekt	Lagerorganisation::ILagervwt::Lager,	Typ
		Lagerorganisation::ILagervwt::Lagerstruktur	Typ
Lagerbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereich,	Typ
		Lagerorganisation::ILagervwt::Lagerbereichsstruktur	Typ
Kommissionierbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Reservebereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Wareneingangsbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Warenausgangsbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Lagerplatz	Infoobjekt	Lagerorganisation::ILagervwt::Lagerplatz	Typ
Einlagerungsplan	Infoobjekt	Lagerorganisation::ILogistik::Einlagerungsplan	Typ
Kommissionierplan	Infoobjekt	Lagerorganisation::ILogistik::Kommissionierplan	Typ
Planposition	Infoobjekt	Lagerorganisation::ILogistik::Planposition	Typ
Wareneingang	Infoobjekt	Lagerorganisation::ILogistik::Wareneingang	Typ
Warenausgang	Infoobjekt	Lagerorganisation::ILogistik::Warenausgang	Typ
Bestand prüfen	Funktion	Lagerorganisation::IBestandsvwt::aktuellerBestand	Methode
PhysischenBestand prüfen	Funktion	Lagerorganisation::IBestandsvwt::physischerBestand	Methode
Bestand reservieren	Funktion	Lagerorganisation::IBestandsvwt::reserviere	Methode
Reservierung löschen	Funktion	Lagerorganisation::IBestandsvwt::loescheReservierung	Methode
Bestandsveränderung buchen	Funktion	Lagerorganisation::IBestandsvwt::buche	Methode
Lagerartikel auslagern	Prozess	Lagerorganisation::IBestandsvwt,	Protokoll
		Lagerorganisation::ILogistik	Protokoll
Kommissionierplan erstellen	Funktion	Lagerorganisation::ILogistik::erstelleKommissionierplan	Methode
Lagerartikel kommissionieren	Funktion	--	
Lagerartikel erfassen	Funktion	Lagerorganisation::ILogistik::erfasseAuslagerung	Methode
Warenausgang buchen	Prozess	Lagerorganisation::IBestandsvwt	Protokoll
Bestandsveränderung ermitteln	Funktion	--	
Nachschubauftrag veranlassen	Funktion	Lagerorganisation::MindestbestandUnterschritten	Ereignis

Abb. 4.73: Entwurfszusammenhang zwischen Fachbegriffen und Architekturbestandteilen.

Zur Spezifikation der *Qualitätseigenschaften* einer Komponente sind zunächst die Qualitätsanforderungen aus dem Pflichtenheft in Teilanforderungen zu zerlegen, die von den verschiedenen Komponenten bzw. den von ihnen angebotenen oder nachgefragten Methoden zu erfüllen sind. Diese Teilanforderungen sind im Rahmen der Grey Pages zu dokumentieren. Darüber hinaus können während des Systementwurfs zusätzliche Qualitätsanforderungen identifiziert und in die Grey Pages aufgenommen werden. Die spezifizierten Qualitätseigenschaften stellen daher im Allgemeinen eine Obermenge der Qualitätseigenschaften dar, die im Pflichtenheft aufgeführt wurden.

Zur Spezifikation von Qualitätseigenschaften kann auf den Katalog des UNSCOM Spezifikationsrahmens zurückgegriffen werden, der eine Vielzahl standardisierter *Merkmalstypen* vordefiniert. Alternativ können jedoch auch spezielle Merkmalstypen festgelegt und definiert werden. Für die aus dem Katalog übernommenen bzw. neu definierten Merkmalstypen sind anschließend *Merkmalsausprägungen* zu definieren, die die konkreten Qualitätsanforderungen beschreiben. Diese sind mit der Komponente bzw. den Diensten der Komponentenschnittstellen in Bezug zu setzen. Zusammengehörige Qualitätsanforderungen

können bei der Definition zu *Service Levels* zusammengefasst werden (vgl. Abschnitt 4.6.1).

```
type Efficiency = contract {
   responseTime : decreasing numeric s;
   throughput : increasing numeric calls/s;
   memoryUtilization : decreasing numeric b;
   discUtilization : decreasing numeric b;
};
```

```
BestandsvwtNormallast for IBestandsvwt = profile {
   from aktuellerBestand, physischerBestand require Efficiency contract {
      responseTime {mean < 0.5 s;};
      throughput {mean > 60 calls/s; percentile 80 > 50 calls/s;};
   };
   from buche, reserviere, loescheReservierung require Efficiency contract {
      responseTime {mean < 1 s;};
      throughput {mean > 20 calls/s; percentile 80 > 15 calls/s;};
   };
};

DatenbankNormallast for IDatenbank = profile {
   from oeffneVerbindung, schliesseVerbindung, beginneTransaktion, beendeTransaktion
      require Efficiency contract {
         responseTime {mean < 0.05 s;};
         throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
      };
   from verarbeiteSQL, rollback require Efficiency contract {
      responseTime {mean < 0.2 s;};
      throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
   };
};

BestandsvwtHochlast for IBestandsvwt = profile {
   from aktuellerBestand, physischerBestand require Efficiency contract {
      responseTime {mean < 0.25 s;};
      throughput {mean > 80 calls/s; percentile 80 > 70 calls/s};
   };
   from buche, reserviere, loescheReservierung require Efficiency contract {
      responseTime {mean < 0.5 s;};
      throughput {mean > 30 calls/s; percentile 80 > 25 calls/s};
   };
};

DatenbankHochlast for IDatenbank = profile {
   from oeffneVerbindung, schliesseVerbindung, beginneTransaktion, beendeTransaktion
      require Efficiency contract {
         responseTime {mean < 0.025 s;};
         throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
      };
   from verarbeiteSQL, rollback require Efficiency contract {
      responseTime {mean < 0.1 s;};
      throughput {mean > 200 calls/s; percentile 80 > 180 calls/s;};
   };
};
```

```
LagermanagementNormallast for Lagermanagement = servicelevel {
   require BestandsvwtNormallast, DatenbankNormallast;
};

LagermanagementHochlast for Lagermanagement = servicelevel {
   require BestandsvwtHochlast, DatenbankHochlast;
};
```

Abb. 4.74: Definition der Qualitätseigenschaften für die Schnittstellen IBestandsvwt und IDatenbank.

Abb. 4.74 zeigt einige der Qualitätseigenschaften, die im Systementwurf für die Komponente Lagermanagement und ihre Angebots- bzw. Nachfrageschnittstellen zu definieren sind. Die Darstellung beschränkt sich auf die Festlegung von Performanzeigenschaften, die sich aus dem Pflichtenheft ableiten lassen. Zur Definition der Qualitätseigenschaften wur-

de dabei auf den Merkmalskatalog des UNSCOM Spezifikationsrahmens zurückgegriffen, der standardisierte Merkmalstypen zur Beschreibung der Antwortzeit und des Durchsatzes bereitstellt. Die in den Grey Pages beschriebenen Qualitätseigenschaften ergänzen die funktionalen Vorgaben der Blue Pages und die architekturspezifischen Vorgaben der Green Pages. Wie die anderen Vorgaben sind auch diese bei der Realisierung der Komponente einzuhalten.

4.7.3.3 Evaluation der Software-Architektur

Parallel zu den zuvor beschriebenen Entwurfsarbeiten ist zu überprüfen, ob die entstehende Software-Architektur dazu beiträgt, dass das zu entwickelnde System den funktionalen und qualitativen Anforderungen aus dem Pflichtenheft bzw. dem Fachkonzept genügt. Hierzu sind die funktionalen und qualitativen Eigenschaften des Systems (das im Systementwurf durch seine Software-Architektur repräsentiert wird) aus den jeweiligen Komponenteneigenschaften abzuleiten. Bei der Ableitung werden die in den Green Pages und den Blue bzw. Grey Pages enthaltenen Informationen der einzelnen Komponentenspezifikationen ausgewertet und unter Rückgriff auf die Struktur-, Kontroll- sowie die Verteilungssicht zu Systemeigenschaften zusammengeführt. Die Ableitung erfolgt mit einem Vorhersagemodell, wobei entweder ein analytisches Verfahren (kompositionales Schließen) oder eine Simulation zum Einsatz kommt. Die Überprüfung der Software-Architektur, die auch als *Architekturevaluation* bezeichnet wird, lässt sich durch eine Vielzahl von Methoden unterstützen, die meist unterschiedliche Sichten der Software-Architektur auswerten (zum Überblick vgl. [BECKER, et al. 2004]). Eine ausführliche Beschreibung von Methoden und Verfahren der Architekturevaluation findet sich bei [CLEMENTS, et al. 2001].

Mit der (positiven) Evaluation der fertig gestellten Systemarchitektur ist der Systementwurf schließlich abgeschlossen. Die erstellten Komponentenspezifikationen gehen zusammen mit dem Bauplan in das *Systemkonzept* ein, das die Basis für die Entwicklung der einzelnen Komponenten und die Konfigurierung des Gesamtsystems bildet. Die einzelnen Komponentenspezifikationen sind dabei zunächst zu vervollständigen, indem *allgemeine Angaben* zur Komponente (wie bspw. der Name, eine Beschreibung und der Ansprechpartner) dokumentiert und im Rahmen der White Pages zusammengefasst werden. Abb. 4.75 zeigt beispielhaft die für die Komponente Lagermanagement erstellten allgemeinen Informationen. Falls die einzelnen Komponentenspezifikationen in einem Entwicklungsrepositorium abzulegen sind, empfiehlt sich darüber hinaus eine *Klassierung*. Anhand der Klassierung, die bspw. nach den Vorgaben der Yellow Pages durchgeführt werden kann, lässt sich die Spezifikation im Katalog ablegen und zu einem späteren Zeitpunkt wieder auffinden. Die Klassierung der Komponente Lagermanagement ist in Abb. 4.75 ebenfalls dargestellt.

Name: Oversoft Lagermanagement
Beschreibung: Oversoft Lagermanagement stellt Funktionen zur Lagerverwaltung zur Verfügung, die den Warenein- und -ausgang sowie die Bestandsverwaltung abdecken. Die Small Business Edition ist dabei für mittelständische Betriebe gedacht und erlaubt die Verwaltung von maximal fünf Lagerstätten ...
Ansprechpartner (Typ: Kontakt)
Name: Sven Overhage
Beschreibung: Software-Architekt, Anwendungsentwicklung (ae)
Adresse (Typ: Adresse)
Adresselement (Typ: Strasse): An der Steinmauer
Adresselement (Typ: Hausnummer): 2
Adresselement (Typ: Postleitzahl): 61191
Adresselement (Typ: Ort): Rosbach
Adresselement (Typ: Land): Deutschland
Kommunikationsmedium (Typ: E-Mail): sven.overhage@oversoft.biz
Kommunikationsmedium (Typ: Telefon): +49 6003 9300-16
Kommunikationsmedium (Typ: Telefax): +49 6003 9300-72

Klassierung (Typ: Klassierung)
Klassem (Typ: Klassifikationseintrag)
Facette (Typ: Technologie)
Taxonomie: (Typ: Component Model Classification System)
Wert: Microsoft .NET 1.1
Klassem (Typ: Klassifikationseintrag)
Facette (Typ: Komponentenart)
Taxonomie: (Typ: Conceptual Component Type Classification System)
Wert: Application-Specific Component
Klassem (Typ: Klassifikationseintrag)
Facette (Typ: Wiederverwendungsart)
Taxonomie: (Typ: Reuse Type Classification System)
Wert: Logical Reuse
Klassem (Typ: Klassifikationseintrag)
Facette (Typ: Anwendungsbereich)
Taxonomie: (Typ: Application Domain Classification System)
Wert: Warehousing
Klassem (Typ: Klassifikationseintrag)
Facette (Typ: Funktionsbereich)
Taxonomie: (Typ: North American Industry Classification System)
Wert: General Warehousing and Storage

Abb. 4.75: Allgemeine Informationen (oben) und Klassierung (unten) der Komponente Lagermanagement.

Hinzuweisen bleibt noch auf die Bedeutung von *Schnittstellenstandards*, die bei der Strukturierung des Systems und der Spezifikation von Komponenten genutzt werden können. Im Rahmen der Fallstudie konnten für die Verbindung zur Buchhaltung bspw. die beiden Nachfrageschnittstellen IKontenvwt und IBuchhaltung unter Nutzung des OMG General Ledger Standards definiert werden, der unter den Bezeichnungen LedgerLifecycle und BookKeeping entsprechende Standardschnittstellen vorgibt [OMG 2001:1-3]. Bemerkens-

wert ist dabei, dass auch im OMG General Ledger Standard verschiedene Aspekte der Methoden beschrieben werden, die den Aspekten des UNSCOM Spezifikationsrahmens bereits recht nahe kommen. So werden für jede Methode *funktionale Eigenschaften* aus fachlicher Sicht und *architekturspezifische Eigenschaften* wie die Signatur oder die Zusicherungen beschrieben [OMG 2001:2-38ff.]. Allerdings werden nur die Signaturen formal-präzise dokumentiert, während zur Beschreibung ansonsten auf die natürliche Sprache zurückgegriffen wird.

4.7.4 Komponentenentwurf

Nach Fertigstellung des Systemkonzepts sind die zu realisierenden Komponenten und Adapter (die im Folgenden ebenfalls als Komponenten betrachtet werden [SIEDERSLEBEN 2004:69]) zu entwickeln. Dabei dient die jeweilige Komponenten- bzw. Adapterspezifikation zunächst als Ausgangspunkt für den Entwurf der internen Komponenten- bzw. Adapterarchitektur im Rahmen des Komponentenentwurfs. Durch die interne *Komponentenarchitektur* wird die Komponentenaußensicht auf mehrere Implementierungseinheiten abgebildet, weshalb die eingenommene Entwurfssicht auch als *Abbildungssicht* bezeichnet wird (vgl. Abschnitt 4.5.1). Das mit dem Komponentenentwurf verbundene Ziel ist die Bereitstellung der Komponentenfunktionalität, die in den Blue Pages der jeweiligen Komponentenspezifikation festgelegt ist. Dabei sind die architekturspezifischen und qualitativen Vorgaben der Green und Grey Pages einzuhalten.

Der Prozess des Komponentenentwurfs beginnt mit der Wahl eines geeigneten *Entwicklungsparadigmas*, nach dessen Vorgaben die Komponentenarchitektur zu erstellen ist. Im Rahmen der Fallstudie ist für die zu erstellenden Komponenten Auftragsabwicklung, Kundenmanagement, Lagermanagement und Rechnungsabwicklung jeweils ein objektorientierter Entwurf zu favorisieren, da das objektorientierte Entwicklungsparadigma von der einzusetzenden Microsoft .NET Plattform direkt unterstützt wird. Dementsprechend sind als Implementierungseinheiten zunächst geeignete *Klassen* zu identifizieren, die die an den Angebotsschnittstellen beschriebene Funktionalität realisieren. Als Grundlage sind dabei die *abstrakten Datentypen* zu verwenden, die sich aus den im Fachentwurf rekonstruierten Informationsobjekten durch Zuordnung der jeweils zugehörigen Funktionen ableiten lassen [MEYER 1997:684f.].

Die identifizierten Klassen sind anschließend zu einem System zu verbinden, das bspw. als UML *Klassendiagramm* dargestellt werden kann [OMG 2003b:80-94]. Dieses Klassendiagramm lässt sich als Komponenteninnensicht darstellen, indem es (unter Verwendung der

sog. *Whitebox-Sicht* [OMG 2003b:139-142]) in das Komponentendiagramm eingezeichnet wird. Für den strukturierten und objektorientierten Entwurf existiert umfangreiche Literatur, so dass auf eine ausführlichere Darstellung an dieser Stelle verzichtet wird. Darstellungen des objektorientierten Entwurfs finden sich bspw. bei [Jacobson, et al. 1992; Booch 1993; Cook und Daniels 1994; Meyer 1997; Schienmann 1997].

4.7.5 Implementierung

Nach der Evaluation der Komponentenarchitektur liegen schließlich die benötigten Spezifikationen vor, um mit der *Implementierung* der einzelnen Komponenten und Adapter zu beginnen. Dabei wird die zuvor erstellte Spezifikation der Komponenteninnensicht unter Einsatz von Implementierungswerkzeugen und einer Programmiersprache in entsprechende physische *Programmstrukturen* überführt. Üblicherweise lassen sich bereits einige Programmteile aus der Spezifikation der Komponenteninnensicht und der Spezifikation der Komponentenaußensicht generieren. Im Rahmen der Fallstudie wurde zur Implementierung der Komponenten die objektorientierte Programmiersprache C# verwendet, wobei die Strukturen der Klassen und Schnittstellen bereits aus der Spezifikation generiert werden konnten (zur Programmierung von .NET Komponenten vgl. [Löwy 2001]).

Anhand der fertig gestellten Implementierung lässt sich zunächst überprüfen, ob die vorgegebenen *Architekturmerkmale* der Außensicht eingehalten wurden. Anschließend ist durch sog. *Modultests* die Fehlerfreiheit der Implementierung zu überprüfen [Spillner und Linz 2005:40-47]. Dabei werden die Komponenten, mit denen die zu realisierende Komponente interagiert, üblicherweise nur durch stark vereinfachte Versionen (sog. Teststümpfe) repräsentiert. Im Rahmen der Modultests ergeben sich neben Hinweisen auf Fehler auch wichtige Hinweise, ob die realisierte Komponente in der Lage ist, die in den Blue Pages der Komponentenspezifikation festgelegte *Funktionalität* zu erbringen und die Qualitätsanforderungen zu erfüllen, die sich auf ihre *Funktionsweise* (d.h. die Genauigkeit, Interoperabilität, Sicherheit und Persistenz) beziehen. Die durchzuführenden Modultests tragen insofern dazu bei, die Einhaltung der Komponentenspezifikation zu validieren.

Mit Abschluss der Implementierung wechselt die *Bedeutung der Komponentenspezifikation*. Sie wird fortan nicht mehr genutzt, um den zu erreichenden Soll-Zustand präskriptiv (in Form eines Modells) zu beschreiben. Statt dessen ist der mit der Realisierung erreichte Ist-Zustand *deskriptiv* (in Form einer Dokumentation) zu beschreiben [Clements, et al. 2003:10]. Dabei werden die Vorgaben des Systementwurfs zunächst übernommen, so dass die erstellte Komponente in einem Entwicklungsrepositorium bspw. bereits durch die Yel-

low Pages *klassiert* wird. In den folgenden Entwicklungsphasen ist jedoch durch weitere Tests zu überprüfen, ob die spezifizierten Eigenschaften von der Realisierung auch tatsächlich erfüllt werden. Bei ggf. erkannten Abweichungen, die sich positiv auswirken oder zumindest toleriert werden können, ist die Spezifikation entsprechend anzupassen. Häufig sind im Rahmen der Dokumentation auch *zusätzliche Eigenschaften* der realisierten Komponente zu beschreiben, die nicht Bestandteil der ursprünglichen Komponentenspezifikation waren und sich bei der Entwicklung zusätzlich ergeben haben. So kann bspw. die zunächst nicht geforderte Einhaltung eines Daten- oder Funktionsstandards dazu führen, dass diese nachträglich im Rahmen der Grey Pages dokumentiert wird. Üblicherweise werden mit dem Abschluss der Implementierung auch *technische Komponentenmerkmale* (wie bspw. die in Abb. 4.76 für die Komponente Lagermanagement dargestellten Systemanforderungen) beschrieben und in die White Pages aufgenommen, sofern diese nicht bereits aus dem Pflichtenheft abzuleiten und in diesem Fall Vorgaben des Systementwurfs waren.

Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1GHz)
Hauptspeichergröße: 256 MB
Festspeichergröße: 512 MB
Betriebssystem (Microsoft Windows 2000)
Beschreibung: Minimale Systemanforderungen
Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1,5GHz)
Hauptspeichergröße: 1024 MB
Festspeichergröße: 2048 MB
Betriebssystem (Microsoft Windows XP)
Beschreibung: Optimale Systemanforderungen

Abb. 4.76: Systemanforderungen der Komponente Lagermanagement.

4.7.6 Konfigurierung

Nachdem alle zu erstellenden Komponenten und Adapter realisiert wurden, lassen sich diese im Rahmen der *Konfigurierung* zu einem Gesamtsystem verbinden. Maßgeblich für die Verbindung zu einem Gesamtsystem ist dabei die im Systementwurf erstellte Software-Architektur. Deren abstrakte Strukturierungseinheiten sind nun durch ihre jeweiligen Realisierungen zu ersetzen. Bei diesem Prozess, der auch als *Konfiguration* bezeichnet wird, sind für alle definierten Schnittstellen konkrete Realisierungen festzulegen, die die jeweils spezifizierten Dienste zur Verfügung stellen [SIEDERSLEBEN 2004:59f.]. Mehrere miteinander interagierende Komponenten lassen sich während der Konfigurierung zudem durch entsprechenden Programmcode explizit zu komplexen Komponenten verbinden. Bei die-

sem Prozess, der auch als *Komposition* bezeichnet wird [SIEDERSLEBEN 2004:61f.], entstehen für die logisch zusammengesetzten Komponenten des Bauplans die jeweiligen physisch zusammengesetzten Pendants. Die miteinander verbundenen Komponenten bleiben dabei entweder für andere Komponenten sichtbar oder werden nach außen verborgen.

Der Zweck der Konfigurierung unterscheidet sich je nachdem, ob eine Anwendungsentwicklung oder eine Komponentenentwicklung für den anonymen Markt durchgeführt werden soll. Bei einem *Anwendungsentwicklungsprojekt* stellt die Konfigurierung die Phase dar, in der mit dem Anwendungssystem das eigentliche Projektziel realisiert wird. Bei der *Komponentenentwicklung* wird die Konfigurierung dagegen nur durchgeführt, um die einzelnen Komponenten im Zusammenhang testen zu können oder diese als komplexe Komponente zu vermarkten. Der Prozess der Konfigurierung ist in der Literatur ausführlich beschrieben worden (vgl. [CHEESMAN und DANIELS 2001:147-165; SZYPERSKI, et al. 2002:475-480; SIEDERSLEBEN 2004:59-66]), weswegen auf eine nähere Darstellung an dieser Stelle verzichtet wird. Die Konfigurierung ist mit einem *Integrationstest* abzuschließen, mit dem die Verbindung der Komponenten auf ihre Fehlerfreiheit überprüft wird [SPILLNER und LINZ 2005:47-55].

4.7.7 Stabilisierung

Der durchgeführte Integrationstest ist jedoch nur der Auftakt zu einer umfassenden Analyse des entwickelten Anwendungs- bzw. Komponentensystems, die im Rahmen der *Stabilisierung* durchgeführt wird. Dabei wird in Zusammenarbeit mit Anwendern und Fachexperten die *materiale Korrektheit* des Gesamtsystems (d.h. die Übereinstimmung mit dem fachlichen Begriffssystem) im Rahmen eines *Abnahmetests* untersucht [SPILLNER und LINZ 2005:62-64]. Als Ausgangspunkt der Untersuchung dienen die im Fachkonzept festgelegten fachlichen Testfälle. Fällt der Abnahmetest positiv aus, kann dem Gesamtsystem und seinen einzelnen Komponenten die korrekte Realisierung der Funktionalität bescheinigt werden. Damit kann die während des Systementwurfs festgelegte *Komponentenfunktionalität* im Rahmen der Blue Pages auch zu Dokumentationszwecken übernommen werden. Zudem lassen sich aus dem Abnahmetest Kennzahlen ableiten, mit denen die fachliche *Eignung* der Komponente im Rahmen der Grey Pages dokumentiert werden kann.

Neben der materialen Korrektheit ist vor allem die *Performanz* des Systems und seiner Komponenten in verschiedenen Nutzungssituationen zu analysieren. Dabei kommen üblicherweise *Lasttests* zum Einsatz, die zunächst Aufschluss über die Performanz bei unterschiedlichen *Benutzungsprofilen* geben [SPILLNER und LINZ 2005:60]. Sollen die einzelnen

Komponenten (bspw. bei einem Komponentenentwicklungsprojekt) möglichst wieder verwendbar sein, ist bei der Untersuchung außerdem die Konfiguration des Systems zu verändern. Die Durchführung sog. *Konfigurationstests* gibt in diesem Fall Aufschluss über das Komponentenverhalten in verschiedenen *Software-Umgebungen*. Dabei ist zumindest bei einer angestrebten logischen Wiederverwendung auch das Verhalten der Komponente in unterschiedlichen *Hardware-Konfigurationen* zu überprüfen.

Die bei diesen Tests gewonnen Ergebnisse sind im Rahmen der Grey Pages zu dokumentieren und ersetzen dabei ggf. die im Systementwurf spezifizierte Performanz. So konnte im Rahmen der Fallstudie bspw. festgestellt werden, dass die Komponente Lagermanagement über eine bessere Performanz verfügt, als im Systementwurf gefordert wurde. Die in Abb. 4.74 gezeigten Qualitätsmerkmalsausprägungen sind deshalb unter Rückgriff auf die gewonnen Testergebnisse anzupassen. Darüber hinaus lassen sich im Rahmen der Stabilisierung auch die *Übergangswahrscheinlichkeiten* für die Zustandsübergänge der Dienstbedarfsprotokolle genauer ermitteln, die im Rahmen der Green Pages zu dokumentieren sind und zur Architekturevaluation in zukünftigen Anwendungsentwicklungsprojekten genutzt werden können. Bei einem Komponentenentwicklungsprojekt sind außerdem ggf. *Nutzungstests* mit Entwicklern durchzuführen, die Aufschluss über die *Verwendbarkeit* der realisierten Komponenten geben. Aus den Nutzungstests lassen sich wiederum Kennzahlen ableiten, die im Rahmen der Grey Pages zu dokumentieren sind.

4.7.8 Einführung

Nach der Stabilisierung folgt schließlich die Einführungsphase, die sich in Abhängigkeit von dem durchgeführten Projekttyp unterscheidet. Bei einem Anwendungsentwicklungsprojekt ist das realisierte Anwendungssystem nach der Betriebsfreigabe in den *Produktivbetrieb* einzuführen und in die Organisation des Unternehmens zu integrieren. Dabei sind Anwenderschulungen für die mit dem Anwendungssystem interagierenden Mitarbeiter durchzuführen und darüber hinaus ggf. die im Fachentwurf beschriebenen Veränderungen in den Produktivprozessen zu implementieren.

Bei einem Komponentenentwicklungsprojekt sind die realisierten Komponenten dagegen in den anonymen *Markt* einzuführen. Bei der Markteinführung erfolgt eine Publikation der Komponenten im Katalog eines unternehmensinternen oder -externen Marktplatzes. Dabei ist üblicherweise eine *Katalogisierung* der Komponente vorzunehmen, bei der die Komponente entsprechend ihrer Eigenschaften klassiert und in die Katalogstruktur eingeordnet wird [PRIETO-DÍAZ und FREEMAN 1987; PRIETO-DÍAZ 1991; SAMETINGER 1997:179-184].

Zur Katalogisierung können die Angaben der Yellow Pages übernommen werden, sofern das Klassifikationsschema des Katalogs zumindest auf einer der dort genutzten, standardisierten Taxonomien basiert. Mit der Publikation auf einem unternehmensexternen Marktplatz sind ferner auch *kommerzielle Komponentenmerkmale* wie bspw. die Vertriebswege, der jeweilige Lieferumfang und Preis sowie die Nutzungsbedingungen festzulegen. Diese Merkmale sind im Rahmen der White Pages in die Komponentendokumentation zu übernehmen, wie Abb. 4.77 und Abb. 4.78 am Beispiel der Komponente Lagermanagement zeigen. Mit der Dokumentation der kommerziellen Komponentenmerkmale wurde außerdem der Anbieter näher beschrieben und der bislang genannte projektinterne Ansprechpartner durch einen offiziellen Ansprechpartner ersetzt (vgl. Anhang A).

Vor der Publikation der Komponenten kann der Hersteller schließlich noch eine *Zertifizierung* der von ihm realisierten Komponenten anstreben, die die Übereinstimmung von Realisierung und Komponentenspezifikation (und darüber hinaus ggf. die Einhaltung von Standards) bescheinigt [IEEE 1991; APPERLY, et al. 2001:769; FLYNT und DESAI 2001:701; MORRIS, et al. 2001]. Dabei ist die zur Dokumentation angefertigte Komponentenspezifikation zusammen mit der Realisierung bei einem neutralen Zertifizierer einzureichen, der dann die Übereinstimmung prüft und ein entsprechendes Zertifikat ausstellt.

Vertriebsweg (Typ: Distributionskanal)
Kanaltyp (Typ: Postversand)
Preis: 1190.00
Waehrung (Typ: EUR)
Bezahlweise (Typ: Kreditkarte)
Bezahlweise (Typ: Nachnahme)
Lieferumfang (Typ: Artefakt)
Bezeichnung: setup.exe
Typ (Typ: Installationsdatei)
Beschreibung: Datei zum Installieren der Komponente
Lieferumfang (Typ: Artefakt)
Bezeichnung: setup.hlp
Typ (Typ: Dokumentation)
Beschreibung: Instruktionen zum Installieren der Komponente
Lieferumfang (Typ: Artefakt)
Bezeichnung: handbook.pdf
Typ (Typ: Dokumentation)
Beschreibung: Instruktionen zur Installation und Verwendung der Komponente
Lieferumfang (Typ: Artefakt)
Bezeichnung: demo.exe
Typ (Typ: Sonstige)
Beschreibung: Informationen über den Einsatz der Komponente und ihr Zusammenspiel mit anderen Komponenten

Abb. 4.77: Kommerzielle Merkmale der Komponente Lagermanagement.

Vertriebsweg (Typ: Distributionskanal)
Kanaltyp (Typ: Internetversand)
Preis: 990.00
Waehrung (Typ: EUR)
Bezahlweise (Typ: Kreditkarte)
Bezahlweise (Typ: Vorauszahlung)
Lieferumfang (Typ: Artefakt)
Bezeichnung: setup.exe
Typ (Typ: Installationsdatei)
Beschreibung: Datei zum Installieren der Komponente
Lieferumfang (Typ: Artefakt)
Bezeichnung: setup.hlp
Typ (Typ: Dokumentation)
Beschreibung: Instruktionen zum Installieren der Komponente
Vertragskondition (Typ:Vertragskondition)
Konditionstyp (Typ: Lizenzvereinbarung)
Kondition: Oversoft End-User Lizenzvereinbarung (EULA) ...
Vertragskondition (Typ:Vertragskondition)
Konditionstyp (Typ: Garantievereinbarung)
Kondition: Oversoft Garantiebestimmungen ...

Abb. 4.78: Kommerzielle Merkmale der Komponente Lagermanagement (fortgesetzt).

5 Schlussbetrachtung

Nachdem in den vorangegangenen Kapiteln die *Grundlagen* des UNSCOM Spezifikationsrahmens, seine einzelnen *Bestandteile* sowie seine *Anwendung* während des Entwicklungsprozesses beschrieben wurden, sind die erreichten Ergebnisse abschließend in der Gesamtschau zu betrachten. Dabei ist der Gedankengang, der der Konzeption des UNSCOM Spezifikationsrahmens zugrunde lag, zunächst noch einmal zu einem Gesamtbild zusammenzufassen (Abschnitt 5.1) und anschließend kritisch zu überprüfen (Abschnitt 5.2). Die Schlussbetrachtung endet mit einem Ausblick auf verbleibende offene Fragestellungen und anzustrebende zukünftige Entwicklungen (Abschnitt 5.3).

5.1 Zusammenfassung

Ausgehend von der Annahme, dass mit der Einführung einer *komponentenorientierten Entwicklungsmethodik* entscheidend dazu beigetragen werden kann, die aktuell bestehenden Herausforderungen in der (betrieblichen) Anwendungsentwicklung zu bewältigen, wurde im ersten Kapitel die Schaffung eines *einheitlichen Spezifikationsrahmens* zur Beschreibung der Komponentenaußensicht als kritischer Erfolgsfaktor hervorgehoben. Damit wurde zunächst die Motivation für die Konzeption des UNSCOM Spezifikationsrahmens gegeben. Gleichzeitig wurden mit der geforderten *Vollständigkeit* und *Abgestimmtheit* bereits erste grundlegende Anforderungen beschrieben, die bei der Konzeption einzuhalten waren.

Im zweiten Kapitel wurden im Rahmen einer Gegenstandsbestimmung weitere *Anforderungen* an einen einheitlichen Spezifikationsrahmen identifiziert. Dazu wurde zunächst das *komponentenorientierte Entwicklungsparadigma* beschrieben und gegen die anderen existierenden Paradigmen abgegrenzt. Anschließend wurde der komponentenorientierte Entwicklungsprozess und die zu seiner Unterstützung bereitzustellende *Entwicklungsmethodik* im Detail betrachtet. Hieraus ergaben sich mit den Forderungen nach *Angemessenheit*, *Vollständigkeit*, *Abgestimmtheit*, *Universalität*, *Verbindlichkeit*, *Verständlichkeit*, *Veränderbarkeit* und *Praktikabilität* schließlich die vom UNSCOM Spezifikationsrahmen zu erfüllenden grundlegenden Anforderungen. Um die Neukonzeption des Spezifikationsrahmens zu begründen, wurde abschließend auf die *Defizite existierender Ansätze* eingegangen und ein *Lösungskonzept* skizziert. Dieses Konzept wurde der Entwicklung des UNS-

COM Spezifikationsrahmens zugrunde gelegt, mit der gleichzeitig eine Integration der existierenden Ansätze durch einen *vereinheitlichenden Ansatz* angestrebt wurde.

Aufbauend auf dem Lösungskonzept wurden im dritten Kapitel zunächst die wesentlichen *Grundlagen* dargestellt, auf denen der UNSCOM Spezifikationsrahmen basiert. Von zentraler Bedeutung ist dabei das *konzeptionelle Komponentenmodell*, das aus dem Komponenten-Konnektor-Modell abgeleitet und gleichzeitig systemtheoretisch begründet wurde. Das so begründete Komponentenmodell wurde anschließend um *typtheoretische Überlegungen* erweitert und mit dem Konzept des *Software-Vertrags* um ein etabliertes Prinzip zur Beschreibung von software-spezifischen Merkmalen ergänzt. Anhand der systemtheoretischen Modellperspektiven und der während der Entwicklung eingenommenen Abstraktionsebenen wurden dann die zu spezifizierenden Komponenteneigenschaften mithilfe eines *Klassifikationsschemas* bestimmt und verschiedenen Vertragsebenen zugeordnet.

Für die identifizierten Vertragsebenen wurden im vierten Kapitel schließlich detaillierte Vorgaben hinsichtlich des zu beschreibenden *Inhalts* und der zur Darstellung zu verwendenden *Repräsentationsformate* vorgestellt, die den Kern des UNSCOM Spezifikationsrahmens bilden. Dabei wurde jeweils ein textbasiertes (UNSCOM/T), ein graphisches (UNSCOM/G) sowie ein XML (UNSCOM/X) Format definiert. Darüber hinaus wurden Vorgaben zur *Geltung* der einzelnen Spezifikationsteile während des Entwicklungsprozesses dargestellt, die Bezug auf ein vereinheitlichtes Vorgehensmodell nehmen. Unter Rückgriff auf dieses Vorgehensmodell wurde abschließend auch die *Anwendung* des UNSCOM Spezifikationsrahmens im Rahmen einer exemplarisch durchgeführten Fallstudie beschrieben.

5.2 Kritische Würdigung

Mit der durchgeführten Fallstudie wurde vor allem die *Anwendbarkeit* des UNSCOM Spezifikationsrahmens bewiesen und dabei zugleich seine grundsätzliche *Eignung* zur Beschreibung der Außensicht von Komponenten demonstriert. Da der UNSCOM Spezifikationsrahmen auf der Basis der in Kapitel 3 dargestellten Grundlagen außerdem so konzipiert wurde, dass die in der Gegenstandsbestimmung identifizierten *grundlegenden Anforderungen* von ihm erfüllt werden (zur ausführlichen Diskussion vgl. Abschnitt 2.3.3), ist er anderen bislang existierenden Ansätzen überlegen. Aus analytischer Sicht stellt der beschriebene UNSCOM Spezifikationsrahmen deshalb einen wichtigen Schritt zur mittelfristig anzustrebenden Schaffung eines *Spezifikationsstandards* dar. Bestätigt wird diese Einschätzung dabei durch den Abschlussbericht eines international anerkannten Workshops, in dessen Rahmen das Thema „Spezifikation der Komponentenaußensicht“ von Experten diskutiert

und die mit dem UNSCOM Spezifikationsrahmen entwickelten Vorschläge in unveränderter Form übernommen wurden [BOSCH, et al. 2003:8-10].

Als *vereinheitlichender Ansatz* umfasst der dargestellte UNSCOM Spezifikationsrahmen mit seinen Vorgaben zugleich existierende Ansätze wie bspw. *Catalysis* [D'SOUZA und WILLS 1999] oder *UML Components* [CHEESMAN und DANIELS 2001] und ist somit in der Lage, diese zu ersetzen. Wegen der vorzugsweise eingesetzten standardisierten Notationen kann zur Spezifikation nach den Vorgaben des UNSCOM Spezifikationsrahmens außerdem auf bereits existierende Entwicklungswerkzeuge zurückgegriffen werden, wodurch eine Neubeschaffung von Werkzeugen prinzipiell entfällt. Um Konzepte wie z.B. die automatische Konvertierung zwischen den verschiedenen Repräsentationsformaten in vollem Umfang nutzen zu können, ist jedoch der Einsatz spezialisierter Werkzeuge erforderlich.

Zu überprüfen bleibt vor allem die Frage, inwieweit der UNSCOM Spezifikationsrahmen tatsächlich in der Lage ist, die *Effizienz* des komponentenorientierten Entwicklungsprozesses zu steigern. Zur Beantwortung dieser Frage ist zunächst hervorzuheben, dass der UNSCOM Spezifikationsrahmen mit (fachlich) funktionalen, architekturspezifischen und qualitativen Eigenschaften mehr Aspekte in die Beschreibung der Komponentenaußensicht einbezieht als andere Ansätze wie bspw. Catalysis oder UML Components (vgl. Abschnitt 2.3.2.2). Durch diese zusätzlich beschriebenen Eigenschaften lässt sich einerseits die *Komponentenentwicklung* präziser steuern. Andererseits tragen die zusätzlich beschriebenen Eigenschaften wesentlich dazu bei, den Prozess der Komponentenauswahl während der *Anwendungsentwicklung* zu verbessern [VITHARANA, et al. 2003].

Im Gegenzug wird jedoch häufig argumentiert, dass die Spezifikation einen *höheren Aufwand* verursacht, der auch bei den erzielbaren Vorteilen nicht zu vertreten ist. Gerade im Vergleich mit den zuvor erwähnten Ansätzen hält sich der Mehraufwand bei der Spezifikation mit dem UNSCOM Spezifikationsrahmen jedoch in Grenzen. So ist mit den Qualitätseigenschaften lediglich *ein* zusätzlicher Aspekt zu spezifizieren, der in der modernen Anwendungsentwicklung noch dazu stetig an Bedeutung gewinnt. Bei der Spezifikation des Begriffssystems im Fachentwurf steigt im Vergleich zum Domain bzw. Business Concept Model der anderen Ansätze dagegen lediglich der Grad der *Formalisierung*. Im Umkehrschluss wird dafür allerdings die Ableitung von Datentypen und Funktionen im Systementwurf erleichtert. Gleichzeitig lässt sich das stärker formalisierte Begriffssystem zur Spezifikation der Komponentenfunktionalität verwenden.

Schließlich besitzt der UNSCOM Spezifikationsrahmen auch deshalb ein Potenzial zur Steigerung der Effizienz des Entwicklungsprozesses, da die mit ihm zu beschreibenden Informationen die Durchführung zahlreicher *Entwicklungsaufgaben* erleichtern. Unter Be-

zugnahme auf die in Abschnitt 2.2.3 vorgestellte Kompositionsmethodik lassen sich dabei zumindest folgende Aufgaben unterstützen:

- **Katalogisierung**: Klassierung [PRIETO-DÍAZ und FREEMAN 1987; PRIETO-DÍAZ 1991], Hinzufügung funktionaler Merkmale [VITHARANA, et al. 2003:649f.]
- **Suche**: Klassifikations- und spezifikationsbasierte Suche [MILI, et al. 1995:550-552]
- **Kompatibilitätstests**: Test der Funktionalität [NOY 2004b], Signaturen [ZAREMSKI und WING 1995], Vor- und Nachbedingungen [ZAREMSKI und WING 1997], Protokolle [NIERSTRASZ 1993] und Qualitätseigenschaften [FROLUND und KOISTINEN 1998:38-54]
- **Adaptergenerierung**: Begriffsabweichungen [ORTNER 1997:32], Signaturen [YELLIN und STROM 1994], Protokolle [YELLIN und STROM 1997; SCHMIDT und REUSSNER 2002] und Qualitätsmerkmale
- **Ableitung von Eigenschaften**: Vorhersage der Zuverlässigkeit [REUSSNER, et al. 2003] und Effizienz [WALLNAU 2003]

5.3 Ausblick

Zur Unterstützung der eben genannten Aufgaben sind jedoch noch konkrete *Werkzeuge* zu realisieren, die die Komponentenspezifikationen des UNSCOM Spezifikationsrahmens verwenden können. Die Realisierung einer *integrierten Methodik*, die eine Vielzahl solcher Werkzeuge zu einer einheitlichen Entwicklungsumgebung zusammenfasst und dabei – wie in der Einleitung beschrieben – auf dem UNSCOM Spezifikationsrahmen aufsetzt, ist dementsprechend auch eines der längerfristig anzustrebenden Ziele. Eine wichtige Voraussetzung für die Entstehung einer Methodik, die über eine möglichst breite Akzeptanz verfügt, ist dabei die Schaffung eines *Spezifikationsstandards*. Dieses Ziel wird in Deutschland von einem Arbeitskreis der Gesellschaft für Informatik verfolgt, der im Jahre 2002 einen entsprechenden Vorschlag vorgelegt hat [ACKERMANN, et al. 2002]. Im Zuge seiner Evolution sollen die Konzepte des UNSCOM Spezifikationsrahmens, die zumindest teilweise auf dem eben genannten Vorschlag aufsetzen und ihn an zahlreichen Stellen weiterentwickeln, deshalb dort eingebracht werden.

Zu überprüfen und ggf. anzupassen ist bei der Standardisierung vor allem der Kompromiss zwischen Ausdrucksmächtigkeit und Anwendbarkeit, der bei der Konzeption des UNSCOM Spezifikationsrahmens unter anderem im Hinblick auf die *Protokollspezifikation* eingegan-

gen wurde. Im Zuge einer solchen Überprüfung sind die derzeit verwendeten deterministischen endlichen Automaten, die nur über eine eingeschränkte Aussagekraft verfügen, ggf. durch aussagekräftigere Konzepte (wie Petrinetze, Prozessalgebren etc.) abzulösen. Zu erhalten ist dabei jedoch unbedingt die *statische Auswertbarkeit* von Protokollinformationen, da diese zur Unterstützung zahlreicher Entwicklungsaufgaben erforderlich ist.

Schließlich bleiben weitere Vorgaben zu entwickeln, mit denen sich bspw. die *Parametrisierung von Komponenten* und ihre möglichen Auswirkungen auf einzelne Komponenteneigenschaften beschreiben lassen. Gerade bei der Entwicklung betrieblicher Anwendungssysteme ergibt sich wegen der hohen Komplexität häufig der Bedarf, parametrisierbare Komponenten zu realisieren, die sich an eine Vielzahl von Anwendungskontexten anpassen lassen. Die zur Anpassung vorgesehenen *Parameter* lassen sich zwar grundsätzlich mit dem UNSCOM Spezifikationsrahmen beschreiben. Dabei kann durch die Zuordnung eines *generalisierten Fachbegriffs* (für den mehrere Spezialisierungen bestehen) sogar die variable Funktionalität dokumentiert werden. Jedoch ergeben sich durch das Setzen eines Parameterwerts meist weitere Auswirkungen (wie z.B. veränderte Protokolle oder Zusicherungen), die derzeit nicht in Zusammenhang mit den jeweiligen Parameter gebracht werden können. Ebenso sind Vorgaben zur Spezifikation von Komponenten zu entwickeln, die bei der Realisierung *graphischer Benutzungsschnittstellen* eingesetzt werden. Diese Komponentenart wurde im Rahmen des konzeptionellen Komponentenmodells bislang nicht untersucht und ist daher spätestens bei der Schaffung eines Spezifikationsstandards noch in die Betrachtung aufzunehmen. Aufgrund ihrer speziellen darstellungsspezifischen Eigenschaften (wie bspw. Breite, Höhe, Icon, Auflösung etc.) ist dabei davon auszugehen, dass eine weitere spezialisierte Vertragsebene zur Spezifikation einzuführen sein wird.

Wie in der Einleitung dargestellt, lässt sich der komponentenorientierte Entwicklungsprozess jedoch bereits durch die Einführung des UNSCOM Spezifikationsrahmens in seiner jetzigen Form entscheidend unterstützen. Diese Unterstützung kann wesentlich dazu beitragen, den bestehenden Herausforderungen der (betrieblichen) Anwendungsentwicklung durch einen komponentenorientierten Entwicklungsansatz besser zu begegnen und mit der Realisierung von flexibel anpassbaren und robusten Anwendungssystemen den Unternehmenserfolg in sich immer schneller verändernden Geschäftsumfeldern zu sichern.

A Spezifikationsbeispiel

Im Folgenden ist die Spezifikation der Komponente Lagermanagement dargestellt, die im Rahmen der zuvor beschriebenen Fallstudie entwickelt und hinsichtlich ihrer Außensicht beispielhaft spezifiziert wurde. Zur Repräsentation wird an dieser Stelle jedoch durchgängig auf das UNSCOM/T Format zurückgegriffen (zur Darstellung der Spezifikation in UNSCOM/G vgl. Abschnitt 4.7). In der nachfolgenden Darstellung werden nur diejenigen Spezifikationsteile gezeigt, die bereits bei der Beschreibung der Fallstudie dargestellt wurden. Das Ziel der Darstellung ist dementsprechend nicht die Vervollständigung des Anwendungsbeispiels, sondern vielmehr seine Zusammenfassung zu einer Gesamtsicht. Der bewusst vorgenommene Formatwechsel verdeutlicht dabei noch einmal die Äquivalenz der UNSCOM/G und UNSCOM/G Formate. Zur vollständigen Darstellung der Zusammenhänge, die der Realisierung der Komponente Lagermanagement zugrunde liegen, vgl. [BECKER und SCHÜTTE 2004].

A.1 White Pages

Name: Oversoft Lagermanagement
Kennung: 206B2F65-E7BC-47A0-8DCF-4E34347322AA
Version: 1.0.3755
Variante: Small Business Edition
Beschreibung: Oversoft Lagermanagement stellt Funktionen zur Lagerverwaltung zur Verfügung, die den Warenein- und -ausgang sowie die Bestandsverwaltung abdecken. Die Small Business Edition ist dabei für mittelständische Betriebe gedacht und erlaubt die Verwaltung von maximal fünf Lagerstätten ...
Installationsbasis: < 10
Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1GHz)
Hauptspeichergröße: 256 MB
Festspeichergröße: 512 MB
Betriebssystem (Microsoft Windows 2000)
Beschreibung: Minimale Systemanforderungen
Systemanforderungen (Typ: Systemanforderungen)
Prozessor (Typ: iPII 1,5GHz)
Hauptspeichergröße: 1024 MB
Festspeichergröße: 2048 MB
Betriebssystem (Microsoft Windows XP)
Beschreibung: Optimale Systemanforderungen

- **Anbieter** (Typ: Unternehmen)
  - **Name**: Oversoft Software
  - **Kennung**: AEFBE5B6-A13F-4CA4-B322-32ADB89EA14C
  - **Beschreibung**: Oversoft Software ist spezialisiert auf die Bereitstellung von Werkzeugen zur Unterstützung des komponenten-orientierten Entwicklungsprozesses und auf die Bereitstellung spezialisierter Komponenten bzw. XML Web-Services ...
  - **URL**: http://www.oversoft.biz
  - **Unternehmenskennung** (Typ: Eingetragene Marken-ID)
    - **ID**: 30071881
    - **Beschreibung**: Eintrag ins deutsche Markenregister
- **Ansprechpartner** (Typ: Kontakt)
  - **Name**: Oversoft Marketing
  - **Adresse** (Typ: Adresse)
    - **Adresselement** (Typ: Strasse): An der Steinmauer
    - **Adresselement** (Typ: Hausnummer): 2
    - **Adresselement** (Typ: Postleitzahl): 61191
    - **Adresselement** (Typ: Ort): Rosbach
    - **Adresselement** (Typ: Land): Deutschland
  - **Kommunikationsmedium** (Typ: E-Mail): sales@oversoft.biz
  - **Kommunikationsmedium** (Typ: Telefon): +49 6003 9300-71
  - **Kommunikationsmedium** (Typ: Telefax): +49 6003 9300-72
- **Vertriebsweg** (Typ: Distributionskanal)
  - **Kanaltyp** (Typ: Postversand)
  - **Preis**: 1190.00
  - **Waehrung** (Typ: EUR)
  - **Bezahlweise** (Typ: Kreditkarte)
  - **Bezahlweise** (Typ: Nachnahme)
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.exe
    - **Typ** (Typ: Installationsdatei)
    - **Beschreibung**: Datei zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.hlp
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: handbook.pdf
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zur Installation und Verwendung der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: demo.exe
    - **Typ** (Typ: Sonstige)
    - **Beschreibung**: Informationen über den Einsatz der Komponente und ihr Zusammenspiel mit anderen Komponenten

- **Vertriebsweg** (Typ: Distributionskanal)
  - **Kanaltyp** (Typ: Internetversand)
  - **Preis**: 990.00
  - **Waehrung** (Typ: EUR)
  - **Bezahlweise** (Typ: Kreditkarte)
  - **Bezahlweise** (Typ: Vorauszahlung)
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.exe
    - **Typ** (Typ: Installationsdatei)
    - **Beschreibung**: Datei zum Installieren der Komponente
  - **Lieferumfang** (Typ: Artefakt)
    - **Bezeichnung**: setup.hlp
    - **Typ** (Typ: Dokumentation)
    - **Beschreibung**: Instruktionen zum Installieren der Komponente
- **Vertragskondition** (Typ:Vertragskondition)
  - **Konditionstyp** (Typ: Lizenzvereinbarung)
  - **Kondition**: Oversoft End-User Lizenzvereinbarung (EULA) ...
- **Vertragskondition** (Typ:Vertragskondition)
  - **Konditionstyp** (Typ: Garantievereinbarung)
  - **Kondition**: Oversoft Garantiebestimmungen ...

A.2 Yellow Pages

- **Klassierung** (Typ: Klassierung)
  - **Klassem** (Typ: Klassifikationseintrag)
    - **Facette** (Typ: Technologie)
    - **Taxonomie**: (Typ: Component Model Classification System)
    - **Wert**: Microsoft .NET 1.1
  - **Klassem** (Typ: Klassifikationseintrag)
    - **Facette** (Typ: Komponentenart)
    - **Taxonomie**: (Typ: Conceptual Component Type Classification System)
    - **Wert**: Application-Specific Component
  - **Klassem** (Typ: Klassifikationseintrag)
    - **Facette** (Typ: Wiederverwendungsart)
    - **Taxonomie**: (Typ: Reuse Type Classification System)
    - **Wert**: Logical Reuse
  - **Klassem** (Typ: Klassifikationseintrag)
    - **Facette** (Typ: Anwendungsbereich)
    - **Taxonomie**: (Typ: Application Domain Classification System)
    - **Wert**: Warehousing
  - **Klassem** (Typ: Klassifikationseintrag)
    - **Facette** (Typ: Funktionsbereich)
    - **Taxonomie**: (Typ: North American Industry Classification System)
    - **Wert**: General Warehousing and Storage

A.3 Blue Pages

A.3.1 Lexikon

Terminologie (Typ: Lexikon)
Begriff (Typ: Informationsobjekt)
Begriffswort: Kunde
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kunde ist eine natürliche oder juristische Person, die Güter nachfragt.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kundenauftrag
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kundenauftrag ist eine juristisch bindende Bestellung von Gütern durch einen Kunden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Artikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Artikel ist ein wirtschaftliches Gut.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lagerartikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Lagerartikel ist ein Artikel, der vom Handelsunternehmen gelagert wird.
Begriff (Typ: Informationsobjekt)
Begriffswort: Verkaufsartikel
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Verkaufsartikel ist ein Artikel, der vom Handelsunternehmen verkauft wird.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kundenauftragsposition
Genus: (Typ: Femininum)
Kurzdefinition: Eine Kundenauftragsposition legt die Menge eines bestellten Artikels fest.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lager
Genus: (Typ: Neutrum)
Kurzdefinition: Ein Lager ist ein Ort, an dem Artikel aufbewahrt werden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Lagerbereich
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Lagerbereich ist ein Ort, an dem Artikel zu einem bestimmten Zweck aufbewahrt werden.
Begriff (Typ: Informationsobjekt)
Begriffswort: Kommissionierbereich
Genus: (Typ: Maskulinum)
Kurzdefinition: Ein Kommissionierbereich ist ein Lagerbereich, an dem Artikel auftragsspezifisch zusammengestellt werden.

Terminologie (Typ: Lexikon)

Begriff (Typ: Funktion)

Begriffswort: Bestand reservieren

Kurzdefinition: Durch das Reservieren eines Bestands wird eine Menge von Lagerartikeln einem Kundenauftrag zugeordnet.

Begriff (Typ: Funktion)

Begriffswort: Reservierung löschen

Kurzdefinition: Durch das Löschen einer Reservierung wird eine Menge von Lagerartikeln freigegeben.

Begriff (Typ: Funktion)

Begriffswort: Kommissionierplan erstellen

Kurzdefinition: Durch das Erstellen eines Kommissionierplans wird die Zusammenstellung von Lagerartikeln vorgegeben.

Begriff (Typ: Funktion)

Begriffswort: Lagerartikel kommissionieren

Kurzdefinition: Durch das Kommissionieren von Lagerartikeln werden diese aus dem Lager zusammengestellt.

Begriff (Typ: Funktion)

Begriffswort: Lagerartikel erfassen

Kurzdefinition: Durch das Erfassen von Lagerartikeln wird ein Warenausgang ermittelt.

Begriff (Typ: Funktion)

Begriffswort: Bestandsveränderung ermitteln

Kurzdefinition: Durch das Ermitteln der Bestandsveränderung wird die Veränderung im Bestand eines Artikels quantifiziert.

Begriff (Typ: Funktion)

Begriffswort: Bestandsveränderung buchen

Kurzdefinition: Durch das Buchen einer Bestandsveränderung wird eine Veränderung im Bestand eines Lagerartikels erfasst.

Begriff (Typ: Funktion)

Begriffswort: Nachschubauftrag veranlassen

Kurzdefinition: Durch das Veranlassen eines Nachschubauftrags wird das Auffüllen eines Artikelbestands eingeleitet.

Begriff (Typ: Funktion)

Begriffswort: PhysischenBestand prüfen

Kurzdefinition: Durch das Prüfen eines physischen Bestands wird die vorhandene Menge eines Lagerartikels ermittelt.

Begriff (Typ: Funktion)

Begriffswort: Bestand prüfen

Kurzdefinition: Durch das Prüfen eines Bestands wird die verfügbare Menge eines Lagerartikels ermittelt.

Begriff (Typ: Prozess)

Begriffswort: Kundenauftrag abwickeln

Kurzdefinition: Das Abwickeln eines Kundenauftrags umfasst alle Tätigkeiten zur Bearbeitung eines Kundenauftrags.

Begriff (Typ: Prozess)

Begriffswort: Kundenauftrag prüfen

Kurzdefinition: Das Prüfen eines Kundenauftrags umfasst alle Tätigkeiten zur Freigabe eines neu erstellten Auftrags.

Begriff (Typ: Prozess)

Begriffswort: Lagerartikel auslagern

Kurzdefinition: Das Auslagern von Lagerartikeln umfasst alle logistischen Aktivitäten zur Bearbeitung eines Kundenauftrags.

Begriff (Typ: Prozess)

Begriffswort: Warenausgang buchen

Kurzdefinition: Das Buchen eines Warenausgangs umfasst alle Aktivitäten zur Verarbeitung einer Bestandsveränderung.

A.3.2 Aussagensammlung

Ein Lieferantenauftrag ist ein Auftrag.
Ein Kundenauftrag ist ein Auftrag.
Ein Auftrag ist ein Kundenauftrag oder ein Lieferantenauftrag.
Ein Kundenauftrag besteht aus ein bis beliebig vielen Kundenauftragsposition.
Ein Kundenauftrag hat eine Auftragsnummer und ein Auftragsdatum und eine Lieferanschrift und eine Rechnungsanschrift und einen Auftragsstatus.
Eine Auftragsnummer ist eine Zeichenkette.
Ein Auftragsdatum ist ein Datum.
Eine Lieferanschrift ist zusammengesetzt aus einem Name und einer Straße und einer Hausnummer und einer Postleitzahl und einem Ort.
Ein Name ist eine Zeichenkette.
Eine Straße ist eine Zeichenkette.
Eine Hausnummer ist eine Zeichenkette.
Eine Postleitzahl ist eine Zeichenkette.
Ein Ort ist eine Zeichenkette.
Eine Rechnungsanschrift ist zusammengesetzt aus einem Name und einer Straße und einer Hausnummer und einer Postleitzahl und einem Ort.
Eine Kundenauftragsposition verknüpft einen Kunde und einen Verkaufsartikel.
Eine Kundenauftragsposition hat eine Artikelmenge.
Eine Artikelmenge ist eine Ganzzahl.
Ein Auftragsstatus ist aktiv oder geprüft oder kommissioniert oder versandt oder storniert.

Ein Lagerartikel ist ein Artikel.
Ein Verkaufsartikel ist ein Artikel.
Ein Artikel ist ein Lagerartikel oder ein Verkaufsartikel.
Ein Artikel hat 1 bis beliebig viele Artikelnummer und einen Name und eine Warengruppe und einen Anlagezeitpunkt und einen Änderungszeitpunkt und eine Steuerklasse und 0 bis 1 Ursprungsland und 0 bis 1 Ursprungsregion.
Eine Warengruppe ist eine Zeichenkette.
Eine Artikelnummer ist eine Zeichenkette.
Ein Anlagezeitpunkt ist ein Datum.
Ein Änderungszeitpunkt ist ein Datum.
Eine Steuerklasse ist voll oder ermäßigt.
Ein Ursprungsland ist eine Zeichenkette.
Eine Ursprungsregion ist eine Zeichenkette.
Ein Lagerartikel hat eine Länge und eine Breite und eine Höhe und 0 bis 1 Volumen und 0 bis 1 Bruttogewicht und 0 bis 1 Nettogewicht und eine Lagereinheit und 1 bis beliebig viele Lagerplatzort.
Eine Länge ist eine Ganzzahl und wird gemessen in Zentimeter.
Ein Volumen ist eine Ganzzahl und wird gemessen in Milliliter.
Ein Bruttogewicht ist eine Ganzzahl und wird gemessen in Gramm.
Ein Nettogewicht ist eine Ganzzahl und wird gemessen in Gramm.
Eine Lagereinheit ist Europalette oder Einzelartikel.

Ein Bestandskonto hat eine Artikelnummer und eine Kontonummer und einen Bestand und einen PhysischerBestand und einen Mindestbestand.
Eine Kontonummer ist eine Ganzzahl.
Ein Bestand ist eine Ganzzahl.
Ein Mindestbestand ist eine Ganzzahl.
Ein PhysischerBestand ist eine Ganzzahl.
Ein Bestandskonto besteht aus 0 bis beliebig vielen Bestandsveränderung und 0 bis beliebig vielen Reservierung.
Eine Bestandsveränderung hat einen Änderungstyp und einen Auftrag und eine Artikelmenge.
Ein Änderungstyp ist Wareneingang oder Warenausgang.
Eine Reservierung hat eine Artikelmenge.

Ein Einlagerungsplan besteht aus 0 bis beliebig vielen Planposition.
Ein Kommissionierplan besteht aus 0 bis beliebig vielen Planposition.
Eine Planposition verknüpft einen Auftrag und einen Lagerartikel und einen Lagerplatz.
Eine Planposition hat eine Artikelmenge.
Ein Einlagerungsplan hat ein Erstellungsdatum.
Ein Kommissionierplan hat ein Erstellungsdatum.
Ein Erstellungsdatum ist ein Datum.

Ein Wareneingang hat ein Ausführungsdatum und 0 bis beliebig viele Lagerposition.
Ein Warenausgang hat ein Ausführungsdatum 0 bis beliebig viele Lagerposition.
Eine Lagerposition ist zusammengesetzt aus einer Artikelnummer und einer Artikelmenge.
Ein Ausführungsdatum ist ein Datum.

Ein Lager besteht aus 0 bis beliebig vielen Lagerbereich.
Ein Lager hat einen Name.
Ein Kommissionierbereich ist ein Lagerbereich.
Ein Reservebereich ist ein Lagerbereich.
Ein Wareneingangsbereich ist ein Lagerbereich.
Ein Warenausgangsbereich ist ein Lagerbereich.
Ein Lagerbereich ist ein Kommissionierbereich oder ein Reservebereich oder ein Wareneingangsbereich oder ein Warenausgangsbereich.
Ein Lagerbereich hat einen Name und eine Lagerstrategie.
Ein Lagerbereich besteht aus 0 bis beliebig vielen Lagerplatz.
Ein Name ist eine Zeichenkette.
Eine Lagerstrategie ist festplatzorientiert oder chaotisch.
Ein Lagerplatz hat einen Lagerplatzort und einen Lagerplatztyp und einen Lagerplatzbestand.
Ein Lagerplatzort ist zusammengesetzt aus einer Gangnummer und einer Regalnummer und einer Ebenennummer.
Ein Lagerplatztyp ist zusammengesetzt aus einer Dimension und einer Tragfähigkeit und einer Lagereinheit und einer Maximalbelegung.
Eine Gangnummer ist eine Zeichenkette.
Eine Regalnummer ist eine Zeichenkette.
Eine Ebenennummer ist eine Zeichenkette.
Eine Dimension ist zusammengesetzt aus einer Breite und einer Höhe und einer Tiefe.

Eine Breite ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Höhe ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Tiefe ist eine Ganzzahl und wird gemessen in Zentimeter.
Eine Tragfähigkeit ist eine Ganzzahl und wird gemessen in Gramm.
Eine Maximalbelegung ist eine Ganzzahl.
Ein Lagerplatzbestand ist zusammengesetzt aus 0 bis beliebig vielen Artikelbestand.
Ein Artikelbestand ist zusammengesetzt aus einer Artikelnummer und einem Bestand.

Lagerartikel auslagern besteht aus Kommissionierplan erstellen und Lagerartikel kommissionieren und Lagerartikel erfassen und Warenausgang buchen.
Ein Lagerverwalter tut einen Kommissionierplan erstellen mit einem Kundenauftrag.
Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel kommissionieren mit einem Kommissionierplan.
Ein Lagerverwalter tut 1 bis beliebig viele Lagerartikel erfassen mit einem Kommissionierplan zu einem Warenausgang.
Ein Lagerverwalter tut einen Warenausgang buchen.
Lagerartikel auslagern beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Kommissionierplan erstellen bei Kommissionierplan erstellen ist durchzuführen.
Nach Kommissionierplan erstellen folgt Lagerartikel kommissionieren bei Kommissionierplan erstellen ist abgeschlossen.
Nach Lagerartikel kommissionieren folgt Lagerartikel erfassen bei Lagerartikel kommissionieren ist abgeschlossen.
Nach Lagerartikel erfassen folgt Warenausgang buchen bei Lagerartikel erfassen ist abgeschlossen.
Nach Warenausgang buchen erfolgt Ablauf beenden mit Lagerartikel auslagern ist abgeschlossen.

Warenausgang buchen besteht aus Bestandsveränderung ermitteln und Bestandsveränderung buchen und Nachschubauftrag erstellen.
Ein Lagerverwalter tut 1 bis beliebig viele Bestandsveränderung ermitteln mit einem Warenausgang.
Ein Lagerverwalter tut eine Bestandsveränderung buchen mit einer Reservierung zu einem Bestand.
Ein Lagerverwalter tut einen Nachschubauftrag veranlassen.
Warenausgang buchen beginnt mit ausführen Ablauf starten.
Nach Ablauf starten folgt Bestandsveränderung ermitteln bei Bestandsveränderung ermitteln ist durchzuführen.
Nach Bestandsveränderung ermitteln folgt Bestandsveränderung buchen bei Bestandsveränderung ermitteln ist abgeschlossen.
Nach Bestandsveränderung buchen folgt Alternativablauf starten.
Nach Alternativablauf starten folgt entweder Nachschubauftrag veranlassen bei Bestand ist kleiner oder gleich Mindestbestand oder Nichts tun bei Bestand ist größer als Mindestbestand.
Nach Nachschubauftrag veranlassen oder Nichts tun folgt Alternativablauf beenden.
Nach Alternativablauf beenden folgt Ablauf beenden mit Warenausgang buchen ist abgeschlossen.
Ein Sachbearbeiter tut einen Bestand reservieren mit einem Kundenauftrag zu einer Reservierung.
Ein Sachbearbeiter tut eine Reservierung löschen.

A.4 Green Pages

A.4.1 Signaturen

```
module Lagerorganisation {
  const long LagerzahlLimit = 5;

  typedef string ID;
  typedef sequence<ID> ID__Seq;
  typedef long Bestand;

  exception AenderungVerweigert {string error;};
  exception LoeschungVerweigert {string error;};

  enum Lagereinheit {
    Artikel,
    Europalette
  };

  interface ILagervwt {
    struct Lagerplatztyp {
      string       name;
      long         hoehe;
      long         breite;
      long         tiefe;
      long         tragfaehigkeit;
      long         maximalbelegung;
      Lagereinheit einheit;
    };

    struct Artikelbestand {
      ID      artikel;
      Bestand b;
    };

    typedef sequence<Artikelbestand> Artikelbestand__Seq;

    struct Lagerplatz {
      string              typ;
      string              gangNummer;
      string              regalNummer;
      string              ebenenNummer;
      Artikelbestand__Seq lagerplatzbestand;
    };

    typedef sequence<Lagerplatz> Lagerplatz__Seq;

    enum Lagerstrategie {
      festplatzorientiert,
      chaotisch
    };

    enum Lagerbereichstyp {
      Reservebereich,
      Kommissionierbereich,
      Wareneingangsbereich,
      Warenausgangsbereich,
      Versandbereich
    };

    struct Lagerbereichsstruktur {
      string            name;
      Lagerbereichstyp  typ;
      Lagerstrategie    strategie;
      Lagerplatz__Seq   plaetze;
    };
```

```
  struct Lagerbereich {
    string           name;
    Lagerbereichstyp typ;
    Lagerstrategie   strategie;
    ID__Seq          plaetze;
  };

  typedef sequence<Lagerbereich> Lagerbereich__Seq;

  struct Lagerstruktur {
    string            name;
    Lagerbereich__Seq bereiche;
  };

  struct Lager {
    string  name;
    ID__Seq bereiche;
  };

  exception LagerzahlLimitUeberschritten {};
  exception LagerplatztypUngueltig {};
  exception LagerplatzUngueltig {};
  exception LagerbereichUngueltig {};
  exception LagerUngueltig {};

  ID                 erstelleLagerplatztyp(in Lagerplatztyp t)
                     raises (LagerplatztypUngueltig);
  Lagerplatztyp      leseLagerplatztyp(in ID typ)
                     raises (LagerplatztypUngueltig);
  void               aendereLagerplatztyp(in ID typ,
                     in long maximalbelegung, in Lagereinheit e)
                     raises (LagerplatztypUngueltig, AenderungVerweigert);
  void               loescheLagerplatztyp(in ID typ)
                     raises (LagerplatztypUngueltig, LoeschungVerweigert);
  ID                 erstelleLager(in Lagerstruktur l)
                     raises (LagerzahlLimitUeberschritten, LagerUngueltig);
  ID__Seq            leseLagerIDs();
  Lager              leseLager(in ID lager) raises (LagerUngueltig);
  void               aendereLager(in ID lager, in string name)
                     raises (LagerUngueltig, AenderungVerweigert);
  void               loescheLager(in ID lager)
                     raises (LagerUngueltig, LoeschungVerweigert);
  ID                 ergaenzeLagerbereich(in ID lager, in Lagerbereichsstruktur b)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     AenderungVerweigert);
  Lagerbereich       leseLagerbereich(in ID bereich)
                     raises (LagerbereichUngueltig);
  void               aendereLagerbereich(in ID lager, in ID bereich,
                     in string name, in Lagerbereichstyp t, in Lagerstrategie s)
                     raises (LagerbereichUngueltig, AenderungVerweigert);
  void               loescheLagerbereich(in ID lager, in ID bereich)
                     raises (LagerbereichUngueltig, LoeschungVerweigert);
  ID                 ergaenzeLagerplatz(in ID lager, in ID bereich, in Lagerplatz p)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig);
  Lagerplatz         leseLagerplatz(in ID lager, in ID bereich, in ID platz)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig);
  void               aendereLagerplatz(in ID lager, in ID bereich, in ID platz,
                     in string typ, in string gang, in string regal, in string ebene)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig, AenderungVerweigert);
  void               loescheLagerplatz(in ID lager, in ID bereich, in ID platz)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig, LoeschungVerweigert);
  Artikelbestand__Seq leseLagerplatzBestand(in ID lager, in ID bereich, in ID platz)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig);
  void               setzeLagerplatzBestand(in ID lager, in ID bereich,
                     in ID platz, in Artikelbestand b)
                     raises (LagerUngueltig, LagerbereichUngueltig,
                     LagerplatzUngueltig, AenderungVerweigert);
};
```

```
interface ILogistik {
  struct Planposition {
    ID   auftrag;
    ID   artikel;
    ID   lager;
    ID   lagerbereich;
    ID   lagerplatz;
    long menge;
  };

  typedef sequence<Planposition> Planposition__Seq;

  struct Kommissionierplan {
    string            erstellungsdatum;
    Planposition__Seq positionen;
  };

  struct Einlagerungsplan {
    string            erstellungsdatum;
    Planposition__Seq positionen;
  };

  struct Lagerposition {
    ID   artikel;
    long menge;
  };

  typedef sequence<Lagerposition> Lagerposition__Seq;

  struct Warenausgang {
    string             ausfuehrungsdatum;
    Lagerposition__Seq positionen;
  };

  struct Wareneingang {
    string             ausfuehrungsdatum;
    Lagerposition__Seq positionen;
  };

  exception AuftragUngueltig {};
  exception KommissionierplanUngueltig {};
  exception EinlagerungsplanUngueltig {};

  Kommissionierplan erstelleKommissionierplan(in ID auftrag)
                    raises (AuftragUngueltig);
  Einlagerungsplan  erstelleEinlagerungsplan(in ID auftrag)
                    raises (AuftragUngueltig);
  Warenausgang      erfasseAauslagerung(in Kommissionierplan k)
                    raises (KommissionierplanUngueltig);
  Wareneingang      erfasseEinlagerung(in Einlagerungsplan e)
                    raises (EinlagerungsplanUngueltig);
};

interface IBestandsvwt {
  struct Konto {
    ID      artikel;
    Bestand b;
    Bestand mindestbestand;
  };

  enum Aenderungstyp {
    Wareneingang,
    Warenausgang
  };

  struct Bestandsveraenderung {
    ID            konto;
    ID            auftrag;
    Aenderungstyp typ;
    long          menge;
  };

  exception ArtikelUngueltig {};
  exception AuftragUngueltig {};
```

```
  exception ReservierungUngueltig {};
  exception KontoUngueltig {};
  exception KontoVorhanden {};
  exception BestandFehlt {};

  ID      erstelleKonto(in Konto k)
          raises (ArtikelUngueltig, KontoVorhanden);
  void    setzeMindestbestand(in ID konto, in Bestand b)
          raises (KontoUngueltig);
  void    loescheKonto(in ID konto)
          raises (KontoUngueltig, LoeschungVerweigert);
  Bestand aktuellerBestand(in ID konto) raises (KontoUngueltig);
  Bestand physischerBestand(in ID konto) raises (KontoUngueltig);
  ID      reserviere(in ID konto, in Bestand b)
          raises (KontoUngueltig, BestandFehlt);
  Bestand buche(in Bestandsveraenderung b, in ID reservierung)
          raises (KontoUngueltig, ReservierungUngueltig,
          AuftragUngueltig);
  void    loescheReservierung(in ID reservierung)
          raises (ReservierungUngueltig, LoeschungVerweigert);
};

interface IArtikelvwt {
  struct LagerplatzID {
    ID lager;
    ID lagerbereich;
    ID lagerplatz;
  };

  typedef sequence<LagerplatzID> LagerplatzID__Seq

  struct Artikel {
    ID__Seq               artikelnummern;
    long                  laenge;
    long                  breite;
    long                  hoehe;
    long                  volumen;
    long                  bruttogewicht;
    long                  nettogewicht;
    Lagereinheit          einheit;
    LagerplatzID__Seq     lagerort;
  };

  exception ArtikelUngueltig {};

  Artikel     leseArtikel(in ID artikel) raises (ArtikelUngueltig);
};

interface IDatenbank {
  exception VerbindungFehlgeschlagen {string error;};
  exception SQLAusdruckUngueltig {string error;};

  void oeffneVerbindung(in string datenbank)
       raises (VerbindungFehlgeschlagen);
  void beginneTransaktion();
  void verarbeiteSQL(in string sqlAusdruck)
       raises (SQLAusdruckUngueltig);
  void rollback();
  void beendeTransaktion();
  void schliesseVerbindung();
};

eventtype MindestbestandUnterschritten {ID artikel;};

component Lagermanagement {
  provides ILagervwt __ILagervwt;
  provides ILogistik __ILogistik;
  provides IBestandsvwt __IBestandsvwt;
  uses IArtikelvwt __IArtikelvwt;
  uses IDatenbank __IDatenbank;
  publishes MindestbestandUnterschritten _MindestbestandUnterschritten;
};
```

```
  module IBestandsvwt__datamodel {
    struct Auftrag {
      long id;
    };

    struct Bestandsveraenderung {
      Auftrag auftr;
    };

    typedef sequence<Bestandsveraenderung> Bestandsveraenderung__Seq;

    struct Artikel {
      long id;
    };

    struct Konto;

    struct Reservierung {
      long  id;
      long  bestand;
      Konto kto;
    };

    typedef sequence<Reservierung> Reservierung__Seq;

    struct Konto {
      long                     id;
      long                     physischerBestand;
      long                     mindestBestand;
      Reservierung_Seq;        journalReservierungen;
      Bestandsveraenderung_Seq journalVeraenderungen;
    };
  };
};
```

A.4.2 Dienstverträge

```
package Lagerorganisation
context IBestandsvwt::erstelleKonto(k: Konto): ID
  pre : k.artikel <> '' and k.b >= 0 and k.mindestbestand >= 0
  -- Die Kontodaten muessen valide sein
  pre : not IBestandsvwt__datamodel::Konto.allInstances()->exists(konto |
          konto.art.id = k.artikel)
  -- Es darf noch kein Konto fuer den angegebenen Artikel existieren
  post: IBestandsvwt__datamodel::Konto.allInstances()->exists(konto |
          konto.id = result and konto.art.id = k.artikel and
          konto.mindestbestand = k.mindestbestand) and
          IBestandsvwt::physischerBestand(result) = k.b
  -- Ein Konto mit der zurueckgelieferten ID und den angegebenen Daten wurde angelegt
  post: IBestandsvwt::aktuellerBestand(result) =
        IBestandsvwt::pysischerBestand(result)
  -- Der frei verfuegbare Bestand entspricht dem physischen Bestand

context IBestandsvwt::setzeMindestbestand(konto: ID, b: Bestand)
  pre : b >= 0
  -- Der Betrag des Mindestbestands darf nicht kleiner als Null sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: IBestandsvwt__datamodel::Konto.allInstances()->exists(k |
          k.id = konto and k.mindestbestand = b)
  -- Der Mindestbestand wurde eingetragen
context IBestandsvwt::loescheKonto(konto: ID)
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: not IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das Konto wurde geloescht

context IBestandsvwt::aktuellerBestand(konto: ID): Bestand
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
```

```
  post: result >= 0
  -- Der frei verfuegbare Bestand eines Artikels ist nicht kleiner als Null

context IBestandsvwt::physischerBestand(konto: ID): Bestand
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: result >= 0
  -- Der physische Bestand eines Artikels ist nicht kleiner als Null

context IBestandsvwt::reserviere(konto: ID, b: Bestand): ID
  pre : b > 0
  -- Der Betrag der Reservierung muss groesser als Null sein
  pre : b <= IBestandsvwt::aktuellerBestand(konto)
  -- Der Betrag der Reservierung darf nicht groesser als der frei verfuegbare Bestand sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = konto)
  -- Das angegebene Konto muss existieren
  post: IBestandsvwt::aktuellerBestand(konto) =
        IBestandsvwt::aktuellerBestand@pre(konto) - b
  -- Nach der Eintragung der Reservierung wurde der frei verfuegbare
     Bestand des Artikels um den genannten Betrag vermindert
  post: IBestandsvwt::aktuellerBestand(konto) >= 0
  -- Nach der Eintragung der Reservierung ist der frei verfuegbare
     Bestand nicht kleiner als Null
  post: IBestandsvwt__datamodel::Reservierung.allInstances()->exists(reservierung |
          reservierung.id = result and reservierung.bestand = b)
  -- Eine Reservierung mit der zurueckgelieferten ID und der angegebenen
     Menge wurde angelegt
  post: IBestandsvwt::physischerBestand(konto) =
        IBestandsvwt::physischerBestand@pre(konto)
  -- Der physische Bestand des Artikels bleibt unveraendert

context IBestandsvwt::buche(b: Bestandsveraenderung, reservierung: ID): Bestand
  pre : b.menge > 0
  -- Der Betrag der Bestandsveraenderung muss groesser als Null sein
  pre : if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            b.menge <= IBestandsvwt::physischerBestand(b.konto)
          else
            true
        endif
  -- Bei einem Warenausgang darf der Betrag der Bestandsveraenderung
     nicht groesser als der physische Bestand des Artikels sein
  pre : IBestandsvwt__datamodel::Konto.allInstances()->exists(k | k.id = b.konto)
  -- Das angegebene Konto muss existieren
  pre : if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
              r.id = reservierung)
          else
          reservierung = ''
        endif
  -- Bei einem Warenausgang muss die angegebene Reservierung existieren
     und bei einem Wareneingang darf keine Reservierung angegeben werden
  post: result = if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
                   then
                     IBestandsvwt::physischerBestand@pre(b.konto) – b.menge
                   else
                     IBestandsvwt::physischerBestand@pre(b.konto) + b.menge
                 endif
  -- Nach der Buchung der Bestandsveraenderung wurde der physische
     Bestand des Artikels um den genannten Betrag vermindert bzw. erhoeht
  post: result >= 0
  -- Nach der Buchung der Bestandsveraenderung ist der physische Bestand
     nicht kleiner als Null
  post: if b.typ = IBestandsvwt::Aenderungstyp::Warenausgang
          then
            IBestandsvwt::aktuellerBestand(b.konto) =
            IBestandsvwt::aktuellerBestand@pre(b.konto)
          else
            IBestandsvwt::aktuellerBestand(b.konto) =
            IBestandsvwt::aktuellerBestand@pre(b.konto) + b.menge
        endif
  -- Der frei verfuegbare Bestand des Artikels bleibt bei einem Warenausgang unveraendert
     und wird bei einem Wareneingang um den genannten Betrag erhoeht
```

```
context IBestandsvwt::loescheReservierung(reservierung: ID)
  def : konto : IBestandsvwt::ID =
        (IBestandsvwt__datamodel::Reservierung.allInstances()->any(r |
          r.id = reservierung)).kto.id
  def : menge : IBestandsvwt::Bestand =
        (IBestandsvwt__datamodel::Reservierung.allInstances()->any(r |
          r.id = reservierung)).bestand
  pre : IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
          r.id = reservierung)
  -- Die angegebene Reservierung muss existieren
  post: not IBestandsvwt__datamodel::Reservierung.allInstances()->exists(r |
          r.id = reservierung)
  -- Die Reservierung wurde geloescht
  post: aktuellerBestand(konto) = aktuellerBestand@pre(konto) + menge
  -- Der reservierte Bestand wurde dem frei verfuegbaren Bestand hinzugefuegt
endpackage
```

A.4.3 Protokolle

```
protocol for ILogistik {
  states { Lageraktivitaet startstate finalstate; };
  transitions {
    Lageraktivitaet->Lageraktivitaet with erstelleKommissionierplan;
    Lageraktivitaet->Lageraktivitaet with erstelleEinlagerungsplan;
    Lageraktivitaet->Lageraktivitaet with erfasseAuslagerung;
    Lageraktivitaet->Lageraktivitaet with erfasseEinlagerung;
  };
};

protocol for IBestandsvwt {
  states { Bestandsverwaltung startstate finalstate; };
  transitions {
    Bestandsverwaltung->Bestandsverwaltung with erstelleKonto;
    Bestandsverwaltung->Bestandsverwaltung with setzeMindestbestand;
    Bestandsverwaltung->Bestandsverwaltung with loescheKonto;
    Bestandsverwaltung->Bestandsverwaltung with aktuellerBestand;
    Bestandsverwaltung->Bestandsverwaltung with physischerBestand;
    Bestandsverwaltung->Bestandsverwaltung with reserviere;
    Bestandsverwaltung->Bestandsverwaltung with buche;
    Bestandsverwaltung->Bestandsverwaltung with loescheReservierung;
  };
};

protocol EinfacheDatenverarbeitung {
  states {
    Start startstate;
    Datenbankverbindung;
    Ende finalstate;
  };
  transitions {
    Start->Datenbankverbindung with IDatenbank::oeffneVerbindung;
    Datenbankverbindung->Datenbankverbindung with IDatenbank::verarbeiteSQL;
    Datenbankverbindung->Ende with IDatenbank::schliesseVerbindung;
  };
};

protocol KomplexeDatenverarbeitung {
  states {
    Start startstate;
    Datenbankverbindung;
    Transaktionsverarbeitung;
    Ende finalstate;
  };
  transitions {
    Start->Datenbankverbindung with oeffneVerbindung;
    Datenbankverbindung->Transaktionsverarbeitung with
      IDatenbank::beginneTransaktion and mean 0.6;
    Transaktionsverarbeitung->Transaktionsverarbeitung with
      IDatenbank::verarbeiteSQL and mean 0.6;
    Transaktionsverarbeitung->Transaktionsverarbeitung with
```

```
      IDatenbank::rollback and mean 0.1;
    Transaktionsverarbeitung->Datenbankverbindung with
      IDatenbank::beendeTransaktion and mean 0.3;
    Datenbankverbindung->Ende with
      IDatenbank::schliesseVerbindung and mean 0.4;
  };
};

protocol ArtikelabfrageDatenverarbeitung {
  states {
    Start startstate;
    Artikelverwaltung;
    Datenbankverbindung;
    Transaktionsverarbeitung;
    Ende finalstate;
  };
  transitions {
    Start->Artikelverwaltung with leseArtikel;
    Artikelverwaltung->Datenbankverbindung with oeffneVerbindung;
    Datenbankverbindung->Transaktionsverarbeitung with
      IDatenbank::beginneTransaktion and mean 0.6;
    Transaktionsverarbeitung->Transaktionsverarbeitung with
      IDatenbank::verarbeiteSQL and mean 0.6;
    Transaktionsverarbeitung->Transaktionsverarbeitung with
      IDatenbank::rollback and mean 0.1;
    Transaktionsverarbeitung->Datenbankverbindung with
      IDatenbank::beendeTransaktion and mean 0.3;
    Datenbankverbindung->Ende with
      IDatenbank::schliesseVerbindung and mean 0.4;
  };
};
```

A.4.4 Komponentenvertrag

```
package Lagerorganisation
context Lagermanagement
  ILagervwt::erstelleLagerplatztyp     requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerplatztyp         requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerplatztyp      requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerplatztyp      requires KomplexeDatenverarbeitung
  ILagervwt::erstelleLager             requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerIDs              requires EinfacheDatenverarbeitung
  ILagervwt::leseLager                 requires EinfacheDatenverarbeitung
  ILagervwt::aendereLager              requires KomplexeDatenverarbeitung
  ILagervwt::loescheLager              requires KomplexeDatenverarbeitung
  ILagervwt::ergaenzeLagerbereich      requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerbereich          requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerbereich       requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerbereich       requires KomplexeDatenverarbeitung
  ILagervwt::ergaenzeLagerplatz        requires EinfacheDatenverarbeitung
  ILagervwt::leseLagerplatz            requires EinfacheDatenverarbeitung
  ILagervwt::aendereLagerplatz         requires KomplexeDatenverarbeitung
  ILagervwt::leseLagerplatzbestand     requires EinfacheDatenverarbeitung
  ILagervwt::setzeLagerplatzbestand    requires KomplexeDatenverarbeitung
  ILagervwt::loescheLagerplatz         requires KomplexeDatenverarbeitung
  ILogistik::erstelleKommissionierplan requires KomplexeDatenverarbeitung
  ILogistik::erstelleEinlagerungsplan  requires ArtikelabfrageDatenverarbeitung
  ILogistik::erfasseAuslagerung        requires KomplexeDatenverarbeitung
  ILogistik::erfasseEinlagerung        requires KomplexeDatenverarbeitung
  IBestandsvwt::erstelleKonto          requires KomplexeDatenverarbeitung
  IBestandsvwt::setzeMindestbestand    requires EinfacheDatenverarbeitung
  IBestandsvwt::loescheKonto           requires KomplexeDatenverarbeitung
  IBestandsvwt::aktuellerBestand       requires EinfacheDatenverarbeitung
  IBestandsvwt::physischerBestand      requires EinfacheDatenverarbeitung
  IBestandsvwt::reserviere             requires KomplexeDatenverarbeitung
  IBestandsvwt::buche                  requires KomplexeDatenverarbeitung
  IBestandsvwt::loescheReservierung    requires EinfacheDatenverarbeitung
endpackage
```

A.4.5 Entwurfszusammenhang

Begriff	Art	Architekturbestandteil	Art
Artikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Lagerartikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Verkaufsartikel	Infoobjekt	Lagerorganisation::IArtikelvwt::Artikel	Typ
Bestandskonto	Infoobjekt	Lagerorganisation::IBestandsvwt::Konto	Typ
Bestandsveränderung	Infoobjekt	Lagerorganisation::IBestandsvwt::Bestandsveraenderung	Typ
Lager	Infoobjekt	Lagerorganisation::ILagervwt::Lager,	Typ
		Lagerorganisation::ILagervwt::Lagerstruktur	Typ
Lagerbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereich,	Typ
		Lagerorganisation::ILagervwt::Lagerbereichsstruktur	Typ
Kommissionierbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Reservebereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Wareneingangsbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Warenausgangsbereich	Infoobjekt	Lagerorganisation::ILagervwt::Lagerbereichstyp	Typ
Lagerplatz	Infoobjekt	Lagerorganisation::ILagervwt::Lagerplatz	Typ
Einlagerungsplan	Infoobjekt	Lagerorganisation::ILogistik::Einlagerungsplan	Typ
Kommissionierplan	Infoobjekt	Lagerorganisation::ILogistik::Kommissionierplan	Typ
Planposition	Infoobjekt	Lagerorganisation::ILogistik::Planposition	Typ
Wareneingang	Infoobjekt	Lagerorganisation::ILogistik::Wareneingang	Typ
Warenausgang	Infoobjekt	Lagerorganisation::ILogistik::Warenausgang	Typ
Bestand prüfen	Funktion	Lagerorganisation::IBestandsvwt::aktuellerBestand	Methode
PhysischenBestand prüfen	Funktion	Lagerorganisation::IBestandsvwt::physischerBestand	Methode
Bestand reservieren	Funktion	Lagerorganisation::IBestandsvwt::reserviere	Methode
Reservierung löschen	Funktion	Lagerorganisation::IBestandsvwt::loescheReservierung	Methode
Bestandsveränderung buchen	Funktion	Lagerorganisation::IBestandsvwt::buche	Methode
Lagerartikel auslagern	Prozess	Lagerorganisation::IBestandsvwt,	Protokoll
		Lagerorganisation::ILogistik	Protokoll
Kommissionierplan erstellen	Funktion	Lagerorganisation::ILogistik::erstelleKommissionierplan	Methode
Lagerartikel kommissionieren	Funktion	--	
Lagerartikel erfassen	Funktion	Lagerorganisation::ILogistik::erfasseAuslagerung	Methode
Warenausgang buchen	Prozess	Lagerorganisation::IBestandsvwt	Protokoll
Bestandsveränderung ermitteln	Funktion	--	
Nachschubauftrag veranlassen	Funktion	Lagerorganisation::MindestbestandUnterschritten	Ereignis

A.5 Grey Pages

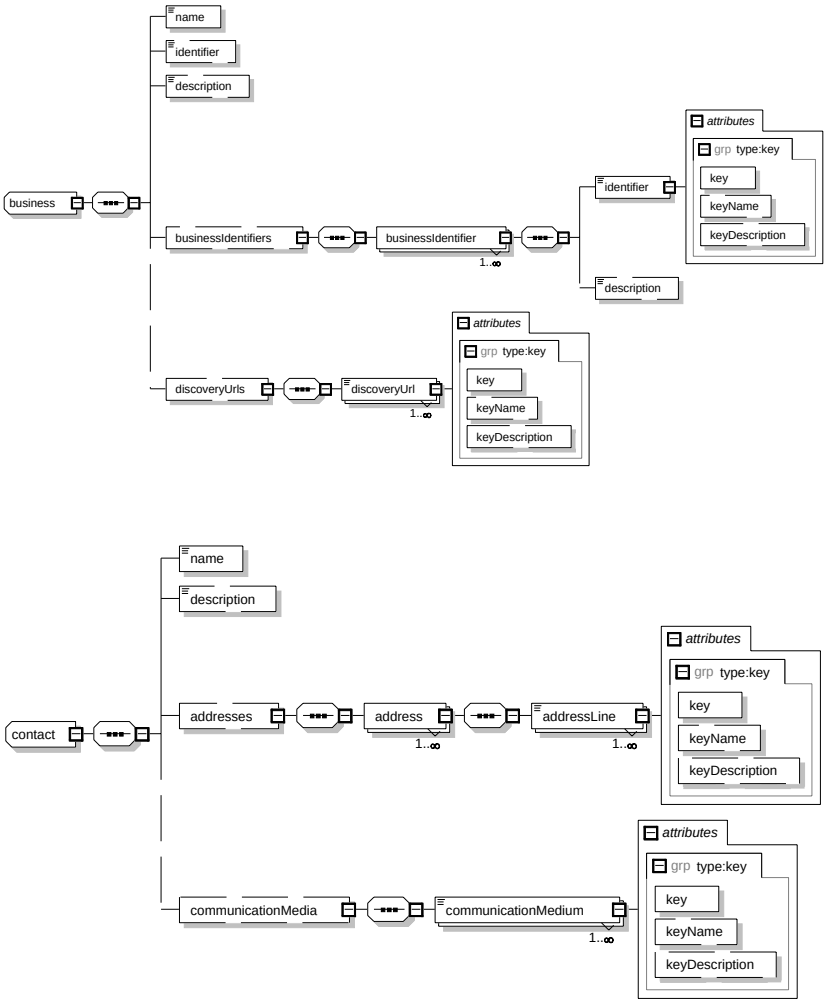
```
type Efficiency = contract {
   responseTime : decreasing numeric s;
   throughput : increasing numeric calls/s;
   memoryUtilization : decreasing numeric b;
   discUtilization : decreasing numeric b;
};

BestandsvwtNormallast for IBestandsvwt = profile {
   from aktuellerBestand, physischerBestand require Efficiency contract {
      responseTime {mean < 0.5 s;};
      throughput {mean > 60 calls/s; percentile 80 > 50 calls/s;};
   };
   from buche, reserviere, loescheReservierung require Efficiency contract {
      responseTime {mean < 1 s;};
      throughput {mean > 20 calls/s; percentile 80 > 15 calls/s;};
   };
};

DatenbankNormallast for IDatenbank = profile {
   from oeffneVerbindung, schliesseVerbindung, beginneTransaktion, beendeTransaktion
      require Efficiency contract {
         responseTime {mean < 0.05 s;};
         throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
      };
   from verarbeiteSQL, rollback require Efficiency contract {
      responseTime {mean < 0.2 s;};
      throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
   };
};
```

```
BestandsvwtHochlast for IBestandsvwt = profile {
   from aktuellerBestand, physischerBestand require Efficiency contract {
      responseTime {mean < 0.25 s;};
      throughput {mean > 80 calls/s; percentile 80 > 70 calls/s};
   };
   from buche, reserviere, loescheReservierung require Efficiency contract {
      responseTime {mean < 0.5 s;};
      throughput {mean > 30 calls/s; percentile 80 > 25 calls/s};
   };
};

DatenbankHochlast for IDatenbank = profile {
   from oeffneVerbindung, schliesseVerbindung, beginneTransaktion, beendeTransaktion
      require Efficiency contract {
         responseTime {mean < 0.025 s;};
         throughput {mean > 100 calls/s; percentile 80 > 80 calls/s;};
      };
   from verarbeiteSQL, rollback require Efficiency contract {
      responseTime {mean < 0.1 s;};
      throughput {mean > 200 calls/s; percentile 80 > 180 calls/s;};
   };
};

LagermanagementNormallast for Lagermanagement = servicelevel {
   require BestandsvwtNormallast, DatenbankNormallast;
};

LagermanagementHochlast for Lagermanagement = servicelevel {
   require BestandsvwtHochlast, DatenbankHochlast;
};
```

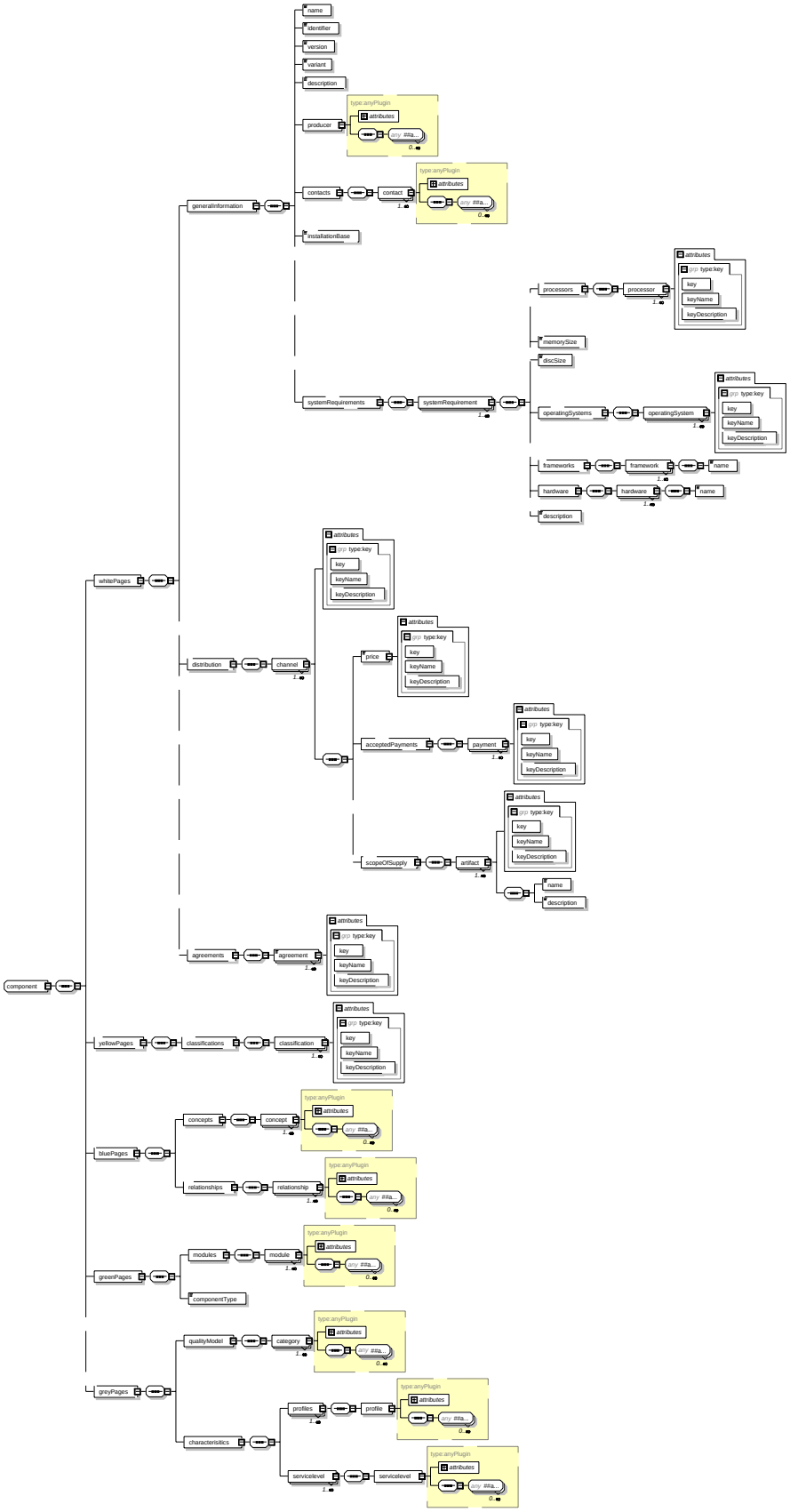
B UNSCOM/X Format

In diesem Abschnitt werden die XML Schemata, die die Formatvorgaben für das UNSCOM/X Format enthalten, graphisch dargestellt. Sie geben die Struktur der XML Dokumente vor, die zum Austausch von Komponentenspezifikationen (zwischen Werkzeugen) zu erzeugen sind. Aus Gründen der Modularität wurden mehrere ineinander verschachtelte XML Schemata definiert, die jeweils Vorgaben für einen bestimmten Spezifikationsteil enthalten. Dadurch wird es möglich, bei Bedarf einzelne Fragmente einer Komponentenspezifikation auszutauschen, so dass nicht jedes Mal der gesamte Inhalt zu übertragen ist.

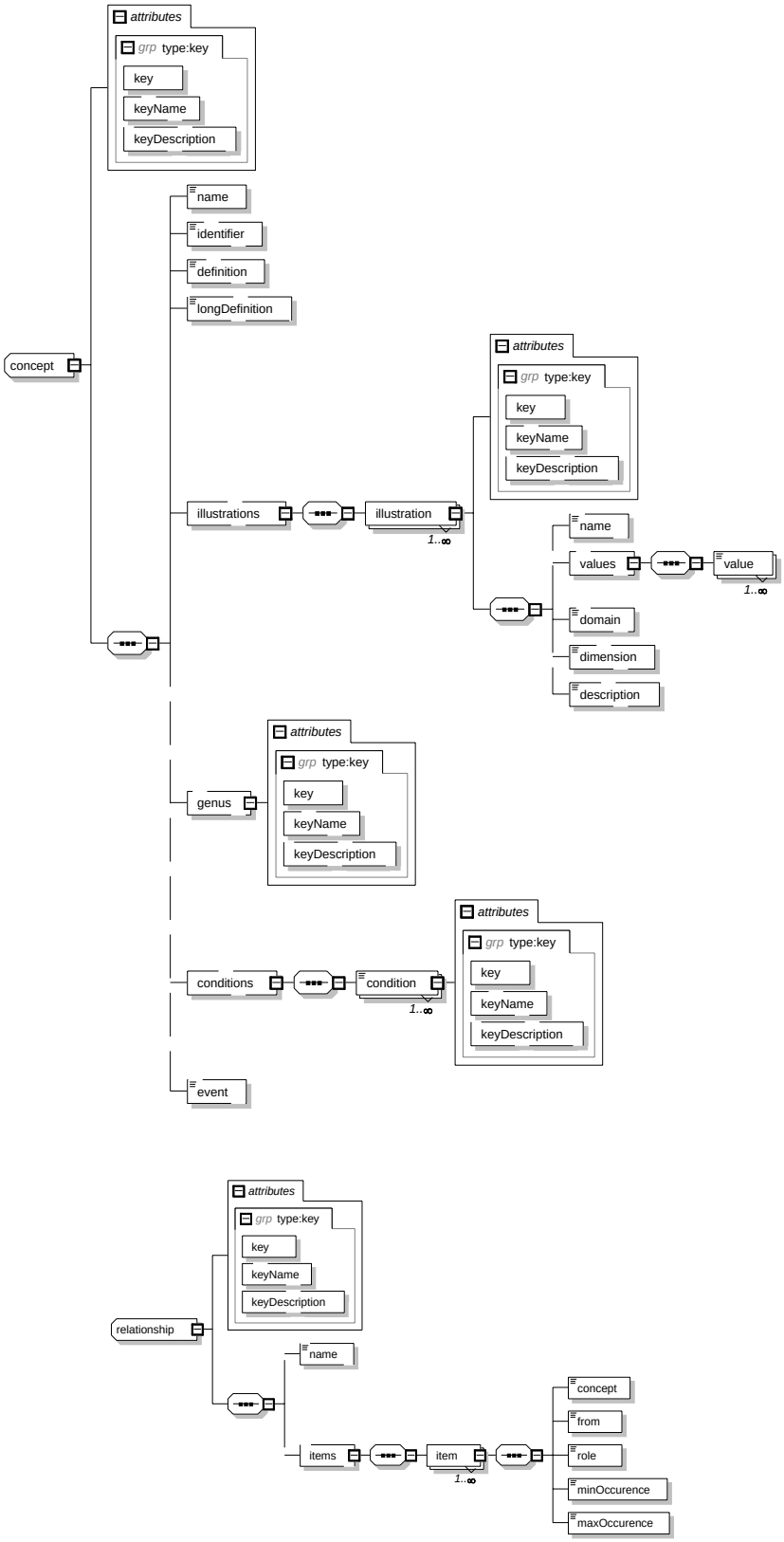
B.1 Anbieter and Ansprechpartner



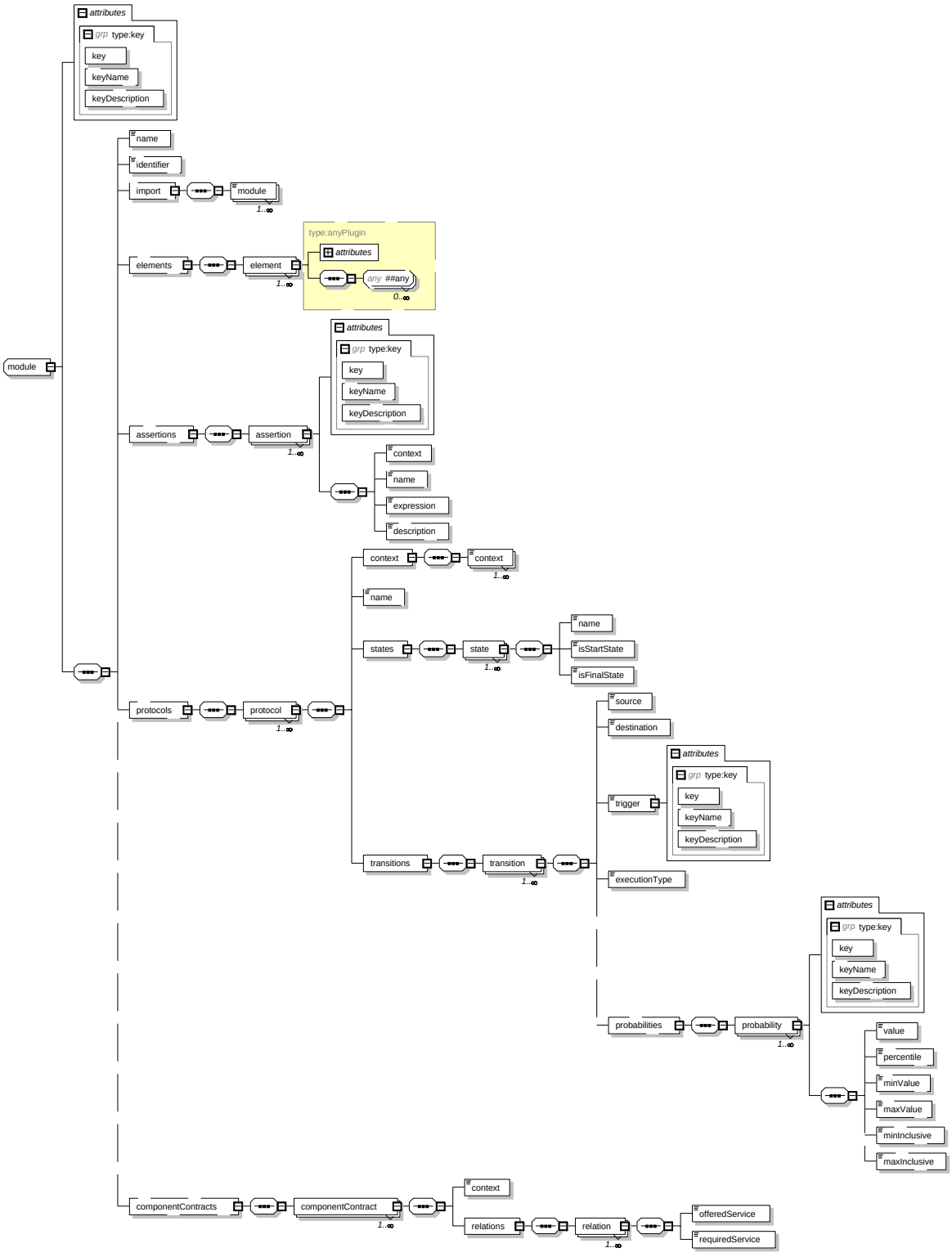
B.2 Komponente

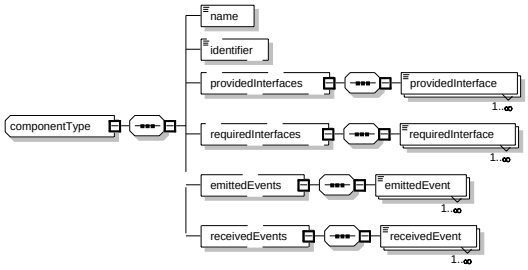


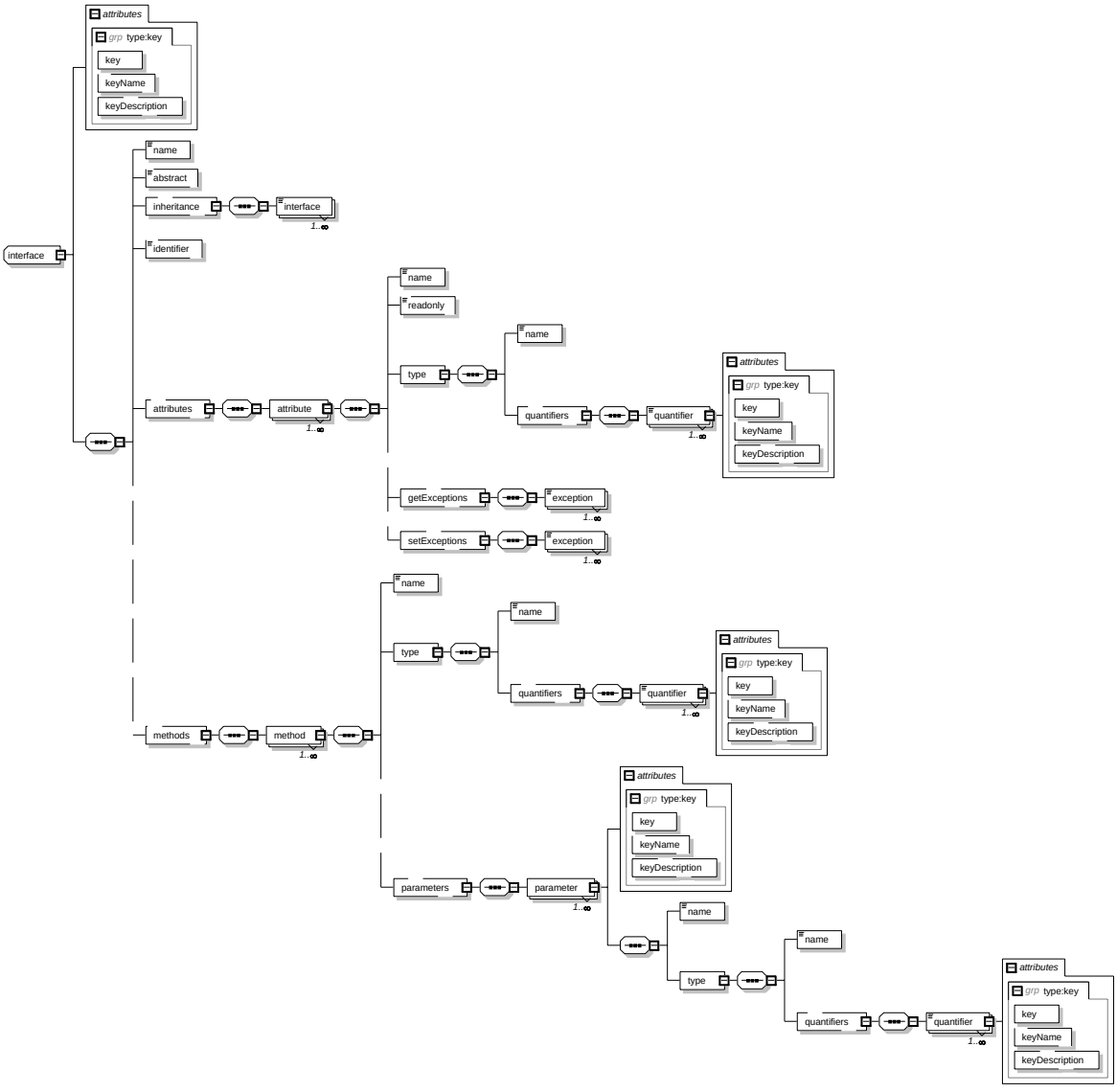
B.3 Begriffe und Aussagen

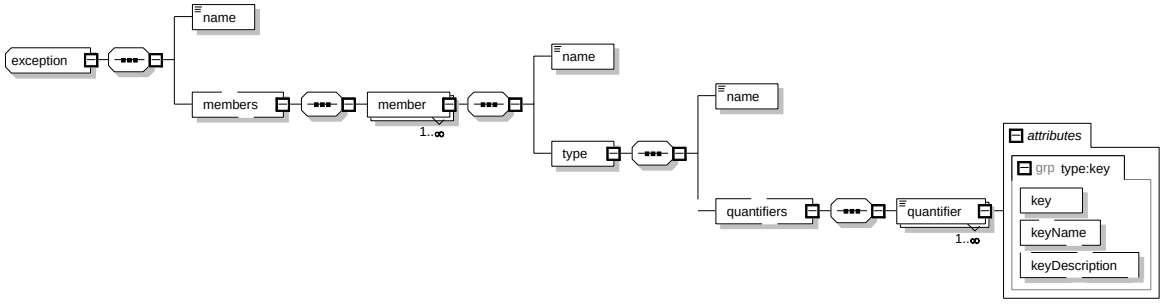


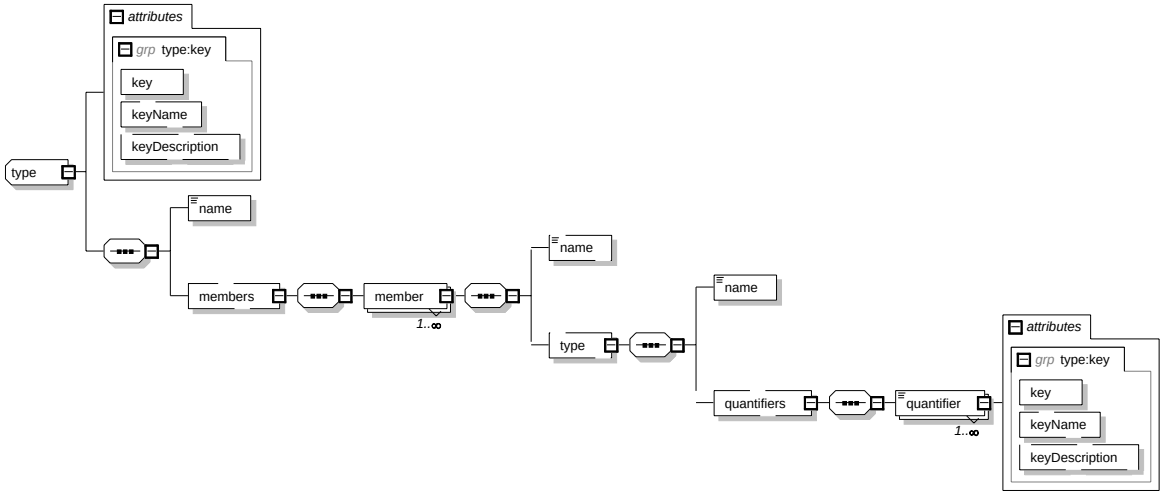
B.4 Architekturmerkmale

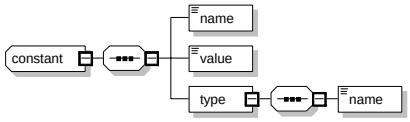



componentType
name
identifier
providedInterfaces
providedInterface
requiredInterfaces
requiredInterface
emittedEvents
emittedEvent
receivedEvents
receivedEvent
1..∞

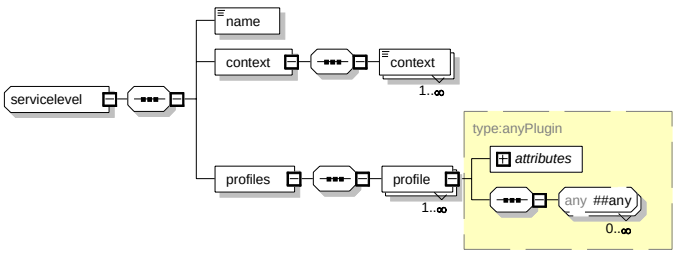

interface
attributes
grp type:key
key
keyName
keyDescription
name
abstract
inheritance
interface
identifier
attributes
attribute
name
readonly
type
name
quantifiers
quantifier
getExceptions
exception
setExceptions
exception
methods
method
name
type
name
quantifiers
quantifier
parameters
parameter
name
type
name
quantifiers
quantifier
1..∞

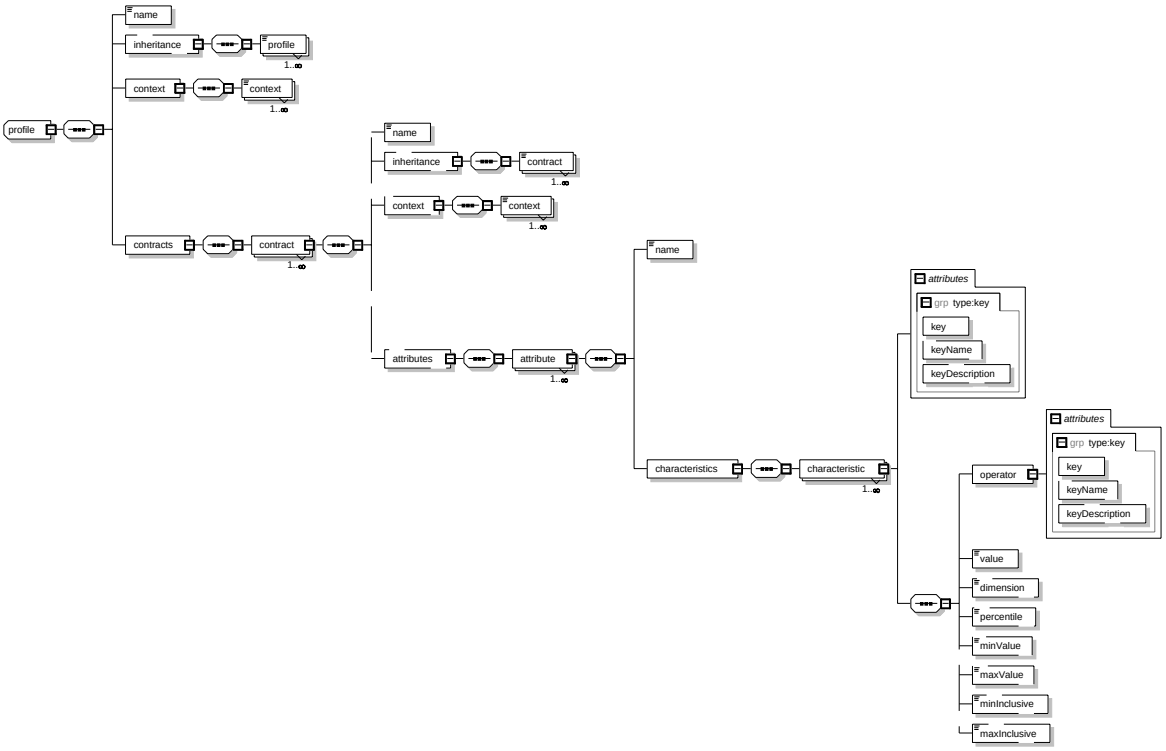


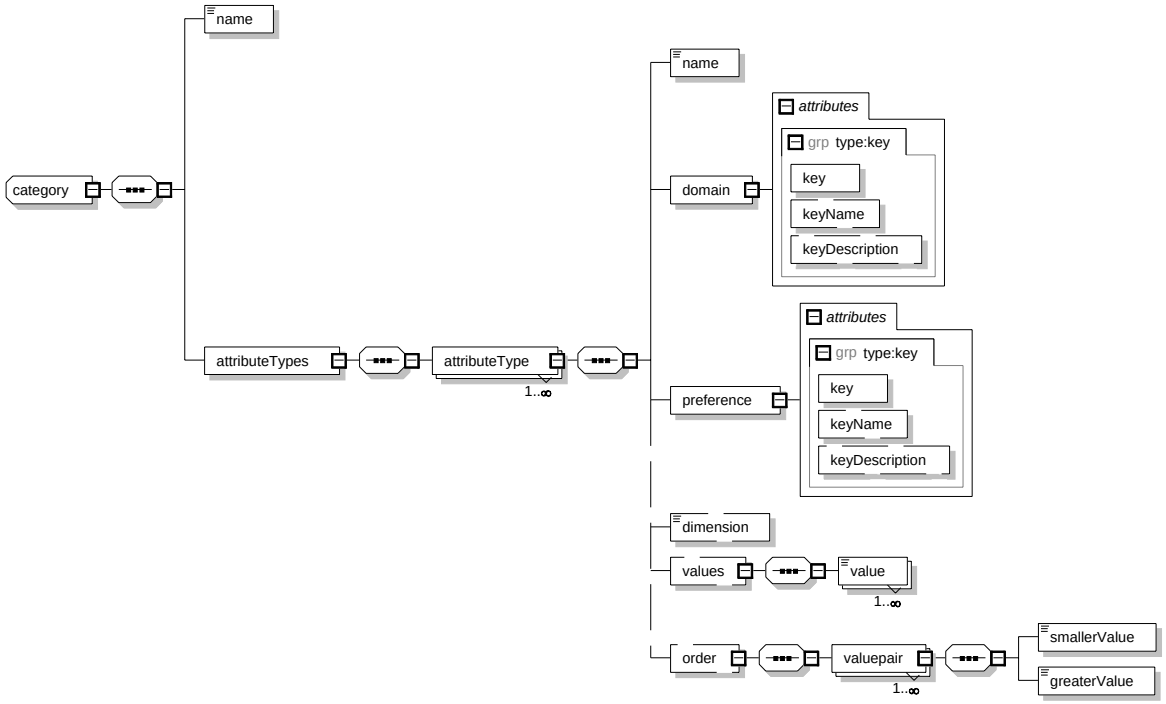




B.5 Qualitätseigenschaften




profile
name
inheritance
profile
context
context
contracts
contract
name
inheritance
contract
context
context
attributes
attribute
name
characteristics
characteristic
attributes
grp type:key
key
keyName
keyDescription
operator
attributes
grp type:key
key
keyName
keyDescription
value
dimension
percentile
minValue
maxValue
minInclusive
maxInclusive


category
name
attributeTypes
attributeType
name
domain
attributes
grp type:key
key
keyName
keyDescription
preference
attributes
grp type:key
key
keyName
keyDescription
dimension
values
value
order
valuepair
smallerValue
greaterValue

C UNSCOM Taxonomien

Nachfolgend sind die mit dem UNSCOM Spezifikationsrahmen eingeführten Taxonomien dargestellt, die zur Klassierung im Rahmen der Yellow Pages verwendet werden können. Die Struktur der darüber hinaus einzusetzenden standardisierten Taxonomien UNSPSC und NAICS kann auf den Internetseiten des jeweiligen Standardisierungsgremiums eingesehen werden. Hinzuweisen bleibt noch darauf, dass die Erarbeitung der letzten, zur Klassifikation von Anwendungsbereichen genutzten Taxonomie derzeit nicht abgeschlossen ist. Für diese Taxonomie wird eine sukzessive Vervollständigung im Rahmen eines Standardisierungsprojekts angestrebt.

```
Component Model Classification System
  Microsoft .NET
    Microsoft .NET 1.0
    Microsoft .NET 1.1
  Sun EJB
    Sun EJB 1.0
  XML Web Services
    XML Web Services 1.0 (SOAP 1.1, WSDL 1.0)
  Microsoft COM
  OMG CORBA Components
    OMG CORBA Component Model 1.0


Conceptual Component Type Classification System
  Application-Specific Component
  Generic Component
  Component Application Framework
  Component System Framework
  Application


Reuse Type Classification System
  Logical Reuse
  Physical Reuse


Application Domain Classification System
  Generic Domains
    Data Management
    Workflow Management
    Expert System
    Security Management
    Communication Management
  Application-Specific Domains
    Business Administration
      BookKeeping
      Reporting
      Organizational Management
      Decision Support
    Economics
    Bioinformatics
    … (to be standardized)
```

Literaturverzeichnis

Aagedal, J. Ø. (2001): *Quality of Service Support in Distributed Systems*. Doctoral Dissertation, University of Oslo. Oslo.

Ackermann, J., Brinkop, F., Conrad, S., Fettke, P., Frick, A., Glistau, E., Jaekel, H., Kotlar, O., Loos, P., Mrech, H., Ortner, E., Overhage, S., Raape, U., Sahm, S., Schmietendorf, A., Teschke, T. und Turowski, K. (2002): *Standardized Specification of Business Components*. Technical Report, German Society of Computer Science.

Albani, A., Keiblinger, A., Turowski, K. und Winnewisser, C. (2003): *Komponentenmodell für die strategische Lieferkettenentwicklung*. In: Uhr, W., Esswein, W. und Schoop, E. (Hrsg.): Wirtschaftsinformatik 2003, Medien, Märkte, Mobilität, Band II, Dresden, Germany, Physica: 61-80.

Allen, P. und Frost, S. (1998): *Component-Based Development for Enterprise Systems: Applying the SELECT Perspective*. Cambridge University Press, Cambridge, MA.

Apperly, H., Booch, G., Councill, W. T., Griss, M., Heineman, G. T., Jacobson, I., Latchem, S., McGibbon, B., Norris, D. und Poulin, J. (2001): *The Near-Term Future of Component-Based Software Engineering*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 753-774.

Aristoteles (300 v. Chr.): *Metaphysik (Übersetzte Auflage)*. Rowohlt, Reinbeck.

Atkinson, C., Gutheil, M. und Hummel, O. (2004): *Komponentenorientierter Entwurf von PIMs und CIMs mit der KobrA-Methode*. In: Turowski, K. (Hrsg.): Architekturen, Komponenten, Anwendungen (AKA 2004), Augsburg, Germany, Lecture Notes in Informatics Nr. P-57, Köllen, Bonn: 93-109.

Bachmann, F., Bass, L., Buhman, C., Cornella-Dorda, S., Long, F., Robert, J., Seacord, R. und Wallnau, K. C. (2000): *Technical Concepts of Component-Based Software Engineering, Second Edition*. Technical Report Nr. CMU/SEI-2000-TR-008, Software Engineering Institute, Carnegie Mellon University.

Backus, J. (1978): *Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs*. In: Communications of the ACM (21) 8: 613-641.

Balzert, H. (1998): *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum, Heidelberg.

Balzert, H. (1999): *Lehrbuch der Objektmodellierung. Analyse und Entwurf*. Spektrum, Berlin, Heidelberg.

Bass, L., Clements, P. und Kazman, R. (1998): *Software Architecture in Practice*. Addison-Wesley, Upper Saddle River, NJ.

Baster, G., Konana, P. und Scott, J. E. (2001): *Business Components - A Case Study of Bankers Trust Australia Limited*. In: Communications of the ACM (44) 3: 92-98.

Becker, C. und Geihs, K. (1999): *Generic QoS Specifications for COBRA*. In: Steinmetz, R. (Hrsg.): Kommunikation in Verteilten Systemen, ITG/GI-Fachtagung, Darmstadt, 1999, Darmstadt, Germany, Springer, Berlin, Heidelberg: 184-195.

Becker, C. und Geihs, K. (2000): *Generic QoS-Support for CORBA*. In: IEEE (Hrsg.): Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000), Antibes, France, IEEE Computer Society Press, Los Alamitos (CA): 60-65.

Becker, J. und Schütte, R. (2004): *Handelsinformationssysteme*. 2. Aufl., Verlag Moderne Industrie, Frankfurt.

Becker, J., Uhr, W. und Vering, O. (2000): *Integrierte Informationssysteme in Handelsunternehmen auf der Basis von SAP-Systemen*. Springer, Berlin, Heidelberg.

Becker, S., Firus, V., Giesecke, S., Hasselbring, W., Overhage, S. und Reussner, R. (2004): *Towards a Generic Framework for Evaluating Component-Based Software Architectures*. In: Turowski, K. (Hrsg.): Architekturen, Komponenten, Anwendungen (AKA 2004), Augsburg, Germany, Lecture Notes in Informatics Nr. P-57, Köllen: 163-180.

Bertalanffy, L. v. (1976): *General System Theory*. George Braziller, New York, NY.

Bertoa, M. und Vallecillo, A. (2002): *Quality Attributes for COTS Components*. In: Brito e Abreu, F., Piattini, M. und Poels, G. (Hrsg.): Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002), Malaga, Spain.

Beugnard, A., Jézéquel, J.-M., Plouzeau, N. und Watkins, D. (1999): *Making Components Contract Aware*. In: IEEE Computer (32) 7: 38-45.

Blevins, D. (2001): *Overview of the Enterprise JavaBeans Component Model*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 589-606.

Boehm, B. W. (1988): *A Spiral Model of Development and Enhancement*. In: IEEE Computer (21) 5: 61-72.

Boehm, B. W. und Abts, C. (1999): *COTS Integration: Plug and Pray?* In: IEEE Computer (32) 1: 135-138.

Böhm, C. und Jacopini, G. (1966): *Flow Diagrams, Turing Machines, and Languages with only two Formation Rules*. In: Communications of the ACM (9) 5: 366-371.

Booch, G. (1993): *Object-Oriented Analysis and Design. With Applications*. Addison-Wesley, Reading, MA.

Booch, G., Jacobson, I. und Rumbaugh, J. (1999): *The Unified Modeling Language User Guide*. Addison-Wesley, Upper Saddle River, NJ.

Bosch, J. (2000): *Design and Use of Software Architectures. Adopting and Evolving a Product-Line Approach*. Addison-Wesley, Reading, MA.

Bosch, J., Szyperski, C. und Weck, W. (2003): *Component-Oriented Programming*. In: Buschmann, F., Buchmann, A. P. und Cilia, M. (Hrsg.): Object-Oriented Technology: ECOOP 2003 Workshop Reader, ECOOP 2003 Workshops, July 21-25, 2003, Final Reports, Darmstadt, Germany, Lecture Notes in Computer Science Nr. 3013, Springer, Berlin, Heidelberg: 34-49.

Brada, P. (2001): *Towards Automated Component Compatibility Assessment*. In: Bosch, J., Szyperski, C. und Weck, W. (Hrsg.): 6th International Workshop on Component-Oriented Programming (WCOP'01), Budapest, Hungary.

Brown, A. W. (2000): *Large-Scale, Component-Based Development*. Prentice Hall, Upper Saddle River, NJ.

Brownsword, L., Oberndorf, T. und Sledge, C. A. (2000): *Developing New Processes for COTS-Based Systems*. In: IEEE Software (17) 4: 48-55.

Broy, M. (1997): *Towards a Mathematical Concept of a Component and its Use*. In: Software - Concepts and Tools (18) 3: 137-159.

Broy, M. und Rombach, D. (2002): *Software Engineering: Wurzeln, Stand und Perspektiven*. In: Informatik Spektrum (16) 6: 438-451.

Broy, M. und Siedersleben, J. (2002): *Objektorientierte Programmierung und Softwareentwicklung. Eine kritische Einschätzung*. In: Informatik Spektrum (1) 25: 3-11.

Budde, R., Kautz, K., Kuhlenkamp, K. und Züllighoven, H. (1992): *Prototyping: An Approach to Evolutionary System Development*. Springer, Berlin, Heidelberg.

Bunge, M. (1977): *Treatise on Basic Philosophy. Vol. 3: Ontology I: The Furniture of the World*. Reidel, Boston, MA.

Cardelli, L. und Wegner, P. (1985): *On Understanding Types, Data Abstraction, and Polymorphism*. In: ACM Computing Surveys (17) 4: 471-521.

Carrière, J. (1998): *Architecture Description Languages*. In: Bass, L., Clements, P. und Kazman, R. (Hrsg.): Software Architecture in Practice. Addison-Wesley, Reading, MA: 267-284.

Cauldwell, P., Chawla, R., Chopra, V., Damschen, G., Dix, C., Hong, T., Norton, F., Ogbuji, U., Olander, G., Richman, M. A., Saunders, K. und Zaev, Z. (2001): *Professional XML Web Services*. Wrox Press, Birmingham.

Cerami, E. (2002): *Web Services Essentials*. O'Reilly, Sebastopol, CA.

Chambers, R. A. (1996): *Finding Reusable Components: By Good Look or Good Management?* In: Sarshar, M. (Hrsg.): Systematic Reuse: Issues in Initiating and Improving a Reuse Program: International Workshop on Systematic Reuse, Liverpool: 70-84.

Cheesman, J. und Daniels, J. (2001): *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, Upper Saddle River, NJ.

Clements, P., Kazman, R. und Klein, M. (2001): *Evaluating Software Architectures. Methods and Case Studies*. Addison-Wesley, Upper Saddle River, NJ.

Clements, P. C., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. und Stafford, J. (2003): *Documenting Software Architectures*. Addison-Wesley, Upper Saddle River, NJ.

Conrad, S. und Turowski, K. (2000): *Vereinheitlichung der Spezifikation von Fachkomponenten auf der Basis eines Notationsstandards*. In: Ebert, J. und Frank, U. (Hrsg.): Workshop "Modellierung 2000" - Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, St. Goar, Germany, Fölbach, Koblenz: 179-194.

Cook, S. und Daniels, J. (1994): *Designing Object Systems: Object-Oriented Modelling with Syntropy*. Prentice Hall, Englewood Cliffs, NJ.

Cox, B. J. (1987): *Object-Oriented Programming - An Evolutionary Approach*. Addison-Wesley, Reading, MA.

Crnkovic, I. (2002): *Component-Based Software Engineering - New Challenges in Software Development*. In: Software Focus (2) 4: 127-133.

Czarnecki, K. und Eisenecker, U. W. (2000): *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Upper Saddle River, NJ.

Dahl, O. J. und Nygaard, K. (1966): *Simula - an ALGOL-Based Simulation Language*. In: Communications of the ACM (9) 9: 671-678.

Daniel, J., Traverson, B. und Vignes, S. (1999): *Integration of Quality of Service in Distributed Object Systems*. In: IFIP (Hrsg.): Proceedings of IFIP TC6 WG6.1 Second International Working Conference on Distributed Applications and Interoperable Systems (DAIS'99), Helsinki, Finnland: 31-43.

Davis, A. M. (1990): *The Analysis and Specification of Systems and Software Requirements*. In: Dorfman, M. und Thayer, R. (Hrsg.): Tutorial: Systems and Software Requirements Engineering. IEEE Computer Society Press, Los Alamitos, CA: 119-144.

Davis, A. M. (1993): *Software Requirements: Objects, Functions, and States*. Prentice Hall, Englewood Cliffs, NJ.

Davis, A. M., Bersoff, E. H. und Comer, E. R. (1988): *A Strategy for Comparing Alternative Software Development Life Cycle Models*. In: IEEE Transactions on Software Engineering (14) 10: 1453-1461.

DeMarco, T. (1978): *Structured Analysis and Systems Specification*. Yourdon Press, New York, NY.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J. und Young, P. R. (1989): *Computing as a Discipline*. In: IEEE Computer (22) 2: 63-70.

DeRemer, F. und Kron, H. H. (1976): *Programming-in-the-Large Versus Programming-in-the-Small*. In: IEEE Transactions on Software Engineering (2) 2: 80-86.

Dijkstra, E. W. (1972): *Notes on Structured Programming*. In: Dahl, O. J., Dijkstra, E. W. und Hoare, C. A. W. (Hrsg.): Structured Programming. Academic Press, London: 1-82.

Dowson, M. (1987): *Iteration in the Software Process. Review of the 3rd International Software Process Workshop*. In: 9th International Conference on Software Engineering, Monterey, Washington, USA, IEEE Computer Society Press, Los Alamitos, CA: 36-39.

D'Souza, D. F. und Wills, A. C. (1999): *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, Upper Saddle River, NJ.

Dumitrascu, N., Murphy, S. und Murphy, L. (2003): *A Methodology for Predicting the Performance of Component-Based Applications*. In: Bosch, J., Szyperski, C. und Weck, W. (Hrsg.): 8th International Workshop on Component-Oriented Programming (WCOP'03), Darmstadt, Germany.

Eberhard, K. (1999): *Einführung in die Erkenntnis- und Wissenschaftstheorie. Geschichte und Praxis der konkurrierenden Erkenntniswege*. 2. Aufl., Kohlhammer, Stuttgart, Berlin, Köln.

Ewald, T. (2001): *Overview of COM+*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 573-588.

Fellner, K., Rautenstrauch, C. und Turowski, K. (1999): *Fachkomponenten zur Gestaltung betrieblicher Anwendungssysteme*. In: Information Management & Consulting (14) 2: 25-34.

Ferstl, O. K. und Sinz, E. J. (1998): *Grundlagen der Wirtschaftsinformatik. Band 1*. 3. Aufl., Oldenbourg, München, Germany.

Ferstl, O. K., Sinz, E. J., Hammel, C., Schlitt, M. und Wolf, S. (1997): *Bausteine für komponentenbasierte Anwendungssysteme*. In: HMD (197) 34: 24-46.

Flynt, J. und Desai, M. (2001): *The Future of Software Components: Standards and Certification*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 693-708.

Frakes, W. B. und Pole, T. P. (1994): *An Empirical Study of Representation Methods for Reusable Software Components*. In: IEEE Transactions on Software Engineering (20) 8: 617-630.

Frank, U. (1999): *Componentware - Software-technische Konzepte und Perspektiven für die Gestaltung betrieblicher Informationssysteme*. In: Information Management & Consulting (14) 2: 11-18.

Frege, G. (1975): *Funktion, Begriff, Bedeutung. Fünf logische Studien. Hrsg. v. G. Patzig. Kleine Vandenhoeck-Reihe Nr. 1144*. Vandenhoeck Ruprecht, Göttingen.

Frolund, S. und Koistinen, J. (1998): *QML: A Language for Quality of Service Specification*. Nr. HPL-98-10, Hewlett-Packard Laboratories.

Gamma, E., Helm, R., Johnson, R. und Vlissides, J. (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

Garlan, D., Allen, R. und Ockerbloom, J. (1995): *Architectural Mismatch: Why Reuse is So Hard*. In: IEEE Software (12) 6: 17-26.

Giesecke, M. und Rappe-Giesecke, K. (1997): *Supervision als Medium kommunikativer Sozialforschung*. Suhrkamp, Frankfurt am Main.

Goldberg, A. und Robson, D. (1983): *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, Reading, MA.

Graham, I. (1994): *Migrating to Object Technology*. Addison-Wesley, Wokingham.

Griffel, F. (1998): *Componentware: Konzepte und Techniken eines Softwareparadigmas*. dpunkt, Heidelberg.

Griss, M. (2001): *CBSE Success Factors: Integrating Architecture, Process, and Organization*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 143-160.

Hall, A. (1990): *Seven Myths of Formal Methods*. In: IEEE Software (7) 5: 11-19.

Han, J. (1998): *A Comprehensive Interface Definition Framework for Software Components*. In: 1998 Asia-Pacific Software Engineering Conference, Taipei, Taiwan, IEEE Computer Society Press: 110-117.

Harmsen, F., Brinkkemper, S. und Oei, H. (1994): *Situational Method Engineering for Information System Project Approaches*. In: Verrijn-Stuart, A. A. und Olle, T. W. (Hrsg.): Methods and Associated Tools for the Information Systems Life Cycle. Elsevier, Amsterdam: 169-194.

Heinrich, L. J. (1993): *Wirtschaftsinformatik: Einführung und Grundlegung*. Oldenbourg, München.

Heinrich, L. J. (1994): *Systemplanung II: Planung und Realisierung von Informatik-Projekten*. 5. Aufl., Oldenbourg, München.

Heinrich, L. J. (1996): *Systemplanung I: Planung und Realisierung von Informatik-Projekten*. 7. Aufl., Oldenbourg, München.

Heinrich, L. J. (1999): *Informationsmanagement: Planung, Überwachung und Steuerung der Informationsinfrastruktur*. 6. Aufl., Oldenbourg, München.

Henderson-Sellers, B. und Edwards, J. M. (1994): *Book Two of Object-Oriented Knowledge*. Addison-Wesley, Sydney.

Herrmann, S., Mezini, M. und Ostermann, K. (2001): *Joint Efforts to Dispel an Approaching Modularity Crisis: Divide et Impera, Quo Vadis?* In: Bosch, J., Szyperski, C. und Weck, W. (Hrsg.): 6th Workshop on Component-Oriented Programming (WCOP'01), Budapest, Hungary.

Herzum, P. und Sims, O. (2000): *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley & Sons, New York, NY.

Hesse, W. (2002): *Ontologie(n)*. In: Informatik Spektrum (26) 6: 477-480.

Hodgson, R. (1999): *Systems Envisioning. Workshop Summary*. In: ACM (Hrsg.): Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'98), Vancouver, British Columbia, Canada, SIGPLAN Notices Nr. 33 (10), ACM.

Hofmeister, C., Nord, R. und Soni, D. (2000): *Applied Software Architecture*. Addison-Wesley, Upper Saddle River, NJ.

Houston, K. und Norris, D. (2001): *Software Components and the UML*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 243-262.

IEEE (1983): *IEEE Standard for Software Configuration Management Plans*. IEEE Standard Nr. 828-1983, Institute of Electrical and Electronics Engineers. New York.

IEEE (1991): *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Standard Nr. 610.12-1990, Institute of Electrical and Electronics Engineers. New York.

IEEE (2000): *Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE Standard Nr. 1471-2000, Institute of Electrical and Electronics Engineers. New York.

ISO/IEC (1994): *Basic Reference Model for Open Distributed Processing - Part 2: Foundations*. Nr. ISO/IEC 10746-2, International Organization for Standardization.

ISO/IEC (1999): *Software Engineering - Product Evaluation*. Nr. ISO/IEC Standard 14598, International Organization for Standardization.

ISO/IEC (2001): *Software Engineering - Product Quality - Part 1: Quality Model*. Nr. ISO/IEC Standard 9126-1, International Organization for Standardization.

ISO/IEC (2003): *Software Engineering - Product Quality - Part 2: External Metrics*. Nr. ISO/IEC Standard 9126-2, International Organization for Standardization.

Jackson, M. A. (1983): *Systems Development*. Prentice Hall, Englewood Cliffs, NJ.

Jackson, M. A. (1995): *Software Requirements und Specifications. A Lexicon of Practice, Principles, and Prejudices*. Addison-Wesley, Upper Saddle River, NJ.

Jacobson, I., Christerson, M., Jonsson, P. und Övergaard, G. (1992): *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley, Wokingham.

Jacobson, L., Griss, M. und Jonsson, P. (1997): *Software Reuse - Architecture, Process, and Organization for Business Success*. Addison-Wesley, Upper Saddle River, NJ.

Kalakota, R. und Robinson, M. (2001): *E-Business 2.0: Roadmap for Success*. Addison-Wesley, Upper Saddle River, NJ.

Knoblauch, S. und Egardt, V. (2003): *Service Level Management als Grundlage erfolgreicher IT-Outsourcing-Kundenbeziehungen*. In: Information Management & Consulting (18) SH: 28-31.

Kontio, J. (1996): *A Case Study in Applying a Systematic Method for COTS Selection*. In: IEEE (Hrsg.): 18th International Conference on Software Engineering, March 25-29, 1996, Berlin, Germany, IEEE Computer Society Press, Los Alamitos, CA: 201-209.

Kruchten, P. (1995): *The 4+1 View Model of Architecture*. In: IEEE Software (12) 6: 42-50.

Kuhn, T. S. (1976): *Die Struktur wissenschaftlicher Revolutionen*. 2. Aufl., Suhrkamp, Frankfurt.

Kurbel, K., Rautenstrauch, C., Opitz, B. und Scheuch, R. (1994): *From "Make or Buy" to "Make and Buy": Tailoring Information Systems Through Integration Engineering*. In: Journal of Database Management (5) 3: 18-30.

Larsson, M. und Crnkovic, I. (1999): *New Challenges for Configuration Management*. In: Estublier, J. (Hrsg.): System Configuration Management, 9th International Symposium, SCM-9, September 5-7, 1999, Toulouse, France, Lecture Notes in Computer Science Nr. 1675, Springer, Berlin, Heidelberg: 232-243.

Lassila, O. und Swick, R. R. (1999): *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation World Wide Web Consortium.

Lau, K.-K. und Ornaghi, M. (2001): *A Formal Approach to Software Component Specification*. In: Giannakopoulou, D., Leavens, G. T. und Sitaraman, M. (Hrsg.): Workshop on Specification and Verification of Component-Based Systems (SAVCBS'01), Tampa Bay, FL: 88-96.

Leavens, G. T. (1991): *Modular Specification and Verification of Object-Oriented Programs*. In: IEEE Software (8) 4: 72-80.

Lee, E. A. und Xiong, Y. (2001): *System-Level Types for Component-Based Design*. In: Henzinger, T. A. und Kirsch, C. M. (Hrsg.): Embedded Software, First International Workshop, EMSOFT 2001, October, 8-10, 2001, Tahoe City, CA, USA, Lecture Notes in Computer Science Nr. 2211, Springer, Berlin, Heidelberg: 237-253.

Lim, W. C. (1994): *Effects of Reuse on Quality, Productivity, and Economics*. In: IEEE Software (11) 5: 23-30.

Liskov, B. H. und Berzins, V. (1986): *An Appraisal of Programming Specifications*. In: Gehani, N. und McGettrick, A. T. (Hrsg.): Software Specification Techniques. Addison-Wesley, Wokingham: 3-24.

Liskov, B. H. und Wing, J. M. (1994): *A Behavioral Notion of Subtyping*. In: ACM Transactions on Programming Languages and Systems (16) 6: 1811-1841.

Liskov, B. H. und Zilles, S. N. (1974): *Programming with Abstract Data Types*. In: ACM SIGPLAN Notices (9) 4: 50-59.

Lorenzen, P. (1987): *Lehrbuch der konstruktiven Wissenschaftstheorie*. BI-Wissenschaftsverlag, Mannheim.

Löwy, J. (2001): *COM and .NET Component Services*. O'Reilly, Sebastopol, CA.

Loyall, J. P., Bakken, D. E., Schantz, R. E., Zinky, J. A., Karr, D. A., Vanegas, R. und Anderson, K. R. (1998): *QoS Aspect Languages and Their Runtime Integration*. In: O'Hallaron, D. R. (Hrsg.): Languages, Compilers, and Run-Time Systems for Scalable Computers, 4th International Workshop, LCR '98, Proceedings, Pittsburgh, PA, Lecture Notes in Computer Science Nr. 1511, Springer, Berlin, Heidelberg: 303-318.

Luckham, D. C., Kenney, J. J., Augustin, L. M., Vera, J., Bryan, D. und Mann, W. (1995): *Specification and Analysis of System Architecture Using Rapide*. In: IEEE Transactions on Software Engineering (21) 4: 336-355.

Lyons, J. (1972): *Structural Semantics: Analysis of Vocabulary of Plato*. Blackwell, Oxford.

Manna, Z. und Pnueli, A. (1991): *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, Berlin, Heidelberg.

Maurer, P. M. (2000): *Components: What If They Gave a Revolution and Nobody Came?* In: IEEE Computer (33) 6: 28-34.

McDermid, J. A. und Rook, P. (1991): *Software Development Process Models*. In: McDermid, J. A. (Hrsg.): Software Engineer's Reference Book. Butterworth-Heinemann, Oxford: 15/01-15/36.

McIlroy, M. D. (1968): *Mass Produced Software Components*. In: Naur, P. und Randell, B. (Hrsg.): Software Engineering: Report on a Conference by the NATO Science Committee, Brussels: 138-150.

McMenamin, M. und Palmer, J. F. (1984): *Essential Systems Analysis*. Prentice Hall, Englewood Cliffs, NJ.

Medvidovic, N. und Taylor, R. N. (1997): *A Framework for Classifying and Comparing Architecture Description Languages*. In: Jazayeri, M. und Schauer, H. (Hrsg.): Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering, September 22-25, 1997, Zurich, Switzerland, Lecture Notes in Computer Science Nr. 1301, Springer, Berlin, Heidelberg: 60-76.

Meyer, B. (1988): *Object-Oriented Software Construction*. Prentice Hall, Cambridge.

Meyer, B. (1992): *Applying "Design by Contract"*. In: IEEE Computer (25) 10: 40-51.

Meyer, B. (1997): *Object-Oriented Software Construction*. 2. Aufl., Prentice Hall, Upper Saddle River, NJ.

Mili, H., Mili, F. und Mili, A. (1995): *Reusing Software: Issues and Research Directions*. In: IEEE Transactions on Software Engineering (21) 6: 528-561.

Mili, R., Mili, A. und Mittermeir, R. T. (1998): *A Survey of Software Storage and Retrieval*. In: Annals of Software Engineering (5) 2: 349-414.

Mittermeir, R. T. (1990): *Requirements Engineering*. In: Kurbel, K. und Strunz, H. (Hrsg.): Handbuch Wirtschaftsinformatik. Poeschel, Stuttgart: 237-256.

Monroe, R. T., Kompanek, A., Melton, R. und Garlan, D. (1997): *Architectural Styles, Design, Patterns, and Objects*. In: IEEE Software (14) 1: 43-52.

Morris, J., Lee, G., Parker, K., Bundell, G. und Peng Lam, C. (2001): *Software Component Certification*. In: IEEE Computer (34) 9: 30-36.

Naur, P. und Randell, B. (Hrsg.) (1969): *Software Engineering: Report on a Conference by the NATO Science Committee*. Brussels.

Nierstrasz, O. (1989): *A Survey of Object-Oriented Concepts*. In: Kim, W. und Lochovsky, F. H. (Hrsg.): Object-Oriented Concepts, Databases, and Applications. Addison-Wesley, Reading, MA: 3-21.

Nierstrasz, O. (1993): *Regular Types for Active Objects*. In: Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Eighth Annual Conference, 26 September - 1 October, Proceedings, Washington, DC, USA, ACM SIGPLAN Notices 28 (10): 1-15.

Nierstrasz, O. (1995): *Regular Types for Active Objects*. In: Nierstrasz, O. und Tsichritzis, D. (Hrsg.): Object-Oriented Software Composition. Prentice Hall, Upper Saddle River, NJ: 99-121.

Nierstrasz, O. (1999): *Piccola - A Small Composition Language*. In: Moreira, A. M. und Demeyer, S. (Hrsg.): Object-Oriented Technology, ECOOP'99 Workshop Reader, ECOOP'99 Workshops, Panels, and Posters, June 14-18, 1999, Lisbon, Portugal, Lecture Notes in Computer Science Nr. 1743, Springer, Berlin, Heidelberg: 317.

Noack, J. und Schienmann, B. (1999): *Objektorientierte Vorgehensmodelle im Vergleich*. In: Informatik Spektrum (22) 3: 166-180.

Noy, N. F. (2004a): *Semantic Integration: A Survey of Ontology-Based Approaches*. In: ACM SIGMOD Record (33) 4: 65-70.

Noy, N. F. (2004b): *Tools for Mapping und Merging Ontologies*. In: Staab, S. und Studer, R. (Hrsg.): Handbook on Ontologies. Springer, Berlin, Heidelberg: 365-384.

Oberndorf, P., Wallnau, K. C. und Moorman Zaremski, A. (1998): *Product Lines: Reusing Architectural Assets within an Organization*. In: Bass, L., Clements, P. und Kazman, R. (Hrsg.): Software Architecture in Practice. Addison-Wesley, Reading, MA: 331-344.

Olle, T. W., Hagelstein, J., MacDonald, I. G. und Rolland, C. (1991): *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, Wokingham.

OMG (1997): *The Common Object Request Broker: Architecture and Specification. Revision 2.0*. Nr. 97-02-25, Object Management Group.

OMG (1999): *Unified Modeling Language (UML). Version 1.3*. Object Management Group.

OMG (2001): *General Ledger Specification. Version 1.0*. Object Management Group.

OMG (2002): *CORBA Components. Version 3.0*. Nr. 02-06-65, Object Management Group.

OMG (2003a): *UML 2.0 OCL Specification*. Adopted Specification Nr. ptc/03-10-14, Object Management Group.

OMG (2003b): *UML 2.0 Superstructure Specification*. Adopted Specification Nr. ptc/03-08-02, Object Management Group.

OMG (2004): *Common Object Request Broker Architecture: Core Specification. Version 3.0.3*. Nr. formal/04-03-12, Object Management Group.

Orfali, R., Harkey, D. und Edwards, J. (1996): *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, New York, NY.

Orfali, R., Harkey, D. und Edwards, J. (1999): *Client/Server Survival Guide*. 3. Aufl., John Wiley & Sons, New York, NY.

Orr, K. (1977): *Structured Systems Development*. Yourdon Press, New York, NY.

Ortner, E. (1997): *Methodenneutraler Fachentwurf*. Teubner, Stuttgart, Leipzig.

Ortner, E. (1998): *Ein Multipfad-Vorgehensmodell für die Entwicklung von Informationssystemen - dargestellt am Beispiel von Workflow-Management Anwendungen*. In: Wirtschaftsinformatik (40) 4: 329-337.

Overhage, S. (2002): *Komponentenkataloge auf Basis eines einheitlichen Spezifikationsrahmens - ein Implementierungsbericht*. In: Turowski, K. (Hrsg.): 3. Workshop Modellierung und Spezifikation von Fachkomponenten, Nürnberg, Germany, Augsburg University Press: 1-15.

Overhage, S. (2003): *Towards a Standardized Specification Framework for Component Development, Discovery, and Configuration*. In: Bosch, J., Szyperski, C. und Weck, W. (Hrsg.): Eighth International Workshop on Component-Oriented Programming (WCOP'03), Darmstadt, Germany.

Overhage, S. (2004): *Zur Spezifikation von Komponenten der Informationssystementwicklung mit Softwareverträgen*. In: Rebstock, M. (Hrsg.): Modellierung betrieblicher Informationssyteme - MobIS 2004, Proceedings zur Tagung, 10. März 2004, Essen, Germany, Lecture Notes in Informatics Nr. P-41, Köllen: 3-20.

Overhage, S. und Thomas, P. (2004): *A Business Perspective on Component Trading: Criteria, Immaturities, and Critical Success Factors*. In: IEEE (Hrsg.): Euromicro 2004, 31 August - 4 September 2004, Rennes, France, IEEE Computer Society Press, Los Alamitos, CA: 108-117.

Overhage, S. und Thomas, P. (2005): *WS-Specification: Ein Spezifikationsrahmen zur Beschreibung von Web-Services auf Basis des UDDI-Standards*. In: Ferstl, O. K., Sinz, E. J., Eckert, S. und Isselhorst, T. (Hrsg.): Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety, Bamberg, Germany, Physica, Heidelberg: 1539-1558.

Pahl, G. und Beitz, W. (2003): *Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung*. 5. Aufl., Springer, Berlin, Heidelberg.

Pantry, S. und Griffiths, P. (1997): *The Complete Guide to Preparing and Implementing Service Level Agreements*. Library Association Publishing, London.

Parnas, D. L. (1972): *On the Criteria to be Used in Decomposing Systems into Modules*. In: Communications of the ACM (15) 12: 1053-1058.

Pine II, B. J. (1993): *Mass Customization: The New Frontier in Business Competition*. Harvard Business School Press, Boston, MA.

Pintado, X. (1995): *Gluons and the Cooperation between Software Components*. In: Nierstrasz, O. und Tsichritzis, D. (Hrsg.): Obejct-Oriented Software Composition. Prentice Hall, Hertfordshire: 321-349.

Plasil, F. und Visnovsky, S. (2002): *Behavior Protocols for Software Components*. In: IEEE Transactions on Software Engineering (28) 11: 1056-1076.

Pohl, K. (1994): *The Three Dimensions of Requirements Engineering: A Framework and its Applications*. In: Information Systems (19) 3: 243-258.

Pree, W. (1994): *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, Wokingham.

Pree, W. (1997): *Komponentenbasierte Softwareentwicklung mit Frameworks*. dpunkt, Heidelberg.

Pressman, R. S. (1997): *Software Engineering: A Practitioner's Approach*. McGraw-Hill, New York, NY.

Prieto-Díaz, R. (1990): *Domain Analysis: An Introduction*. In: Software Engineering Notes (15) 2: 47-54.

Prieto-Díaz, R. (1991): *Implementing Faceted Classification for Software Reuse*. In: Communications of the ACM (34) 5: 89-97.

Prieto-Díaz, R. und Freeman, P. (1987): *Classifying Software for Reusability*. In: IEEE Software (4) 1: 6-16.

Ramamoorthy, C. V., Prakash, A., Tsai, W.-T. und Usuda, Y. (1984): *Software Engineering: Problems and Perspectives*. In: IEEE Computer (17) 10: 191-209.

Reussner, R., Schmidt, H. W. und Poernomo, I. (2003): *Reliability Prediction for Component-Based Software Architectures*. In: Journal of Systems and Software (66) 3: 241-252.

Reussner, R. H. (2001): *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. Logos, Berlin.

Reussner, R. H. und Schmidt, H. W. (2002): *Using Parameterised Contracts to Predict Properties of Component-Based Software Architectures*. In: Crnkovic, I., Larsson, S. und Stafford, J. (Hrsg.): Workshop on Component-Based Software Engineering (in association with 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems), Lund, Sweden.

Ropohl, G. (1978): *Einführung in die allgemeine Systemtheorie*. In: Ropohl, G. und Lenk, H. (Hrsg.): Systemtheorie als Wissenschaftsprogramm. Athenäum, Bodenheim: 8-49.

Ropohl, G. (1999): *Allgemeine Technologie*. 2. Aufl., Hanser, München, Wien.

Royce, W. W. (1970): *Managing the Development of Large Software Systems: Concepts and Techniques*. In: IEEE WESCON, Washington, DC,USA, IEEE Computer Society Press, Los Alamitos, CA: 1-9.

Ruhe, G. (2002): *Intelligent Support for Selection of COTS Products*. In: Chaudhri, A. B., Jeckle, M., Rahm, E. und Unland, R. (Hrsg.): Web, Web-Services, and Database Systems, Proceedings of the NODe 2002 Web and Database-Related Workshops, October 7-10, 2002, Erfurt, Germany, Lecture Notes in Computer Science Nr. 2593, Springer, Berlin, Heidelberg: 34-45.

Sametinger, J. (1997): *Software Engineering with Reusable Components*. Springer, Berlin, Heidelberg.

Sattler, K. U. (1997): *A Framework for Component-Oriented Tool Integration*. In: 4th International Conference on Object-Oriented Information Systemes (OOIS'97), Brisbane, Australia: 455-465.

Schach, S. R. (1990): *Software Engineering*. Aksen, Homewood.

Scheer, A.-W. (1998): *ARIS: Vom Geschäftsprozess zum Anwendungssystem*. 3. Aufl., Springer, Berlin, Heidelberg.

Schienmann, B. (1997): *Objektorientierter Fachentwurf: Ein terminologiebasierter Ansatz für die Konstruktion von Anwendungssystemen*. Teubner, Stuttgart, Leipzig.

Schmidt, H. W. und Reussner, R. H. (2002): *Generating Adapters for Concurrent Protocol Synchronisation*. In: Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems.

Schöning, U. (1995): *Theoretische Informatik - kurzgefasst*. 2. Aufl., Spektrum, Heidelberg, Berlin.

Schwegmann, A. (1999): *Objektorientierte Referenzmodellierung*. Deutscher Universitäts-Verlag, Wiesbaden.

Seco, J. C. und Caires, L. (2000): *A Basic Model of Typed Components*. In: Bertino, E. (Hrsg.): ECOOP 2000 - Object-Oriented Programming, 14th European Conference, June 12-16, 2000, Sophia Antipolis and Cannes, France, Lecture Notes in Computer Science Nr. 1850, Springer, Berlin, Heidelberg: 108-128.

Shaw, M. und Garlan, D. (1996): *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ.

Siedersleben, J. (2004): *Moderne Softwarearchitektur. Umsichtig bauen, robust planen mit Quasar*. dpunkt, Heidelberg, Germany.

Simons, P. (1987): *Parts. A Study in Ontology*. Clarendon Press, Oxford.

Sinz, E. J. (1999): *Anwendungssysteme aus fachlichen Komponenten*. In: Wirtschaftsinformatik (41) 1: 3.

Snyder, A. (1986): *Encapsulation and Inheritance in Object-Oriented Programming Languages*. In: Meyrowitz, N. K. (Hrsg.): Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'86), Proceedings, Portland, Oregon, ACM SIGPLAN Notices 21 (11): 38-45.

Sommerville, I. (1992): *Software Engineering*. 4. Aufl., Addison-Wesley, Wokingham.

Speed, J., Councill, W. T. und Heineman, G. T. (2001): *Component-Based Software Engineering as a Unique Engineering Discipline*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 673-691.

Spillner, A. und Linz, T. (2005): *Basiswissen Softwaretest*. 3. Aufl., dpunkt, Heidelberg.

Stahlknecht, P. und Hasenkamp, U. (1997): *Einführung in die Wirtschaftsinformatik*. 8. Aufl., Springer, Berlin, Heidelberg.

Stets, R. J., Hunt, G. C. und Scott, M. L. (1999): *Component-Based APIs for Versioning and Distributed Applications*. In: IEEE Computer (32) 7: 54-61.

Stevens, P. und Pooley, R. (2000): *Using UML Software Engineering with Objects and Components*. Addison-Wesley, Menlo Park, CA.

Stojanovic, N. (2005): *On the Query Refinement in Searching a Bibliographic Database*. In: Ferstl, O. K., Sinz, E. J., Eckert, S. und Isselhorst, T. (Hrsg.): Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety, Bamberg, Germany, Physica, Heidelberg: 1329-1346.

Szyperski, C. (1998): *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Harlow.

Szyperski, C., Gruntz, D. und Murer, S. (2002): *Component Software: Beyond Object-Oriented Programming*. 2. Aufl., Addison-Wesley, Harlow.

Thayer, R. H. und Dorfman, M. (1990): *System and Software Requirements Engineering*. IEEE Computer Society Press, Los Alamitos, CA.

Turowski, K. (2001): *Spezifikation und Standardisierung von Fachkomponenten*. In: Wirtschaftsinformatik (43) 3: 269-281.

Turowski, K. (2003): *Fachkomponenten: Komponentenbasierte betriebliche Anwendungssysteme*. Shaker, Aachen.

van Solingen, R. und Berghout, E. (1999): *The Goal/Question/Metric Method*. McGraw-Hill, Columbus, OH.

Vitharana, P., Zahedi, F. und Jain, H. (2003): *Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and an Empirical Analysis*. In: IEEE Transactions on Software Engineering (29) 7: 649-664.

Wallnau, K. C. (2003): *A Technology for Predictable Assembly from Certifiable Components*. Technical Report Nr. CMU/SEI-2003-TR-009, Software Engineering Institute, Carnegie Mellon University.

Wallnau, K. C., Hissam, S. A. und Seacord, R. C. (2002): *Building Systems from Commercial Components*. Addison-Wesley, Upper Saddle River, NJ.

Wallnau, K. C., Stafford, J., Hissam, S. und Klein, M. (2001): *On the Relationship of Software Architecture to Software Component Technology*. In: Bosch, J., Szyperski, C. und Weck, W. (Hrsg.): 6th Workshop on Component-Oriented Programming (WCOP'01), Budapest, Hungary.

Wang, N., Schmidt, D. C. und O'Ryan, C. (2001): *Overview of the CORBA Component Model*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 557-571.

Weske, M. (1999): *Business-Objekte: Konzepte, Architekturen, Standards*. In: Wirtschaftsinformatik (41) 4: 4-11.

Wessel, H. (1976): *Logik und Philosophie*. Deutscher Verlag der Wissenschaften, Berlin.

Weyuker, E. J. (2001): *The Trouble with Testing Components*. In: Councill, W. T. und Heineman, G. T. (Hrsg.): Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, Upper Saddle River, NJ: 499-512.

Widhalm, R. und Mück, T. (2002): *Topic Maps*. Springer, Berlin, Heidelberg.

Wiener, N. (1965): *Cybernetics - 2nd Edition: Or the Control and Communication in the Animal and the Machine*. MIT Press, Cambridge.

Wilkie, G. (1993): *Object-Oriented Software Engineering*. Addison-Wesley, Wokingham.

Williams, J. D. (1996): *Managing Iteration in OO Projects*. In: IEEE Computer (29) 9: 39-43.

Wirth, N. (1971): *Program Development by Stepwise Refinement*. In: Communications of the ACM (14) 4: 221-227.

Yellin, D. M. und Strom, R. E. (1994): *Interfaces, Protocols, and the Semiautomatic Construction of Software Adaptors*. In: 9th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'94), Portland, OR, USA, ACM SIGPLAN Notices Nr. 29 (10): 176-190.

Yellin, D. M. und Strom, R. E. (1997): *Protocol Specifications and Component Adaptors*. In: ACM Transactions on Programming Languages and Systems (19) 2: 292-333.

Yourdon, E. (1989): *Modern Structured Analysis*. Yourdon Press, Englewood Cliffs, NJ.

Yourdon, E. und Constantine, L. L. (1979): *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, Englewood Cliffs, NJ.

Zaremski, A. und Wing, J. M. (1995): *Signature Matching: A Tool for Using Software Libraries*. In: ACM Transactions on Software Engineering and Methodology (4) 2: 146-170.

Zaremski, A. und Wing, J. M. (1997): *Specification Matching of Software Components*. In: ACM Transactions on Software Engineering and Methodology (6) 4: 333-369.

Zinky, J. A., Bakken, D. E. und Schantz, R. E. (1997): *Architectural Support for Quality of Service for CORBA Objects*. In: Theory and Practice of Object Systems (3) 1: 55-73.