Dienstgüte-Management in verteilten Objektsystemen

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

vorgelegt beim Fachbereich Biologie und Informatik

der Johann Wolfgang Goethe – Universität

in Frankfurt am Main

von

Christian Robert Becker

aus Hanau/Main

Frankfurt am Main 2001
(D F 1)

vom Fachbereich Biologie und Informatik der

Johann Wolfgang Goethe – Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Karl-Dieter Entian
Gutachter:	Prof. Dr. Kurt Geihs Johann Wolfgang Goethe – Universität, Frankfurt
	Prof. Dr. Andreas Oberweis Johann Wolfgang Goethe – Universität, Frankfurt
Datum der Disputation:	9.2.2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungs- und Symbolverzeichnis

$\leq$	Subtyp-Relation
i, j, k, l, m, n	stellen im Verlauf der Arbeit frei wählbare Zahlen aus dem Bereich der natürlichen Zahlen dar, soweit sie nicht weiter eingeschränkt wurden
$C \rightarrow S.op$	Aufruf der Methode *op* des Dienstes S durch den Klienten C
ϕ	Kleine griechische Buchstaben bezeichnen Dienstgüte-Charakteristiken
M_ϕ	Dienstgüte-Mechanismus, der Dienstgüte-Charakteristik ϕ implementiert
$S(M_\phi)$	Schnittstelle des Dienstgüte-Mechanismus M_ϕ
$S(D)$	Schnittstelle des Dienstes D
$< D, Q >$	Dienstgüte-Bereitschaft eines Dienstes D mit $Q = \{\phi_1, \ldots \phi_n\}$ mögliche Dienstgüte-Charakteristiken
$[K, D]$	Bindung zwischen Klient K und Dienst D
$[K, D, Q]$	Bindung zwischen Klient K und Dienst D mit $Q = \{\phi_1, \ldots \phi_n\}$ mögliche Dienstgüte-Charakteristiken
$[K, D, \overline{Q}]$	Dienstgüte-Vereinbarung zwischen Klient K und Dienst D mit $\overline{Q} = \{\overline{\phi}_j, \ldots \overline{\phi}_k\}$ $1 \leq j \leq k \leq n$ Dienstgüte-Niveaus
ANSA	Advanced Network Systems Architecture
AOP	Aspektorientierte Programmierung
API	Application Programming Interface
ART	Adaptive Runtime Technology
ATM	Asynchronous Transfer Mode
CORBA	Common Object Request Broker Architecture

DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
GIOP	General Inter ORB Protocol
IDL	Interface Definition Language; Schnittstellenbeschreibungssprache
IIOP	Internet Inter ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Organization for Standardization
ITU	International Telecommunication Union
Java/RMI	Java Remote Method Invocation
MAQS	Management Architecture for Quality of Service
MICO	MICO is CORBA
OCI	Open Communications Interface
OGS	Object Group Service
OMA	Object Management Architecture
OMG	Object Management Group
OOAD	Object-Oriented Analysis and Design; Objektorientierte Analyse und Entwurf
ORB	Object Request Broker
QML	QoS Modeling Language
QDL	Quality of Service Description Language
QDII	Quality of Service Dynamic Invocation Interface
QDSI	Quality of Service Dynamic Skeleton Interface
QIDL	Quality of Service IDL
QoS	Quality of Service; Dienstgüte
QuO	Quality Objects
RAID	Redundant Array of Inexpensive Disks
RM-ODP	Reference Model of Open Distributed Processing
R-RIO	Reflective-Reconfigurable Interconnectable Objects
RSVP	Resource Reservation Protocol
TAO	The Ace ORB
TCP	Transmission Control Protocol
TINA	Telecommunications Information Networking Architecture
UDP	User Datagram Protocol

UML	Unified Modeling Language
WWW	World Wide Web
XML	Extensible Markup Language

Kapitel 1

Einleitung

Die Erbringung elektronischer Dienstleistungen hängt zunehmend von Faktoren ab, die über die rein funktionale Beschreibung einer Dienstleistung hinausgehen. Qualitätsanforderungen wie Bildgröße oder Farbtiefe im Multimediabereich müssen von Anwendungen bzw. zugrundeliegenden Mechanismen durchgesetzt werden. Mit der zunehmenden Verbreitung verteilter Systeme sind die dort auftretenden Charakteristika der Diensterbringung neben anwendungsbezogenen Anforderungen nach einer bestimmten Dienstqualität zu integrieren. So ist der Betrieb verteilter Systeme Störungen unterworfen, die durch die Verteilung induziert werden. Nachrichtenverlust, Verzögerungen der Bearbeitung oder Ausfälle von Rechnern oder Kommunikationsstrecken sind allgegenwärtig. Für die Nutzung elektronischer Dienstleistungen müssen Vorkehrungen getroffen werden, um einen robusten Betrieb eines solchen Systems zu gewährleisten. Unter Dienstgüte-Management werden Vorkehrungen zum Betrieb verteilter Systeme verstanden, die anwendungsspezifische Qualitätsanforderungen bei der Diensterbringung sicherstellen.

Die Popularität von Verteilungsinfrastrukturen – inbesondere objektorientierte – zur Entwicklung verteilter Systeme macht eine Integration von Dienstgüte-Management wünschenswert. Als Verteilungsinfrastrukturen, auch Middleware genannt, werden Softwareschichten zwischen Anwendungen und der Plattform – Betriebssystem und Netzwerk – verstanden, die das Erstellen und den Betrieb verteilter Anwendungen vereinfachen. Die Vereinheitlichung der Schnittstellen zu solcher Middleware erlaubt das Erstellen portabler Anwendungen. Als Leitbild einer solchen Integration soll hier die Entkopplung von Netzwerk- und Anwendungsprotokollen durch

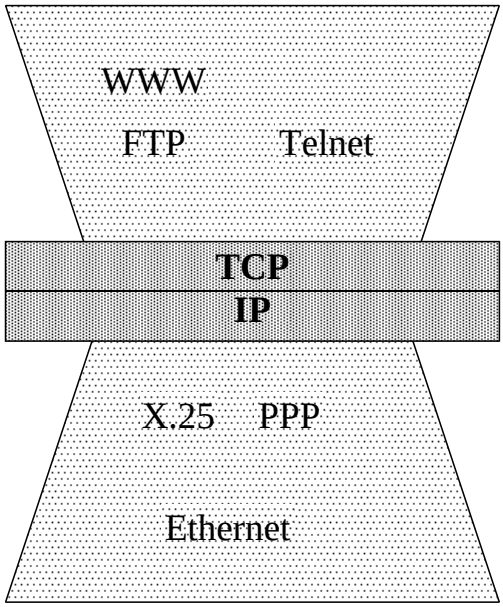


Abbildung 1.1: Entkopplung der Protokoll-Schichten durch TCP/IP

das Transmission Control Protocol über dem Internet Protocol (TCP/IP) dienen. Abbildung 1.1 illustriert, wie verschiedene Anwendungsprotokolle durch TCP/IP von den unterschiedlichen Netzwerkprotokollen entkoppelt werden. Dies erlaubt die unabhängige Entwicklung auf beiden Schichten.

Aktuelle Bestrebungen im Internet-Bereich adressieren die Erweiterung von TCP/IP um Dienstgüte-Befähigung. Dabei ist allerdings die Anwendungsanforderung nach einer bestimmten Dienstgüte auf Abstraktionen des Netzwerkes, typischerweise Bandbreite und Verzögerung, abzubilden. Abgesehen davon, daß die dabei betrachteten Dienstgüte-Anforderungen hauptsächlich den Multimedia- und Echtzeit-Bereich abdecken, ist die Abstraktion von Protokollen und Dienstgüte-Vorkehrungen auf dieser Ebene für die Entwicklung von Anwendungen wenig erstrebenswert. Zum einen sollten Anwendungen auf höheren Abstraktionen als TCP/IP aufbauen können. Dies bedeutet die Integration von Dienstgüte-Vorkehrungen in eine Verteilungsinfrastruktur. Desweiteren sind die alleine auf der Netzwerk-Schicht realisierten Dienstgüte-Vorkehrungen für viele Dienstgüte-Anforderungen auf der Anwendungsebene nicht ausreichend. Beispielsweise wird für die Zusicherung einer bestimmten Verfügbarkeit durch Redundanz mehr als nur ein angepaßtes Protokoll notwendig. Die Schicht zwischen Anwen-

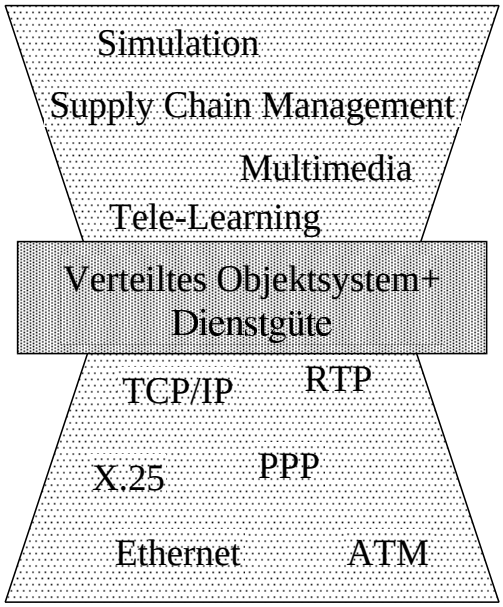


Abbildung 1.2: Entkopplung von Anwendungen und Plattform durch eine Verteilungsinfrastruktur

dung und Plattform wird durch Verteilungsinfrastrukturen gebildet. Neben den Abstraktionen für die einfachere Programmierung durch Anwendungsentwickler erlaubt diese zusätzliche Schicht die Integration von Dienstgüte-Vorkehrungen.

In Abbildung 1.2 ist dargestellt, wie sich eine Verteilungsinfrastruktur zwischen den Netzwerkprotokollen und den Anwendungen einbettet. Dabei ist die dargestellte Sicht auf das System eine andere als in Abbildung 1.1 [12]. Die Verteilungsinfrastruktur, die als verteiltes Objektsystem den Anwendungen eine Abstraktion oberhalb der Protokolle anbietet, entkoppelt diese auf einer höheren Abstraktion von den zugrundeliegenen Mechanismen der Plattform. Die Dienstgüte-Vorkehrungen der Verteilungsinfrastruktur sollten für eine große Menge von Anwendungen ausreichend sein. Die in der Abbildung skizzierten Anwendungen zeigen schon, daß eine Reihe unterschiedlicher Dienstgüte-Vorkehrungen zu berücksichtigen ist. Die Dienstgüte-Integration in eine solche Verteilungsinfrastruktur sollte generisch erfolgen, d.h. neue Dienstgüte-Vorkehrungen können durch geeignete Schnittstellen der Infrastruktur eingefügt werden. Dabei sind gegenüber der Anwendung geeignete Programmierschnittstellen für die Nutzung von Dienstgüte-Vorkehrung zu etablieren. Die Dienstgüte-bezogenen Vor-

kehrungen tieferer Schichten sollten – wie es die Verteilungsinfrastruktur bereits für die Kommunikation macht – vor der Anwendung verborgen werden.

In dieser Arbeit wird ein Rahmenwerk vorgestellt, das die Integration von Dienstgüte-Vorkehrungen in verteilten Objektsystemen ermöglicht. Als Basis des Rahmenwerks wird ein verteiltes Objektmodell, wie es gängigen objektorientierten Verteilungsinfrastrukturen zugrundeliegt, vorausgesetzt. Die Aufgaben und Anforderungen an Dienstgüte-Management in verteilten Objektsystemen werden analysiert und insbesondere wird auf Besonderheiten bei der Dienstgüte-Integration eingegangen. Abhängig der Phasen eines verteilten Systems – hier unterteilt in den Entwurf und die Laufzeit – werden die Aufgaben und Anforderungen in Beziehung zu den von der Verteilungsinfrastruktur angebotenen Mechanismen und Diensten motiviert und klassifiziert. Die insbesondere zur Laufzeit notwendigen Dienste werden im Rahmen dieser Arbeit nicht vertieft. Ziel ist das Bereitstellen einer Abstraktion für die Dienstgüte-Vorkehrungen, analog zu der von der Verteilungsinfrastruktur realisierten Abstraktion der zugrundeliegenden Plattform. Besonderes Augenmerk liegt dabei auf der Entkopplung von Anwendungsobjekten und Dienstgüte-Vorkehrungen um die Wiederverwendbarkeit beider zu gewährleisten.

Dienstgüte-Integration in verteilten Objektsystemen stellt einen Aspekt im Sinne der aspektorientierten Programmierung dar. Der Schwerpunkt der vorliegenden Arbeit liegt in dem Entwurf einer Dienstgüte-Spezifikation und deren Integration in eine Verteilungsinfrastruktur. Diese Integration folgt dem Leitbild der aspektorientierten Programmierung und ermöglicht durch eine geeignete Übersetzung der Spezifikation eine weitgehende Entkopplung der Dienstgüte-Vorkehrungen von den Anwendungsobjekten. Der Ende-zu-Ende-Eigenschaft der Dienstgüte-Erbringung wird durch die Integration der Dienstgüte-Mechanismen-Hierarchie in den Kern der Verteilungsinfrastruktur Rechnung getragen. Die Einbettung der Dienstgüte-Vorkehrungen zugrundeliegender Schichten, wie Netzwerk oder Betriebssystem, in den Kern einer Verteilungsinfrastruktur und deren Anbindung an die Dienstgüte-Vorkehrungen der Anwendungsschicht komplettieren die Integration in das verteilte Objektmodell.

Die im Rahmen dieser Arbeit betrachteten Dienstgüte-Anforderungen adressieren systemnahe, Verteilungs-induzierte Aspekte, wie Verfügbarkeit, Antwortzeit oder Sicherheit. Diese weisen – trotz der in Kapitel 3 gezeigten

Vermaschung mit der Anwendung – eine hohe Orthogonalität in Bezug auf die Trennung zu Anwendungsobjekten auf. Die Integration von Dienstgüte-Anforderungen mit höherem Anwendungsbezug ist ein bislang wenig untersuchtes Gebiet. Solche Dienstgüte-Anforderungen können mit der Qualität einer Audio-Übertragung und der dabei verwendeten Kompression – bspw. dem Verlustfaktor – oder anderen Kriterien, wie Stereo- oder Mono-Daten verknüpft sein. Während die Kompression weitgehend orthogonal zu den Verantwortlichkeiten der Anwendung ist, setzt die Anforderung nach Stereo-Daten auch die Möglichkeit diese zu verarbeiten in den jeweiligen Endgeräten voraus. Die dafür notwendige Integration von Dienstgüte-Mechanismen tieferer Schichten in eine Verteilungsinfrastruktur wird von dem in dieser Arbeit vorgestellten Ansatz geleistet.

Die Integration von Dienstgüte-Management in Verteilungsinfrastrukturen berührt dabei Kerngebiete anderer Disziplinen. So ist die Bewertung von Dienstleistungen und der geleisteten Qualität ebenso wie die Entwicklung von Abrechnungsmodellen eine Kernkompetenz der Betriebswirtschaftslehre. Die Gestaltung von Verträgen zwischen elektronischen Partnern und deren Durchsetzung bzw. Abwicklung in Hinblick auf Qualität und Preis wirft juristische Fragen auf, die Verbindlichkeit und Gültigkeit solcher Verträge betreffen. Schließlich hängen die Dienstgüte-Anforderungen einer Anwendung von deren Zweck ab. Das bedeutet, schon bei der Modellierung sind Dienstgüte-Anforderungen zu beachten. Um diese Dienstgüte-Anforderungen früh berücksichtigen zu können, ist insbesondere Wissen um die anwendungsspezifischen Anforderungen und die Einschränkungen der zugrundeliegenden Plattform notwendig. Hier sind Anwender – mit ihren Vorgaben –, Modellierer und Implementierer gleichermaßen gefordert.

Die vorliegende Arbeit ist im Rahmen des Sonderforschungsbereichs SFB-403, "Vernetzung als Wettbewerbsfaktor am Beispiel der Region Rhein-Main", entstanden. Die große Breite der beteiligten Disziplinen (BWL, Jura, Politologie, Soziologie, Arbeitswissenschaften, Wirtschaftsgeographie, Wirtschaftsinformatik und Informatik) hat zu vielen angrenzenden Fragestellungen des Dienstgüte-Managements Diskussionen mit kompetenten Ansprechpartnern erlaubt. Bei der Gestaltung der Klientenpräferenzen konnte so auf Grundlagen des Utilitarismus zurückgegriffen werden und Modelle der Ökonomie auf ihre Tauglichkeit für die Repräsentation der Präferenzen im Umfeld des Dienstgüte-Managements untersucht werden. Die Bepreisung elektronischer Dienstleistungen unter Einbezug von Qualitätsaspekten

wie auch der rechtlichen Implikationen bei der Ausgestaltung von Verträgen zwischen elektronischen Kommunikationspartnern lassen den Einbezug von anderen Disziplinen, wie der Betriebswirtschaftslehre oder der Rechtswissenschaften, bei weiteren Forschungsarbeiten als gewinnbringend erscheinen. In [5] ist als Ergebnis der Zusammenarbeit innerhalb des SFBs ein Forschungsvorhaben beschrieben, das die Abbildung Anreiz-integrierender Verträge aus der Logistik unter Einbezug der technischen Möglichkeiten in Verteilungsinfrastrukturen untersucht.

Die Arbeit gliedert sich wie folgt:

Kapitel 2: Verteilte Objektsysteme Die Grundlagen verteilter Systeme – soweit sie die hier vorliegende Arbeit betreffen – werden eingeführt. Anschließend wird das verteilte Objektmodell und seine Komponenten vorgestellt. Die Grenzen der Verteilungstransparenz im Kontext des Dienstgüte-Managements werden diskutiert.

Kapitel 3: Dienstgüte In diesem Kapitel wird der Begriff "Dienstgüte" eingeführt und durch verschiedene Definitionen abgegrenzt. Eine vertiefte Betrachtung erfährt die Dienstgüte-Integration in verteilten Objektsystemen. Dienstgüte wird als ein Aspekt im Sinne der aspektorientierten Programmierung klassifiziert. Dies bedeutet, daß Anwendungs-Objekte und Dienstgüte-Vorkehrungen in einem hohem Maße abhängig voneinander sind und insbesondere im allgemeinen nicht eingekapselt werden können. Zwei Lösungsansätze zu den aufgezeigten Problemen werden vorgestellt. Das Kapitel endet mit einer Darstellung der Aufgaben des Dienstgüte-Managements und einem Überblick des Stands der Forschung.

Kapitel 4: Ein Dienstgüte-Rahmenwerk Zu den in Kapitel 3 motivierten Anforderungen des Dienstgüte-Managements wird ein Rahmenwerk für das Dienstgüte-Management in verteilten Objektsystemen vorgestellt. Den Schwerpunkt der Arbeit bildet die Spezifikation von Dienstgüte-Charakteristiken und die Integration der zugehörigen Dienstgüte-Mechanismen in die Verteilungsinfrastruktur. Die weiteren Bestandteile des Rahmenwerks, wie Dienste und Spezifikationen von Klientenwünschen für die Verhandlung werden in diesem Kapitel kurz vorgestellt.

Kapitel 5: Spezifikation von Dienstgüte Dieses Kapitel erläutert eine Integration von Dienstgüte-Spezifikationen in die Schnittstellenbeschreibungssprache einer Verteilungsinfrastruktur und deren Umsetzung in die Zielsprache. Eingangs werden Alternativen zu dieser Integration diskutiert und der eingeschlagene Weg motiviert. Die Umsetzung erlaubt eine Verwebung zwischen Dienstgüte-Aspekten und Anwendungs-Aspekten. Wegen der in Kapitel 3 aufgezeigten Anomalien bei der Integration von Dienstgüte-Vorkehrungen in verteilte Objektsysteme wird hier ein aspektorientierter Ansatz gewählt, um eine Entkopplung der Anwendung von den Dienstgüte-Vorkehrungen zu erreichen.

Kapitel 6: Dienstgüte-Mechanismen-Integration Die Erbringung einer bestimmten Dienstgüte zwischen Klient und Dienst hängt von allen beteiligten Schichten entlang des Kommunikationspfades ab. An dieser Stelle wird auf die Integration von Dienstgüte-Mechanismen in eine Verteilungsinfrastruktur eingegangen. Grundsätzliche Modelle für die Architektur von Verteilungsplattformen werden skizziert und bei dem vorgeschlagenen Modell berücksichtigt. Die Verbindung zwischen den generierten Einheiten der Dienstgüte-Spezifikation und den Dienstgüte-Mechanismen der zugrundeliegenden Plattform wird so ermöglicht.

Kapitel 7: Zusammenfassung und Ausblick Nach einer Zusammenfassung der in dieser Arbeit vorgestellten Ergebnisse werden offene Fragen diskutiert und mögliche Richtungen weiterer Forschung aufgezeigt.

Kapitel 2

Verteilte Objektsysteme

In diesem Kapitel werden grundlegende Eigenschaften verteilter Systeme diskutiert. Der Schwerpunkt liegt dabei auf verteilten Objektsystemen.

2.1 Charakterisierung verteilter Systeme

Unter einem *verteilten System* wird ein Zusammenschluß von Rechnersystemen verstanden, der eine gemeinsame Aufgabe bearbeitet. Dazu müssen Softwarekomponenten auf den einzelnen Rechnern installiert sein, die miteinander kommunizieren[1]. Die Topologie eines verteilten Systems kann durch einen Graphen dargestellt werden. Dabei repräsentieren die Knoten des Graphen die kommunizierenden Komponenten, und die Kanten stellen die Kommunikationsverbindungen dar. Abhängig von der Sicht auf das System können die Kanten und Knoten unterschiedlicher Natur sein. Knoten können sowohl Softwarekomponenten als auch Hardwarekomponenten sein. Im Rahmen dieses Kapitels sollen Knoten als Softwarekomponenten verstanden werden, die als Bestandteil eines verteilten Systems fungieren. Dabei greifen diese Komponenten auf die Funktionalität ihrer Ausführungsumgebung zurück. Diese Ausführungsumgebung stellt somit die *Plattform* für solche Komponenten dar. Liegt den Softwarekomponenten das Objektmodell für Entwurf und Realisierung zu Grunde, spricht man von *verteilten Objektsystemen*. Insbesondere wird die Grundlage von verteilten Systemen und deren Realisierung durch sogenannte Middleware vorgestellt.

[1] Leslie Lamport wird die – sicher nicht ganz ernst gemeinte – Definition eines verteilten Systems zugeschrieben: *One on which I cannot get any work done because some machine I have never heard of has crashed [93].*

Eigenschaften verteilter Systeme sind:

- **Kommunikation:** Die Komponenten eines verteilten Systems müssen kommunizieren, damit die Gesamtfunktionalität erbracht wird. Dies bedeutet, daß Adreßraum- und Rechnergrenzen bei der Kommunikation überwunden werden müssen.

- **Ressourcen-Nutzung:** Die in einem verteilten System verfügbaren Ressourcen stellen nominell die Summe der existierenden Ressourcen der beteiligten Komponenten dar. Durch die Integration von Standardkomponenten, wie Personal Computers oder Workstations, erreichen verteilte Systeme bezogen auf die nominelle Rechenleistung ein gutes Preis/Leistungs-Verhältnis. Ein weiterer positiver Effekt ist die gemeinsame Nutzung von Ressourcen. So können Ressourcen, die für einzelne Knoten übermäßig teuer im Betrieb wären, durch das Netzwerk mit anderen Knoten geteilt werden.

- **Parallelität:** In verteilten Systemen werden typischerweise autarke Rechnersysteme, wie Arbeitsplatzrechner verbunden. Somit ist Parallelität in einem solchen System inhärent gegeben.

- **Koordination:** Die Existenz von Parallelität birgt – wie in nichtverteilten Systemen auch – potentielle Probleme, beispielsweise die Synchronisation von Datenzugriffen. Durch entsprechende Vorkehrungen, z.B. kritische Abschnitte, müssen solche Synchronisationsprobleme gelöst werden. Das Fehlen einer globalen Sicht auf solche Systeme führt dazu, daß Terminierung von Programmen oder zeitliche Abstimmungen nicht ohne besondere Vorkehrungen etabliert werden können.

- **Dynamik:** Während der Laufzeit eines verteilten Systems können nicht vorhersagbare Lasten auf den Knoten und Kanten auftreten. Im Gegensatz zu a priori bekannten Konfigurationen eines Systems, wie in geschlossenen verteilten Systemen, können potentiell beliebige Knoten in ein offenes verteiltes System eintreten. Diese können sich in Hardware, Systemsoftware oder aber auch den Implementierungssprachen der Anwendungskomponenten unterscheiden. Offene verteilte Systeme zeichnen sich also durch ihre *Heterogenität* aus.

- **Fehlercharakteristik:** Die Fehlercharakteristik verteilter Systeme unterscheidet sich von der nicht-verteilter Systeme beträchtlich. Einzelne Knoten oder Kanten können ausfallen, während andere Knoten und Kanten weiterhin funktionieren. Die Verzögerung von Nachrichten kann nicht-deterministische Laufzeiten zur Folge haben.

2.2 Das Client/Server-Modell

Komponenten eines verteilten Systems können auf unterschiedliche Art und Weise miteinander kommunizieren. Das Grundproblem der Interaktion der Komponenten in einem verteilten System besteht darin, daß nicht nur Adreßraumgrenzen, sondern auch Rechnergrenzen durch die Kommunikation überwunden werden müssen. Unabhängig von der konkreten Realisierung des Datenaustausches existiert ein weit verbreitetes Interaktionsschema, das die Kommunikation in verteilten Systemen beschreibt: das *Client/Server-Modell* [21]. Dabei werden zwei Rollen unterschieden:

- **Klient (*Client*):** Klienten stellen den aktiven Teil einer Beziehung zwischen zwei Komponenten dar. Klienten treten an Dienste mit einem Auftrag heran, der die Diensterbringung (Service) initiiert.

- **Dienst (*Server*):** Dienste sind Komponenten, die Klienten eine bestimmte Dienstleistung (Service) anbieten. Dabei warten Dienste passiv auf einen Auftrag von einem Klienten, den sie ausführen. Damit ist eventuell die Übermittlung eines Ergebnisses an den Klienten verbunden.

Klient und Dienst sind Rollen, die von Komponenten wechselseitig angenommen werden können. So können Komponenten die Rolle wechseln, um beispielsweise als Klient eine Funktionalität in Anspruch zu nehmen, die sie zur Erfüllung ihrer Dienstrolle benötigen.

Das Client/Server-Modell kann durch verschiedene Mechanismen realisiert werden. Es stellt somit ein universelles Modell zur Beschreibung der Interaktion dar. Im Objektmodell kann der Austausch von Nachrichten zwischen Objekten durch das Client/Server-Modell beschrieben werden.

2.3 Das Objektmodell

Das Objektmodell stellt die Grundlage der objektorientierten Programmierung dar. Die objektorientierte Programmierung gehört neben der imperativen, der logischen und der funktionalen Programmierung zu den grundlegenden Paradigmen für die Softwareentwicklung in der Informatik [17].

2.3.1 Elemente des Objektmodells

Das Objektmodell nach Booch [17] besteht aus vier Haupt- und drei Nebenelementen. Hauptelemente des Objektmodells sind[2]:

- **Abstraktion:** Hiermit ist die essentielle Charakteristik eines Objekts gegenüber anderen Objekten gemeint.
- **Einkapselung:** Die für den Benutzer irrelevanten Details (Implementierung) eines Objekts werden durch die Einkapselung vor ihm verborgen.
- **Modularität:** Die logische Partitionierung eines Problems wird durch die Modularität reflektiert. Die Modularität kann durch sprachliche Konstrukte (Klassen) oder Verwaltungseinheiten (Übersetzungseinheiten, Dateien) unterstützt werden.
- **Hierarchie:** Logisch zusammenhängende Abstraktionen werden durch Hierarchien abgebildet. Hierarchien können durch Mechanismen wie Vererbung oder Delegation gebildet werden.

Booch zählt Typisierung, Nebenläufigkeit und Persistenz zu den Nebenelementen.

Die oben aufgeführten Elemente stellen Konzepte dar, die durch geeignete Mechanismen bzw. Sprachkonstrukte in einer objektorientierten Programmiersprache realisiert werden. Im folgenden soll auf das klassenbasierte Objektmodell eingegangen werden.

[2]Ein Modell ohne eines dieser vier Hauptelemente gilt nach Booch nicht mehr als objektorientiert.

2.3.2 Das klassenbasierte Objektmodell

Das Objektmodell wird in vielen Programmiersprachen durch die Bildung von Klassen und Instantiierung dieser zu Objekten umgesetzt. Beispiele solcher Sprachen sind Smalltalk [37], Java [31], C++ [90] oder Eiffel [64]. Grundlage des klassenbasierten Objektmodells ist ein Objekt. Booch definiert ein Objekt [17] wie folgt:

Ein Objekt besitzt Zustand, Verhalten und Identität. Der Zustand und das Verhalten ähnlicher Objekte ist in deren gemeinsamen Klassen definiert. Die Begriffe Instanz und Objekt sind austauschbar.

Klassen stellen Vorlagen dar, in denen Gemeinsamkeiten von Objekten modelliert werden. Durch das Instanziieren einer Klasse werden Objekte erzeugt. Klassen alleine sind nur auf der Meta-Ebene des Entwurfs und als Element der Programmiersprache existent und nicht als Einheiten zur Laufzeit[3]. In der Klasse werden die Namen und Typen der Attribute eines Objekts vereinbart. Ein Objekt speichert in diesen Attributen seinen Zustand. Das Verhalten eines Objekts bestimmt, wie es agiert und in Bezug auf Zustandsänderungen und Nachrichtenaustausch reagiert. Das Verhalten wird über die *Schnittstelle* des Objekts der Außenwelt, also potentiellen Benutzern des Objekts, zugänglich gemacht. Einkapselung wird dadurch errreicht, daß Objekte nur über die Schnittstelle Manipulationen des Zustands zulassen. Instanzen derselben Klasse sind durch die Identität unterscheidbar.

Die Partitionierung eines Problems in Klassen und Objekte alleine ist für die Bewerkstelligung einer Aufgabe nicht hinreichend. Eine Anwendung wird aus dem Zusammenspiel von Objekten gebildet, die kooperativ die Gesamtfunktionalität realisieren. Dazu müssen diese Objekte interagieren.

Die Interaktionen im Objektmodell steuern das Verhalten der Objekte. Jede Interaktion zwischen zwei Objekten induziert zwei Rollen. Der Klient C initiiert eine Aktion beim Dienst S. Diese Rollen entsprechen denen des Client/Server-Modells aus Abschnitt 2.2. Man spricht bei der Interaktion zwischen zwei Objekten vom *Versenden einer Nachricht* oder dem *Aufruf einer Methode bzw. Funktion.* Diese Sprechweisen sind synonym. In dieser Arbeit soll die Interaktion zwischen einem Klient und einem Dienst durch $C \rightarrow S.op$ ausgedrückt werden, wobei hier der Aufruf der Funktion *op* des

[3]Wobei hier Besonderheiten wie Metaklassen in Smalltalk bewußt ausgelassen werden.

Dienstes S durch den Klienten C dargestellt wird. Diese Interaktion kann mit Parametern zur Eingabe und Ausgabe der Funktion versehen sein.

2.4 Das verteilte Objektmodell

Die Interaktionen in verteilten Systemen werden häufig durch dieselben Rollen (Client/Server) wie die Interaktionen zwischen Objekten beschrieben. Bis auf das Element der Hierarchie finden sich die Kernelemente des Objektmodells in Client/Server-Systemen wieder. Diese Entsprechung erlaubt die Anwendung objektorientierter Entwurfsmethoden für die Modellierung verteilter Systeme. Die Umsetzung eines objektorientierten Entwurfs macht eine objektorientierte Zielsprache wünschenswert. Dadurch erfolgt kein Bruch zwischen der Modellierung und der Implementierung.

Bedingt durch die Verteilung der Objekte über Rechnergrenzen hinweg und die zugrundeliegende Heterogenität der Zielplattformen erschwert sich die Interaktion im verteilten Objektmodell. Damit eine Interaktion $C \rightarrow S.op$ durchgeführt werden kann, muß der Klient den Dienst lokalisieren können, damit der Aufruf zugestellt werden kann. Für die Zustellung über Rechnergrenzen hinweg muß eine Datenkonvertierung (Marshalling) die korrekte Wiederherstellung der Daten beim Zielsystem garantieren.

Sind die drei hier angesprochenen Punkte (Lokalisierung, Zustellung, Marshalling) gelöst, kann zwischen Klient und Dienst kommuniziert werden. Wesentlich für die einfache Benutzbarkeit ist dabei eine für den Anwendungsentwickler leicht zu nutzende Integration in die Programmierumgebung der Anwendung. Das bedeutet eine Einbettung in eine Programmiersprache durch Bibliotheken oder Erweiterung der Syntax oder Semantik der Programmiersprache. Es haben sich Verteilungsinfrastrukturen etabliert, welche die Implementierung und den Betrieb verteilter Systeme umfangreich unterstützen. Verteilungsinfrastrukturen für verteilte Objektsysteme basieren häufig auf einer Erweiterung des Objektmodells. Der folgende Abschnitt stellt ein exemplarisches verteiltes Objektmodell vor.

2.4.1 Architektur eines verteilten Objektmodells

Das hier vorgestellte Objektmodell basiert auf der Object Management Architecture (OMA) [70] der Object Management Group (OMG). Anderen

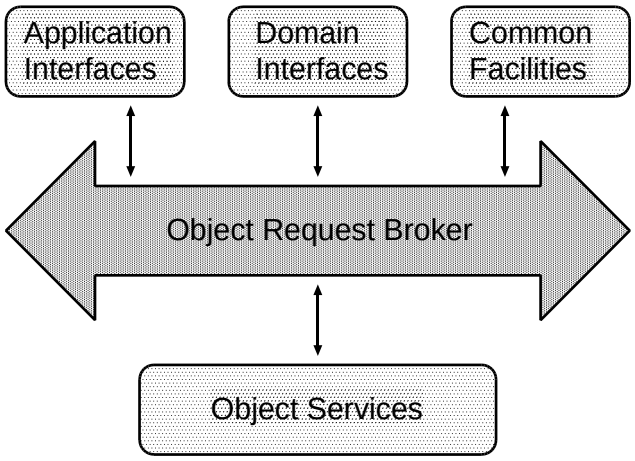


Abbildung 2.1: Object Management Architecture

Verteilungsinfrastrukturen, wie DCOM [28], liegt ein ähnliches Objektmodell zugrunde [20], so daß die OMA hier exemplarisch vorgestellt wird. Die OMA bildet die Basis der Common Object Request Broker Architecture (CORBA) – einer Verteilungsinfrastruktur für verteilte Objektsysteme.

Grundlage dieses Objektmodells ist die Definition eines Auftrags (Request) und die explizite Trennung der Schnittstelle eines Objektes von der Implementierung. Ein Auftrag enthält eine Referenz auf das Zielobjekt (Dienst), die vom Klienten gewünschte Operation und deren aktuelle Parameter. Die Interaktion im verteilten Objektmodell erfolgt durch den Austausch der Aufträge zwischen Klient und Dienst. Die Schnittstelle von Diensten wird in einer Schnittstellenbeschreibungssprache (interface definition language, IDL) spezifiziert. Zu der Schnittstelle eines Dienstes gehört der Name der Schnittstelle, die Operationen sowie die Parameter der Operationen und deren Typen. Die Entkopplung von Schnittstelle und Implementierung eines Objektes geschieht durch die IDL, welche in eine Zielsprache übersetzt wird. Diese Umsetzung ist dafür verantwortlich, daß die Interaktion durch Anfragen in die Zielsprache so eingebettet werden, daß sie so weit wie möglich der Syntax für lokale Objektinteraktion entsprechen.

Abbildung 2.1 zeigt die Basiselemente der OMA. Kernelement ist der sogenannte Object Request Broker (ORB), der die Interaktion zwischen den verteilten Objekten ermöglicht. Häufig wird der ORB auch als *Objektbus* bezeichnet. Diese Analogie ist berechtigt, da wie in einem Bus-System die

Hauptaufgabe des ORBs darin besteht, Ziele von Aufträgen zu lokalisieren (adressieren) und Aufträge weiterzuleiten.

Die durch den ORB verbundenen Objekte werden in der OMA in vier Gruppen eingeteilt:

- **Application Interfaces:** In dieser Gruppe finden sich die anwendungsspezifischen Objekte. Die Schnittstellen dieser Objekte werden typischerweise von Anwendungsentwicklern entworfen und implementiert.

- **Domain Interfaces:** Spezielle Anwendungsbereiche, wie Telekommunikation, werden durch maßgeschneiderte Dienste unterstützt. Diese Dienste sind in der Regel für andere Anwendungsbereiche nicht relevant.

- **Common Facilities:** Für die Entwicklung größerer Systeme wiederkehrende Aufgaben sind als Rahmenwerk im Bereich der Common Facilities gelöst. Im Gegensatz zu den anwendungsspezifisch maßgeschneiderten Diensten der Domain Interfaces stellen diese Objekte ein Rahmenwerk zur Anpassung dar.

- **Object Services:** Ziel der Object Services ist das Bereitstellen von Funktionalität, die den Kern des ORBs erweitert, aber nicht für alle Anwendungen von Belang ist (Zeit-Dienst, Transaktionen, Ereignisse etc.).

Diese Gruppen stellen lediglich eine Strukturierung der Objekte nach ihrem Anwendungszweck dar. Aus Sicht des verteilten Objektmodells sind diese als Dienste durch ihre jeweilige Schnittstelle zu erreichen. Das von der OMA abstrakt vorgegebene Objektmodell muß entsprechend realisiert werden, damit es nutzbar ist. Dazu notwendig ist eine Spezifikation, in der die Schnittstellenbeschreibungssprache, deren Umsetzung in Zielsprachen, Schnittstelle des ORBs sowie Protokolle zur Kommunikation zwischen den ORBs definiert sind. Die OMG hat mit der Common Object Request Broker Architecture (CORBA) [75] eine solche Spezifikation herausgegeben.

2.4.2 Objektorientierte Middleware

Verteilungsinfrastrukturen werden auch oft durch den Begriff *Middleware* bezeichnet. Objektorientierte Middleware realisiert ein verteiltes Objekt-

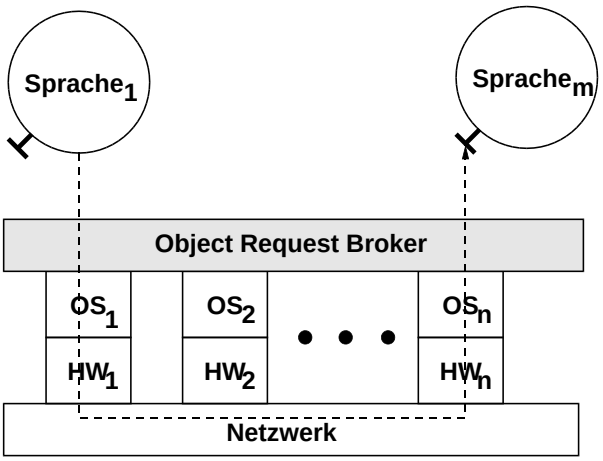


Abbildung 2.2: Middleware-Architektur

modell. Die dabei wesentlichen Elemente sind der Object Request Broker und die Schnittstellenbeschreibungssprache sowie deren Umsetzung in Zielsprachen.

In Abbildung 2.2 ist der exemplarische Aufbau einer Middleware-Architektur dargestellt. Zwei in potentiell unterschiedlichen Sprachen realisierte Objekte kommunizieren miteinander. Die Objekte können auf unterschiedlichen Plattformen ablaufen. Eine Abstraktion oberhalb des Betriebssystems gleicht diese unterschiedlichen Plattformen aus. In der Abbildung ist dies der Object Request Broker (ORB). Dieser bildet eine horizontale Abstraktion, die darauf implementierten Objekten eine gemeinsame Schnittstelle zur Kommunikation bietet. Der ORB lokalisiert entfernte Objekte und leitet Nachrichten weiter.

Zur Laufzeit müssen die Objekte zwischen potentiell heterogenen Plattformen Daten austauschen. Dafür ist ein standardisiertes Protokoll erforderlich. Man spricht dann von der Interoperabilität dieser Plattformen.

Im folgenden wird eine objektorientierte Middleware-Plattform in ihren Komponenten vorgestellt. Hierbei werden die Komponenten entsprechend der Phasen der Systementwicklung für die Unterstützung der Design- und Implementierungs-Phase und der Laufzeitunterstützung unterschieden.

2.4.3 Design und Implementierung

Basis für die Interaktion von Objekten, die in verschiedenen Programmiersprachen implementiert sind, ist eine gemeinsame, programmiersprachenunabhängige Schnittstellenbeschreibungssprache.

Ein IDL-Compiler übersetzt die IDL-Definitionen in Objekte einer konkreten Programmiersprache. Typischerweise sind diese Objekte schon mit dem notwendigen Verhalten für die Kommunikation mit dem ORB versehen, so daß der Programmierer nur noch das Objekt-Verhalten implementieren muß. Folgt die Umsetzung der IDL in eine konkrete Sprache immer demselben Schema mit zugrundeliegendem Application Programming Interface (API), ist durch die so realisierte Abstraktion die Portabilität des Codes gegeben.

Es werden entsprechende Objekte für die Klienten- und die Dienst-Seite generiert. Diese Umsetzung muß für jede verwendete Zielsprache erfolgen und sichert somit die Trennung von Schnittstelle eines Objektes und deren Implementierung zu. Der nächste Unterabschnitt charakterisiert die Kommunikation und erläutert, welche Objekte involviert sind.

2.4.4 Laufzeit

Für die Interaktion über Adreßraumgrenzen hinweg und insbesondere zwischen verschiedenen Plattformen muß eine geeignete Form der Interprozeßkommunikation etabliert werden. Diese ist einzukapseln, damit in der jeweiligen objektorientierten Programmiersprache die Verteilung nicht sichtbar ist. Dies erfordert neben der Schnittstellenbeschreibungssprache auch weitere Laufzeitunterstützung durch die Middleware.

Abbildung 2.3 zeigt exemplarisch typische Komponenten einer Middleware und deren Zusammenarbeit. Ein Klient initiiert eine Aktion bei einem Dienst. Im lokalen Fall erfolgt dies durch Aufruf einer Methode auf dem Dienst-Objekt. Sind Klient und Dienst durch Adreßräume getrennt, muß Kommunikation über den Austausch von Nachrichten stattfinden. Der IDL-Compiler erzeugt aus den Schnittstellendefinitionen entsprechende Objekte für Klienten- und Dienst-Seite. Im Bild sind dies das *Proxy*- und das *Skeleton-Objekt*.

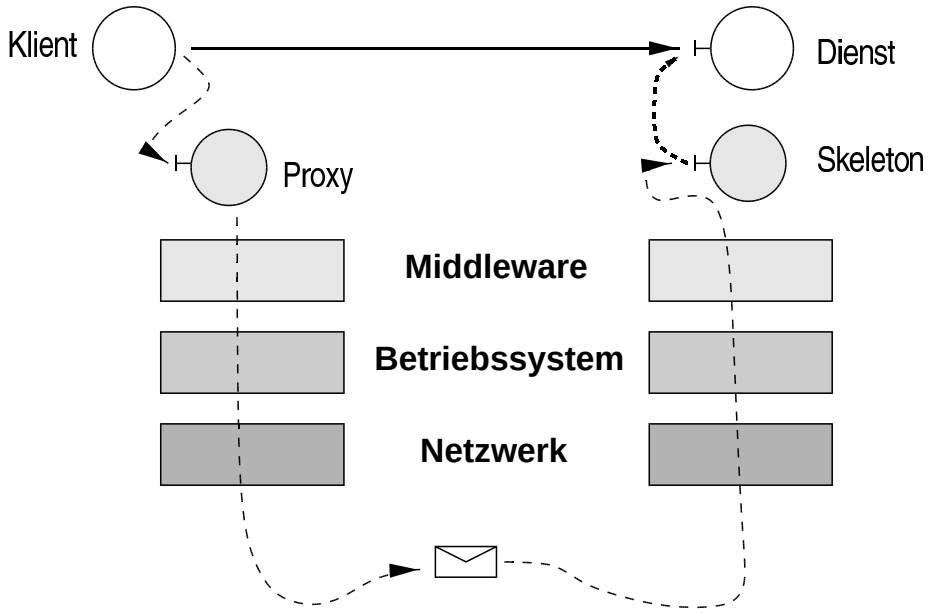


Abbildung 2.3: Middleware-Komponenten

Proxy

Das Stellvertreter-Objekt (Proxy oder Stub) auf der Klientenseite fungiert für den Klienten als lokales Dienst-Objekt. Alle Anfragen an das Dienst-Objekt werden vom Klient an das Proxy gestellt. Dieses ist nun verantwortlich, die Anfrage an das Dienst-Objekt weiterzuleiten. Das Proxy generiert für jeden Aufruf eine Nachricht, welche die Anfrage beinhaltet und schickt diese an das Dienst-Objekt.

Skeleton

Das Pendant zum Proxy auf der Klientenseite ist das Skeleton. Einkommende Nachrichten von Proxy-Objekten werden empfangen und an das Dienst-Objekt weitergeleitet. Damit die Behandlung der einkommenden Nachrichten für das Dienst-Objekt transparent ist, wird dafür notwendiges Verhalten durch das Skeleton dem Dienst mittels Mechanismen wie Vererbung hinzugefügt.

ORB

Das Verschicken von Nachrichten, die Konvertierung zwischen verschiedenen Datenformaten und die Lokalisierung von Objekten sind wiederkehrende Aufgaben in der Netzwerkprogrammierung. Ohne eine geeignete Zwischenschicht müßten diese Aufgaben in jedem Proxy und jedem Skeleton implementiert werden. Es ist daher naheliegend, diese Aufgaben in eine eigene Schicht auszulagern, die Objekten darüber zur Verfügung steht. In Abbildung 2.3 ist diese Schicht durch die Blöcke "Middleware" dargestellt. Die zentrale Komponente in objektorientierter Middleware (vgl. Abbildung 2.2) ist der Object Request Broker (ORB). Aufgaben eines ORBs sind:

- **Lokalisieren von Objekten:** Durch die Identität (Objektreferenz) des Dienstes muß der ORB einen Aufruf vom Klient zum Dienst weiterleiten können. Dazu ist insbesondere der Ort des Dienst-Objekts aufzulösen.

- **Weiterleiten von Aufrufen:** Stellt ein Klient eine Anfrage an ein lokales Proxy-Objekt, muß neben der Ortsbestimmung des korrespondierenden Objektes der Aufruf weitergeleitet werden und die Antwort in eine lokale Antwort des Proxys an den Klienten gewandelt werden.

- **Interoperabilität:** Sind Klient und Dienst auf unterschiedlichen Plattformen implementiert, müssen Datentypen in ein einheitliches Format gewandelt werden. Insbesondere ist für die Interoperabilität ein einheitliches Nachrichtenformat von Aufrufen notwendig.

- **Einheitliche Schnittstelle:** Um Objekte zwischen verschiedenen ORBs portierbar zu halten, sollten die Schnittstellen zu dem Proxy bzw. Skeleton einheitlich sein, wie auch die sonstigen Schnittstellen zur Verwaltung.

Die Realisierung eines ORBs kann durch verschiedene Mechanismen geschehen. Eine Einbindung in die Anwendung über den Binder (library-based ORB) ist genauso möglich, wie die Kommunikation über lokale Interprozeß-Kommunikation mit einem ORB als eigenständigem Prozeß (server-based ORB).

Das Ziel von Middleware-Plattformen ist die weitgehende *Verteilungstransparenz* gegenüber der Anwendung. Durch die Verwendung von geeigneten Abstraktionen, wie dem Proxy und dem Skeleton, ist es dem Nutzer

von Objekten durch das Proxy verborgen, wo das Objekt, das eine Aktion ausführt, lokalisiert ist. Analog stellt das Skeleton für Dienste die notwendige Funktionalität bereit, so daß diese nur die funktionale Schnittstelle implementieren und die Nutzung durch entfernten Klienten transparent für den Diensterbringer geschieht.

2.5 Verteilungstransparenz

Objektorientierte Middleware-Plattformen bieten dem Programmierer scheinbare Verteilungstransparenz. Im Reference Model of Open Distributed Processing (RM-ODP [42]) wird Verteilungstransparenz nach verschiedenen Gesichtspunkten klassifiziert:

- **Zugriffstransparenz:** Der Zugriff auf entfernte und lokale Dienste erfolgt auf dieselbe Art und Weise. Entsprechende Einbettung in die Aufrufmechanismen oder Stellvertreterobjekte ermöglichen dies.
- **Fehlertransparenz:** Das mögliche Fehlverhalten oder das Versagen eines Dienstes wird nach außen verborgen. Klienten solcher Dienste können von ordnungsgemäßer Funktion der Dienste ausgehen und benötigen insbesondere keine weiteren Vorkehrungen zur Fehlerbehandlung.
- **Ortstransparenz:** Die Bindung eines Klienten an einen Dienst benötigt keine Informationen über den Ort des Dienstes.
- **Migrationstransparenz:** Die Erbringung eines Service ist unabhängig vom Ort des Dienstes. Dies ermöglicht die Migration des Dienstes.
- **Bindungstransparenz:** Die Bindung einer Schnittstelle an einen Dienst, der den Service realisiert, kann zu anderen Diensten wechseln, ohne daß Klienten dies bemerken.
- **Replikationstransparenz:** Der Zugriff auf einen ursprünglichen Dienst oder seine Replikate sind gleich und unterscheiden sich nicht im Service.
- **Persistenztransparenz:** Das Aktivieren eines Dienstes bei Bedarf, also bei einer Anfrage, und die darauffolgende Deaktivierung werden vor Klienten verborgen.

- **Transaktionstransparenz:** Die Durchführung von Abstimmungsprotokollen, um transaktionelles Verhalten innerhalb einer Gruppe von Diensten zu gewährleisten, wird gegenüber Klienten maskiert.

Es ist fraglich, ob Verteilungstransparenz in einem solchem Maße wünschenswert und überhaupt technisch realisierbar ist. Gerade die Vielzahl an Fehlermöglichkeiten in offenen verteilten Systemen macht entsprechende Vorkehrungen notwendig, die an das Verhalten und die Designziele der Anwendung angepaßt sind. Waldo et. al. argumentieren in [101], daß Verteilungstransparenz ein unerreichbares Ziel ist. In diesem Abschnitt soll kurz anhand einiger Beispiele motiviert werden, daß unterschiedliches Verhalten zur Behandlung von Verteilungsaspekten notwendig ist.

Die unterschiedlichen Fehler- und Störquellen von verteilten Systemen in Kombination mit verschiedenen Anwendungsanforderungen erschweren eine allgemeine Definition und Klassifikation. Prinzipiell lassen sich bei einer Interaktion von zwei Objekten beispielsweise folgende Probleme feststellen:

- **Nichterhalt eines Ergebnisses:** Ein Klient, der kein Ergebnis einer Operation erhält, kann dadurch blockiert sein bzw. nie sicher sein, ob die Operation ausgeführt wurde.

- **Verzögerung der Bearbeitung:** Die Bearbeitung von Operationen kann aus der Sicht des Klienten nicht-deterministische unterschiedliche Dauer haben.

- **Offenlegung von vertraulichen Aufrufen:** Daten, die zwischen Klient und Dienst ausgetauscht werden, können von Dritten verfälscht oder ausspioniert werden, wenn die Netzwerkverbindung zwischen den beteiligten Objekten nicht besonders geschützt ist.

Die Ursachen und die damit verbundenen Mechanismen zur Vermeidung der oben genannten Probleme sind vielfältig und eng mit der zugrundeliegenden Architektur des verteilten Systems und der darauf ablaufenden Software verknüpft:

- **Partielle Fehler:** Im Gegensatz zu nicht-verteilten Systemen, die ein einheitliches Fehlerverhalten an den Tag legen, können Ausfälle

von Komponenten in einem verteilten System unterschiedliche Auswirkungen auf die Gesamtfunktionalität haben. Durch einen Ausfall können Störungen in der Kommunikation auftreten, wenn z. B. die Netzwerkverbindung zwischen zwei Objekten unterbrochen wird. Andere Objekte können dennoch ihre Arbeit fortsetzen. Zu den möglichen Ausfällen gehören sowohl Kanten als auch Knoten. Je nach Netzwerk, Auslastung und gewähltem Protokoll sind auch nur Störungen der Übertragungen möglich, die sich in Verlusten von Nachrichten äußern können.

- **Dynamische Leistungsschwankungen:** Die unterschiedliche Auslastung von Komponenten und deren dynamische Änderung können zu verzögerter Auslieferung von Anfragen führen. Die Bearbeitung von Anfragen durch Dienst-Objekte kann auch aufgrund erhöhter Last auf Knoten unterschiedliche Laufzeiten aufweisen.

- **Ressourcen-Konflikte:** Die Konkurrenz um gemeinsame Ressourcen kann bei Belegung einer Ressource zu Verzögerung bis hin zur Nicht-Ausführung von Anfragen führen. Auch Verklemmungen sind bei der Nutzung verteilter Ressourcen möglich. Vorkehrungen werden aufgrund des dynamischen Auftretens solcher Konflikte erschwert.

- **Nicht-Determinismus:** Verteilte Systeme sind in einem starken Maße nicht-deterministisch. So können Nachrichten in unterschiedlichen, nicht vorhersehbaren Reihenfolgen zugestellt werden. Die internen Details (Zustand, Implementierung) eines Dienstes sind für den Klienten nicht ersichtlich. Als Resultat können scheinbar gleiche Dienste unterschiedliche Ergebnisse auf dieselbe Anfrage liefern.

Um in einer solchen Umgebung – je nach Anwendungsanforderung – dennoch korrektes Verhalten zu bieten, müssen die beteiligten Objekte Maßnahmen zur Sicherstellung der Anwendungsanforderungen treffen. Dies bedeutet die Aufgabe der Verteilungstransparenz, da solche Maßnahmen nur mit Informationen über die Verteilung realisierbar sind.

Im Sinne der Einkapselung und der Abstraktion werden wesentliche Elemente des Objektmodells durch diese Modellierung verletzt. Die Effekte auf so modellierte Objekte sind in Hinblick auf Universalität und Wiederverwendbarkeit störend.

Es ist daher wünschenswert, geeignete Abstraktionen für die Modellierung und Einkapselung von Mechanismen zur Begegnung solcher Effekte zu finden und in eine objektorientierte Middleware zu integrieren.

Im weiteren Verlauf der Arbeit wird der Begriff der Dienstgüte eingeführt und der Bereich von Dienstgüte-Management abgegrenzt. Die Einbettung von Dienstgüte-Management in objektorientierte Middleware bietet eine Lösung zu den oben aufgeworfenen Strukturproblemen.

Kapitel 3

Dienstgüte

Unter dem Begriff der Dienstgüte (Quality of Service, QoS), bzw. des Dienstgüte-Managements, werden Lösungsansätze zu den in Kapitel 2 geschilderten Problemen verteilter Systeme subsumiert. Betrachtet wird neben der eigentlichen Diensterbringung, dem funktionalen Bestandteil einer Diensterbringung, auch die nicht-funktionalen mit der Güte bzw. Qualität der Diensterbringung verbundenen Effekte. In diesem Kapitel werden grundlegende Definitionen eingeführt. Anforderungen an das Dienstgüte-Management und die Besonderheiten der Dienstgüte-Integration bei einem zugrundeliegenden Objektmodell werden diskutiert, und abschließend wird ein kurzer Abriß zum Stand der Forschung präsentiert.

3.1 Überblick

Die Bewertung einer Dienstleistung unterliegt zwei Kriterien. Neben dem Resultat der Dienstleistung kann auch die Art und Weise, in der diese erbracht wird, bewertet werden. Der erste Fall soll als der *funktionale Aspekt* der Dienstleistung bezeichnet werden, während der zweite die *nicht-funktionalen Aspekte* der Dienstleistung darstellt.

Die funktionalen Aspekte einer Interaktion zwischen Klient und Dienst sind durch die Schnittstelle des Dienstes manifestiert. In der Schnittstelle werden die zulässigen Operationen eines Dienstes sowie deren Ein- und Ausgabeparameter festgelegt. Unter dem Begriff Dienstgüte lassen sich die qualitätsbezogenen, nicht-funktionalen Aspekte der Diensterbringung klassifizieren.

In [21] wird Dienstgüte unter den Gesichtspunkten *Performance*, *Reliability* und *Availability* sowie *Security* eingeführt. Entscheidend ist hier wiederum die Sicht des Nutzers eines Dienstes. Die in Abschnitt 2.5 vorgestellten Varianten der Verteilungstransparenz stellen zum Teil schon bestimmte Dienstgüten bzw. Mechanismen zur Realisierung bestimmter Dienstgüten dar. Fehlertransparenz bedeutet, daß der Klient einen fehlerfreien Service von einem Dienst angeboten bekommt. Dies ist eine andere "Qualität" als ein potentiell fehleranfälliger Service. Mechanismen zur Etablierung von Migrationstransparenz können zur Lastbalancierung benutzt werden, indem ein Dienst auf einen nicht ausgelasteten Rechner migriert wird.

Das "Quality of Service Framework" der ISO [44] definiert Quality of Service als:

a set of qualities related to the collective behaviour of one or more objects.

Offen in dieser Definition ist die Bewertung der "Qualität" des Verhaltens. Die ITU-T konkretisiert den Qualitätsbegriff in E.800 [96] als Kombination aus "service support performance", "service operability performance", "service ability performance" und "service security performance" sowie anderer Service-spezifischer Faktoren. Diese Faktoren bestimmen den nichtfunktionalen Anteil der Diensterbringung. Bei gleichem funktionalem Anteil der Diensterbringung (Service) werden durch verschiedene Qualitäten die Quality of Service – Dienstgüte – der Diensterbringung determiniert.

Die folgenden Beispiele skizzieren den Bereich von Dienstgüte; sie stellen aber keine vollständige Abgrenzung von Dienstgüte dar.

Ein Beispiel für Dienstgüte aus dem Bereich der Computernetzwerke liefert das Transmission Control Protocol (TCP) [78]. Im Gegensatz zu darunterliegenden Protokollen, wie dem Internet Protocol (IP), das einen verbindungslosen best-effort[1] Dienst bereitstellt, bietet TCP zuverlässige, verbindungsorientierte Kommunikation durch Bestätigungen und Flußkontrolle. Dabei wird das zugrundeliegende IP benutzt, und die TCP-Protokollschicht realisiert Nachrichten-Wiederholung bei Datenverlust und eine Anpassung an die verfügbare Bandbreite durch ein Schiebefensterprotokoll.

[1] Best effort, wörtlich "beste Bemühung", bedeutet hier ohne Zusage über die Zustellung einer Nachricht

Diese Anpassung geschieht transparent für den Programmierer durch das Protokoll.

Die Verfügbarkeit von Diensten ist für viele Klienten in verteilten Systemen von großer Wichtigkeit. So kann der Ausfall eines Dienstes einen "single-point-of-failure" darstellen und den Gesamtausfall des Systems nach sich ziehen. Die Bildung von Replikatgruppen ist ein Mechanismus, um Verfügbarkeit zu gewährleisten. Das ISIS-System [15] bietet die notwendige Funktionalität, um Dienstgruppen zu verwalten, konsistente Zustände in der Gruppe zu gewährleisten (virtuelle Synchronität) und die Gruppenkommunikation zu steuern.

Echtzeitsysteme stellen andere Anforderungen als die zuvor genannten. So ist der Ausfall eines Dienstes ggf. einer verzögerten Antwort vorzuziehen, solange noch ein Dienst bereit steht, der die Anfrage im gewünschten Zeitrahmen bearbeitet. Architekturen, welche die Integration von Echtzeitgesichtspunkten und Middleware bewerkstelligen, müssen neben Problemen wie der Übertragung von Daten über das Netzwerk auch die Auslieferung an die Anwendung lösen. Dies führt zu komplexen interagierenden Komponenten vom Netzwerk, über das Betriebssystem bis hin zur Anwendung. The ACE ORB (TAO) [85] ist ein Beispiel für eine solche Architektur, die in CORBA Echtzeitgesichtspunkte integriert.

Eine wichtige Domäne von Dienstgüte ist der Multimedia-Bereich. Multimedia-Systeme operieren auf kontinuierlichen Datenströmen, z.B. für Audio- oder Video-Daten. Wichtige Parameter für solche Systeme sind die *Kontinuität* der Übertragung und die *Synchronität* von zusammengehörigen Strömen, wie Audio- und Video-Strom bei einem Bildtelephon.

Im Bereich numerischer Berechnungen kann Leistungsfähigkeit durch Parallelisierung erreicht werden. Aber auch die Verwendung bestimmter numerischer Algorithmen kann eine höhere Genauigkeit – bspw. Intervallarithmetik – oder Berechnungsgeschwindigkeit zusichern und ist ein nichtfunktionaler, qualitätsbezogener Bestandteil der Diensterbringung. Dienstgüte ist aber nicht nur von Leistungsgesichtspunkten bzw. der Begegnung von Kapazitätsengpässen abhängig. Ein Vermittlungsdienst mit einer größeren Datenbasis an Angeboten offeriert eine andere Qualität – also Dienstgüte – als einer anderer mit derselben Funktionalität, aber mit einer kleineren Datenbasis. Hingegen ist die Form der Zugriffs ein funktionaler Aspekt. Beispielsweise könnte ein Dienst – ähnlich den Suchmaschinen des WWW –

nur eine einfache Schlagwortsuche zulassen, ein anderer aber durch reguläre Ausdrücke eine höhere Ausdruckskraft der Suchausdrücke erlauben.

Auch wenn der Fokus von Dienstgüte-Vorkehrungen originär im Netzwerkbereich liegt und sich dies durch neue Medien und deren Technologien gerade im Multimediabereich fortsetzt, ist doch von Standardisierungsgremien und Industriekonsortien die allgemeinere Bedeutung von Dienstgüte-Management erkannt worden. Dies manifestiert sich in den folgenden Definitionen, die an die Definitionen der ITU-T (E.800 [96]), OMG (QoS green paper [71]) und der ISO (QoS framework [44]) angelehnt sind.

3.2 Definitionen

Dienstgüte-Charakteristik

Eine *Dienstgüte-Charakteristik* ist eine quantifizierbare Einheit, durch die ein Teil der Dienstgüte-Merkmale eines Systems beschrieben wird. Der aktuelle Wert einer Dienstgüte-Charakteristik ist an eine Klient/Dienst-Interaktion gebunden. Dienstgüte-Charakteristiken sind konzeptionelle Einheiten, die keine unmittelbare Entsprechung im System besitzen müssen. So kann eine Dienstgüte-Charakteristik, die Latenzzeiten der Übertragung abbildet, durch Messungen genähert und durch entsprechende Dienstgüte-Parameter repräsentiert werden.

Beispiele für Dienstgüte-Charakteristiken sind Verfügbarkeit und Antwortzeit. Im Sinne der Quantifizierbarkeit können beide unterschiedlich definiert werden. Verfügbarkeit kann zum einen als der Quotient aus "Uptime" und der Summe von "Uptime" und "Downtime" definiert werden – also basierend auf Daten, die statistisch ermittelt bzw. gemessen werden – oder aber durch bestimmte Parameter eines zugrundeliegenden Dienstgüte-Mechanismus in Form von "Anzahl verfügbarer Replikate". Analog kann die Antwortzeit in Abhängigkeit des zugrundliegenden Dienstgüte-Mechanismus als Ende-zu-Ende-Latenz des Auftrags oder aber durch die Anzahl verfügbarer Dienste und zugeteilter Klienten bestimmt werden.

Dienstgüte-Kategorie

Dienstgüte-Kategorien stellen Sammlungen von Dienstgüte-Charakteristiken derselben Anwendungsklasse dar. Beispiele dafür sind Klassen für

Echtzeit, Multimedia oder Fehlertoleranz. Dienstgüte-Kategorien stellen nur ein Klassifikationsmerkmal dar und werden nicht durch besondere Einheiten im System umgesetzt.

Dienstgüte-Parameter

Die Repräsentation von Dienstgüte-Charakteristiken im System wird durch *Dienstgüte-Parameter* vorgenommen. Dabei existiert keine Eins-zu-Eins-Zuordnung zwischen Dienstgüte-Charakteristiken und Dienstgüte-Parametern. Dienstgüte-Parameter für die Güte einer Videoübertragung können in Qualitätsstufen [48] wie $\{gut, mittel, schlecht\}$ ausgedrückt oder aber durch systemnahe Werte, wie Bandbreite und Verzögerung, dargestellt werden. Dienstgüte-Parameter werden benutzt, um Soll-Werte der Dienstерbringung festzulegen, wie auch, um den Ist-Wert einer Charakteristik (*Dienstgüte-Niveau*) im System abzufragen.

Dienstgüte-Mechanismus

Die Durchsetzung von Dienstgüte-Anforderungen im System erfolgt durch *Dienstgüte-Mechanismen*. Diese steuern die Einhaltung einer zwischen Klient und Dienst ausgehandelten *Dienstgüte-Vereinbarung*. Dienstgüte-Mechanismen können sowohl als Bibliotheken (Multicast- und Gruppenkommunikation), Funktionalität des Betriebssystem (Ablaufsteuerung von Prozessen mit Zeitabhängigkeiten), Eigenschaften des Netzwerks (Bandbreitenreservierung) oder durch Hardware-Zusätze des Rechners (Kompressions- oder Verschlüsselungschips) realisiert werden. Oft wird eine Kombination aus mehreren dieser Dienstgüte-Mechanismen für eine konkrete Dienstgüte-Implementierung benutzt.

Dienstgüte-Vereinbarung

Aufgrund der Dynamik in verteilten Systemen ist eine Festlegung einer bestimmten Dienstqualität ohne Einbezug der aktuellen Gegebenheiten im System nicht möglich. Eine Dienstgüte-Vereinbarung spiegelt die Einigung von Klient und Dienst auf eine bestimmte Ausprägung einer Dienstgüte-Charakteristik wider. Diese wird typischerweise durch eine Dienstgüte-Verhandlung etabliert.

Dienstgüte-Architektur

Eine *Dienstgüte-Architektur* ist eine Verteilungsinfrastruktur, die Interaktionen zwischen Klienten und Diensten mit Dienstgüte-Zusicherungen ermöglicht. Wesentliche Aufgabe einer Dienstgüte-Architektur ist die Bestimmung und die Durchsetzung von Dienstgüte-Vereinbarungen. Dienstgüte-Architekturen lassen sich anhand der folgenden Kriterien klassifizieren:

- **Generizität:** Generische Dienstgüte-Architekturen erlauben die freie Definition von Dienstgüte-Charakteristiken. Dies bedeutet die Abbildung der Dienstgüte-Charakteristiken auf Dienstgüte-Parametern und die Integration der zugehörigen Dienstgüte-Mechanismen. Im Gegensatz dazu bieten nicht-generische Dienstgüte-Architekturen nur einen festgelegten, nicht veränderbaren Satz von Dienstgüte-Parametern und -Mechanismen.

- **Mehrkategorie:** Abhängig davon, ob eine Dienstgüte-Architektur auf eine Dienstgüte-Kategorie eingeschränkt ist oder aber Dienstgüte-Charakteristiken verschiedener Dienstgüte-Kategorien anbietet, soll von Mehrkategorie- bzw. Einzelkategorie-Dienstgüte-Architekturen gesprochen werden.

- **Dynamik:** Das Spektrum der Dynamik in Dienstgüte-Architekturen ist weitreichend. So kann die Zuordnung von Dienstgüte-Charakteristiken zu Diensten zur Übersetzzeit geschehen oder noch zur Laufzeit möglich sein.

3.3 Notation

Für den weiteren Verlauf der Arbeit soll die folgende Notation in der Präsentation für eine prägnante Darstellung benutzt werden.

- Kleine griechische Buchstaben (ϕ, δ, etc.) bezeichnen Dienstgüte-Charakteristiken.

- Das Tupel $< D, Q >$ mit $Q = \{\phi_1, \ldots, \phi_n\}$ drückt die Bereitschaft eines Dienstes D aus, seine Diensterbringung mit Dienstgüte-Charakteristiken aus der Menge Q zu leisten. Diese Erbringung ist aber

jeweils von den gegebenen Ressourcen-Verhältnissen im System abhängig. Dabei bildet D den funktionalen Anteil des Service durch den Dienst und Q die möglichen nicht-funktionalen Anteile durch die Dienstgüte-Charakteristiken.

- Eine Beziehung zwischen Klient und Dienst wird durch das Tupel $[K, D]$ ausgedrückt. Man spricht dabei auch von der Bindung des Klienten an den Dienst.
- Eine Bindung zwischen einem Klient und einem Dienst mit Dienstgüte-Befähigung wird mit $[K, D, Q]$ bezeichnet.
- Durch eine Verhandlung wird eine Dienstgüte-Vereinbarung zwischen einem Klienten K und einem Dienst D erzielt: $[K, D, \overline{Q}]$, $\overline{Q} = \{\overline{\phi}_j, \ldots, \overline{\phi}_k\}$. Dabei ist $\overline{\phi}_l$ ein Dienstgüte-Niveau der Dienstgüte-Charakteristik ϕ_l und $1 \leq j \leq k \leq n$.
- Ein Mechanismus, der eine Dienstgüte-Charakteristik ϕ im System implementiert, wird durch M_ϕ dargestellt. Dabei ist zu beachten, daß dies keine Eins-zu-Eins-Zuordnung darstellt, d.h. eine Dienstgüte-Charakteristik kann durch verschiedene Mechanismen implementiert werden.
- Die Schnittstelle eines Dienstgüte-Mechanismus wird durch $S(M_\phi)$ bezeichnet. Analog wird für die Schnittstelle eines Dienstes $S(D)$ verwendet.
- Parameter eines Dienstgüte-Mechanismus werden durch $P(M_\phi)$ bezeichnet.

3.4 Eigenschaften von Dienstgüte

Für die Betrachtung von Dienstgüte-Erbringung sind zwei Eigenschaften wesentlich: zum einen die klientenbezogene Bewertung und zum anderen die Ende-zu-Ende-Sicht auf die Dienstgüte-Erbringung.

3.4.1 Ende-zu-Ende-Sicht

Die Kommunikation in verteilten Systemen zwischen Klient und Dienst erfolgt entlang eines Kommunikationspfades, der verschiedene Software- und

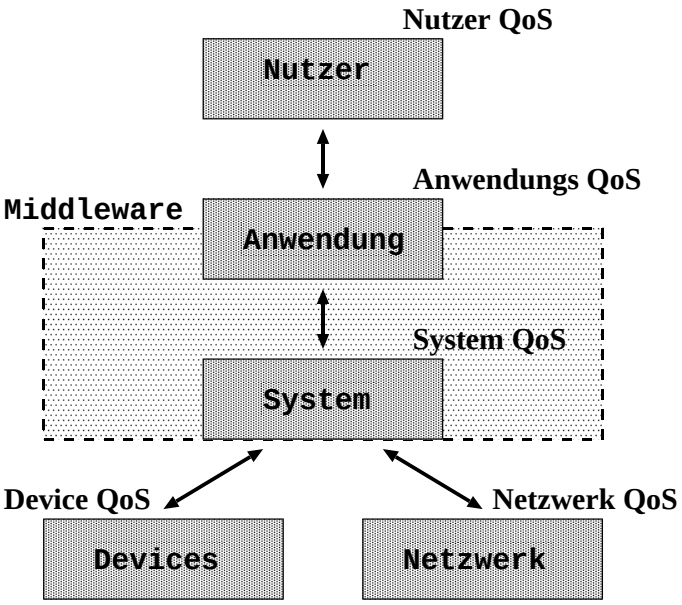


Abbildung 3.1: Hierarchien von Dienstgüte-Mechanismen

Hardware-Schichten durchläuft. So liegen auf dem Kommunikationspfad die Middleware, das Betriebssystem und die Netzwerksoftware und -hardware. Dienstgüte-Mechanismen finden sich entsprechend entlang des Kommunikationspfades. Abbildung 3.1 zeigt die in [100] vorgestellte Hierarchie von Dienstgüte-Mechanismen. Dabei müssen die Parameter der Dienstgüte-Mechanismen einer Schicht n auf die zugrundeliegende Schicht $n-1$ abgebildet werden.

Die Dienstgüte-Mechanismen einer Schicht n beruhen auf den Mechanismen von Schicht $n-1$. Versagt eine unterliegende Schicht, können im allgemeinen die darüber liegenden ihre Dienstgüte-Erbringung nicht leisten.

Darüber hinaus erlauben Hierarchien die Strukturierung von Dienstgüte-Mechanismen und die Wiederverwendbarkeit einzelner Schichten. Abbildung 3.2 zeigt zwei exemplarische Konstellationen. Dabei ist im ersten Fall die Struktur durch einen Mechanismus M_i auf Schicht i und zwei Mechanismen $M_{i-1}1$ und $M_{i-1}2$ auf der zugrundeliegenden Schicht $i-1$ gebildet. M_i kapselt das Verhalten der zugrundeliegenden Schichten ein. Ein Beispiel für solche Einkapselungen findet sich im Bereich unterschiedlicher Netzwerkprotokolle. So kann eine höhere Schicht Bandbreitereservierung virtualisieren und auf zugrundeliegende, netzwerkbezogene Schichten — wie ATM (Asynchronous Transfer Mode) [92], [89] oder RSVP (Resource Reservati-

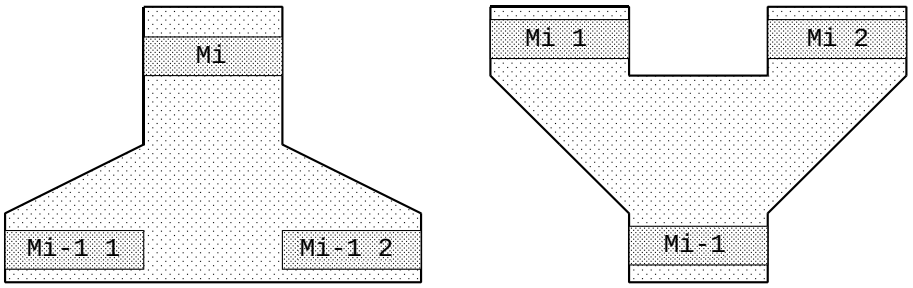


Abbildung 3.2: Struktur der Hierarchien von Dienstgüte-Mechanismen

on Protocol) [102] — abbilden. Im zweiten Fall wird ein Basismechanismus M_{i-1} der Schicht $i-1$ von zwei darüberliegenden Mechanismen M_i1 und M_i2 benutzt. So kann ein Multicast-Mechanismus auf der Transportschicht verwendet werden, um sowohl Fehlertoleranz durch k-Ausfallsicherheit oder Diversität durch Mehrheitsentscheid zu realisieren.

Die Ende-zu-Ende-Sicht auf die Dienstgüte-Erbringung spiegelt sich in der Struktur tieferer Schichten entsprechend wider. Als Beispiel soll hier das Nemesis-Projekt [82] genannt werden. Dort wird ein Mikrokern-Betriebssystem mit Dienstgüte-Befähigung, hauptsächlich für die Multimedia-Kategorie, versehen. Auf einem Mikrokern aufbauend, wird eine Hierarchie von Dienstgüte-Mechanismen bereitgestellt.

3.4.2 Klientenbezogene Bewertung

Die Erbringung eines bestimmten Dienstgüte-Niveaus unterliegt einer klientenbezogenen Bewertung. So kann das erbrachte Dienstgüte-Niveau eines Dienstes für bestimmte Klienten ausreichend sein und für andere wiederum nicht. Insbesondere bei der Verhandlung über das zu erbringende Dienstgüte-Niveau ist die klientenspezifische Sicht einzubeziehen.

Eine Bindung mit Dienstgüte-Bereitschaft kann aus Klientensicht potentiell unterschiedliche Dienstgüte-Vereinbarungen erlauben. Diese Dienstgüte-Vereinbarungen sind zum einen nicht von derselben Gestalt und nicht gleich präferiert. So kann die zugrundeliegenden Dienstgüte-Architektur nur eine Dienstgüte-Vereinbarung mit jeweils einer Dienstgüte-Charakteristik erlauben ($|\overline{Q}| = 1$) oder eine Kombination unterschiedlicher Dienstgüte-Charakteristiken mit jeweiligen Dienstgüte-Niveaus. Sind aus Klientensicht unterschiedliche Dienstgüte-Vereinbarungen möglich, kann ei-

ne Verhandlung dazu benutzt werden, die aus Klientensicht beste, realisierbare Dienstgüte-Vereinbarung zu erzielen.

3.5 Dienstgüte-Integration in Objektsysteme

Die Integration von Dienstgüte-Vorkehrungen für Objektsysteme, die auf einem verteilten Objektmodell beruhen, kann auf unterschiedliche Art und Weise erfolgen. In diesem Abschnitt sollen im Hinblick auf das Objektmodell Eigenschaften der Dienstgüte dargestellt und die Aufgaben konkretisiert werden.

Im wesentlichen sind die Aufgaben der Integration die Repräsentation einer Dienstgüte-Charakteristik und die Durchsetzung von Dienstgüte-Vereinbarungen durch Dienstgüte-Mechanismen. Dabei sind für die Dienstgüte-Charakteristiken geeignete Repräsentationen durch Dienstgüte-Parameter zu finden. Die Integration der Dienstgüte-Mechanismen muß die Ende-zu-Ende-Sicht berücksichtigen. Das bedeutet, daß sowohl auf der Klienten-, wie auch auf der Dienst-Seite korrespondierendes Verhalten integriert werden muß. Dieses Verhalten kann sich durch Dienstgüte-Mechanismen des Netzwerks – wie Aktive Netze [63] – auch auf Einheiten zwischen Klient und Dienst erstrecken, bspw. Router. Die Dienstgüte-Mechanismen sollten die Strukturierung von Dienstgüte-Mechanismen-Hierarchien erlauben. Dies ermöglicht die Wiederverwendbarkeit von Schichten aus der Dienstgüte-Mechanismen-Hierarchie und das Nutzen von Dienstgüte-Mechanismen auf der Netzwerk- oder Betriebssystem-Seite.

3.5.1 Varianten der Dienstgüte-Integration

Prinzipiell können zwei Vorgehensweisen der Dienstgüte-Integration unterschieden werden. Bei der impliziten Dienstgüte-Integration ist das Ziel eine für Anwendungsentwickler transparente Erbringung von Dienstgüte-Vereinbarungen, während bei der expliziten Dienstgüte-Integration Bewußtsein der Anwendung gegenüber der Dienstgüte-Erbringung gefordert wird.

Implizite Dienstgüte-Integration

Das vorrangige Ziel der impliziten Dienstgüte-Integration ist die Erhaltung der Verteilungstransparenz für Anwendungsobjekte. Die Konfigura-

tion, Überwachung und Steuerung der Dienstgüte-Erbringung geschieht bspw. durch den Systemadministrator. Essentiell bei der impliziten Dienstgüte-Integration ist die Wahl entsprechender Abstraktionen zur Integration der Dienstgüte-Vorkehrungen. Diese Abstraktionen werden von der zugrundeliegenden Verteilungsinfrastruktur bereitgestellt, damit darauf basierende Anwendungen keine besonderen Vorkehrungen zur Nutzung der Dienstgüte-Erbringung bereitstellen müssen.

Im Bereich verteilter Objektsysteme haben sich implizite Dienstgüte-Integrationen hauptsächlich im Bereich der Fehlertoleranz etabliert. An dieser Stelle seien kurz verschiedene Beispiele skizziert, um die prinzipielle Vorgehensweise zu illustrieren.

Der Object Group Service (OGS) [29] basiert auf einer Erweiterung von Standard-CORBA-Diensten. So wird der Event-Service als Basis für die Vermittlung von Nachrichten an eine Gruppe von Diensten gewählt. Die Gruppe wird über entsprechend ergänzte Dienste verwaltet. Klienten einer Gruppe kommunizieren durch Ereignisse. Die angebotene Dienstgüte-Charakteristik ist "Fehlertoleranz" gegenüber Dienst-Ausfällen durch Replikation. Die Konfiguration der Dienstgruppen erfolgt durch einen Verwaltungsdienst. Es existiert keine ausgezeichnete Abstraktion für Dienstgüte-Parameter oder -Charakteristiken. Klienten nehmen den Service über den Event-Service in Anspruch. Solange noch ein Dienst in der Gruppe existiert, steht der Service dem Klienten zur Verfügung.

Das Eternal-System [65] basiert wie OGS auf CORBA. Statt die Vorkehrungen für die Interaktion innerhalb einer Objektgruppe durch erweiterte Dienste zu realisieren, wird der ORB erweitert. Eine Nachricht an die Gruppe wird im ORB durch einen entsprechenden Mechanismus abgefangen und über ein Multicast-Protokoll verteilt. Fehlertoleranz – im Sinne der Maskierung von Dienst-Ausfällen – ist auch hier die unterstützte Dienstgüte-Charakteristik.

Maestro [99] wiederum geht einen gänzlich anderen Weg, der die Interoperabilität CORBAs durch das Internet-Inter-ORB-Protocol (IIOP) ausnutzt. Transparente Nutzung der Fehlertoleranz ist für Klienten möglich. Anwendungsobjekte, die einen Dienst realisieren, müssen bestimmte Verantwortlichkeiten gegenüber einem Gruppenprotokoll bereitstellen. Klienten benutzen ein Stub-Objekt, übergeben diesem den Aufruf, und eine IIOP-Nachricht wird vom Klienten-ORB über das Netzwerk geschickt. Ein Maestro-Dienst verteilt die Nachricht über ein proprietäres Protokoll an

die Dienste. Der Vorteil dieses Ansatzes ist die Transparenz auf der Klientenseite. Allerdings darf die Verbindung zwischen dem Klienten und dem Verteilerobjekt des Maestro-Dienstes nicht gestört sein, da diese Verbindung mit keinen Vorkehrungen gegen Ausfall geschützt ist.

Ken Birman, einer der Maestro-Autoren, kommt zu dem Schluß, daß die Nutzung von Multicast-Protokollen in Objektsystemen explizit behandelt werden sollte [14]. Damit ist die implizite Dienstgüte-Integration stark in Frage gestellt, da diese in verteilten Objektsystemen hauptsächlich für die Dienstgüte-Kategorie "Fehlertoleranz" basierend auf Gruppenkommunikation realisiert wurde. In anderen Anwendungsszenarien ist dies auch unmittelbar ersichtlich. Anwendungen mit Anforderungen der Dienstgüte-Kategorie "Multimedia" müssen auf Änderungen der Dienstgüte-Erbringung reagieren. Eine Anpassung an eine geringere Bandbreite kann beispielsweise durch Reduktion der Farben oder Größe einer Videoübertragung erfolgen. Eine solche Reaktion ist allerdings nur mit dem Bewußtsein der Anwendung über die Dienstgüte-Erbringung möglich.

Explizite Dienstgüte-Integration

Die explizite Dienstgüte-Integration stellt Schnittstellen für Anwendungsprogrammierer zur Verfügung, mit denen Objekte (Klienten, Dienste) Dienstgüte-befähigt werden können. Weiterhin muß die Anpassung von Anwendungen an sich ändernde Dienstgüte-Niveaus und die Steuerung der Dienstgüte-Erbringung unterstützt werden.

Dienstgüte-Repräsentation: Die Dienstgüte-Repräsentation muß die Dienstgüte-Charakteristiken im System durch Dienstgüte-Parameter abbilden. Dabei kann entweder eine feste Menge von Dienstgüte-Parametern vorgegeben sein oder aber eine Spezifikation für beliebige Dienstgüte-Parameter unterstützt werden. Die Zuordnung zwischen Dienstgüte-Charakteristiken und Klient/Dienst-Interaktionen ($[K, D, Q]$) muß in einer geeigneten Form angeboten werden. Zu Zwecken der Aushandlung wie auch der Überwachung ist es notwendig, Dienstgüte-Parameter zwischen Klient und Dienst und evtl. weiteren Einheiten auszutauschen. Daher sollte die Beschreibung der Dienstgüte-Parameter in einer Form erfolgen, welche die zugrundeliegende Funktionalität der Verteilungsinfrastruktur nutzt.

Dienstgüte-Mechanismen-Integration: Dienstgüte-Mechanismen müssen in das verteilte Objektmodell der Verteilungsinfrastruktur integriert werden. Bei dieser Integration der Dienstgüte-Mechanismen sind zwei Schnittstellen zu berücksichtigen:

- **Dienstgüte-Mechanismen und Anwendungsobjekte:** Bei der Interaktion zwischen Klient und Dienst müssen Dienstgüte-Mechanismen das vereinbarte Dienstgüte-Niveau sicherstellen. An bestimmten Stellen des Kommunikationspfades sind Schnittstellen für Dienstgüte-Mechanismen vorzusehen. Weiterhin ist für die Konfiguration der Dienstgüte-Mechanismen eine Schnittstelle für die Anwendungsobjekte bereitzustellen. Darüber hinaus muß die Zuordnung einer Klienten/Dienst-Interaktion mit einer bestimmten Dienstgüte-Vereinbarung zu korrespondierenden Dienstgüte-Mechanismen ermöglicht werden.

- **Dienstgüte-Mechanismen und Plattform:** Die dem ORB und den Anwendungsobjekten zugrundeliegende Plattform konstituiert sich typischerweise aus dem Betriebssystem und dem Netzwerk. Dienstgüte-Mechanismen dieser Schicht sollten von den Dienstgüte-Mechanismen, wie sie in die Verteilungsplattform integriert werden, nutzbar sein.

Die Integration von Dienstgüte-Mechanismen erfordert unterschiedliche Vorkehrungen, abhängig von der Ebene der Integration (Anwendung, Netzwerk). Die plattformspezifische Integration von Dienstgüte-Mechanismen ist abhängig von der Architektur der zugrundeliegenden Verteilungsinfrastruktur. Im Gegensatz dazu ist bei der Integration von Dienstgüte-Mechanismen in die Anwendungsobjekte auf eine saubere Trennung nach den Verantwortlichkeiten (Service/Dienstgüte-bezogen) zu achten.

Ein denkbarer Ansatz besteht darin, Dienstgüte-Charakteristiken als Objekte abzubilden. Die Dienstgüte-Parameter stellen den Objektzustand dar, und die Dienstgüte-Mechanismen werden durch das Objektverhalten realisiert. Unabhängig von der Form der Einbettung durch Delegation oder Vererbung treten hierbei aber Anomalien auf, die eine gesonderte, softwaretechnische Betrachtung erforderlich machen. Die enge Kopplung zwischen Anwendungsverhalten und den Dienstgüte-Vorkehrungen macht eine Trennung der Verantwortlichkeiten schwer. Daher ist eine Untersuchung über

geeignete Ansätze zur Strukturierung dieser Verantwortlichkeiten notwendig. Im folgenden Unterabschnitt werden Entwurfsmuster zur Integration von Dienstgüte-Vorkehrungen in verteilte Objektsysteme betrachtet. Anschließend wird die aspektorientierte Programmierung als ein weiterer Ansatz vorgestellt.

3.5.2 Entwurfsmuster

Entwurfsmuster [35] (engl. *Design Patterns*) stellen generische Lösungen für wiederkehrende Probleme dar. Durch eine genaue Beschreibung des Problems, Hinweise zur Anpassung und eine allgemeine Lösungsvorlage dienen Entwurfsmuster einer funktionellen, problemorientierten Gruppierung von Einheiten. Dies ermöglicht es, in einem Softwareentwurf gröbere Einheiten als Klassen und feinere Einheiten als Module zu modellieren. Das Objektmodell bietet aufgrund seiner Eigenschaft der Hierarchie-Bildung eine wesentliche Eigenschaft, allgemeine Eigenschaften zu abstrahieren und durch Vererbung spezialisierte hinzuzufügen.

Eigenschaften von Kommunikationssystemen sind schon durch entsprechende Entwurfsmuster modelliert worden. ACE [84] stellt eine Kommunikationsbibliothek dar, die durch Entwurfsmuster für aktive und passive Kommunikationspartner die Modellierung und Implementierung von verteilten Objektsystemen erleichtert. Die ComServ [3] ist eine auf Entwurfsmustern basierende Kommunikationsbibliothek. In der ComServ werden hauptsächlich in [35] beschriebene Entwurfsmuster eingesetzt, um unter derselben Schnittstelle eines Dienstes unterschiedliche Implementierungen anzubieten. Die angebotenen Transportdienste sind recht einfacher Natur. Unter einfachen Dienstprimitiven für die Kommunikation können verschiedene Mechanismen zur Steuerung der Übertragung zum Einsatz kommen. Diese Mechanismen erlauben eine einfache Form von Dienstgüte-Erbringung durch unterschiedliche Flußkontrolle (best-effort, stop-and-wait- und Schiebefenster-Synchronisation). Eine auf CORBA basierende Dienstgüte-Architektur für Verfügbarkeit und Zuverlässigkeit stellt Elektra [61] dar. Basierend auf netzwerkseitig implementierter Gruppenkommunikation werden Anwendungsobjekte mit Replikationsmechanismen versehen. Dieser Lösungsansatz bildet ein Entwurfsmuster [62].

Ein Beispiel für den Einsatz spezieller Entwurfsmuster und Dienstgüte ist TAO (The ACE ORB) [85], das unter Verwendung einer auf Ent-

wurfsmustern basierenden Bibliothek entworfen und realisiert wurde [84], [86]. Die dabei entworfenen Entwurfsmuster finden sich mittlerweile auch im Open Communications Interface (OCI) [76] und der CORBA Realtime Spezifikation [74] wieder.

Die Partitionierung einer Modellierung mittels Entwurfsmustern führt zum besseren Verständnis komplexer Strukturen, da Zusammenhänge zwischen verschiedenen Objekten durch die Funktionalität der Entwurfsmuster erklärt werden. Auch weniger erfahrene Programmierer erhalten durch Entwurfsmuster Anleitungen für stabile, zielorientierte Entwürfe. Dies alles bedingt aber einen Katalog von Entwurfsmustern, der von allen Beteiligten benutzt wird. Da Entwurfsmuster keine Widerspiegelung in Sprachelementen haben, ist ihre Dokumentation begleitend und in dem Quelltext zu leisten.

3.6 Dienstgüte und aspektorientierte Programmierung

3.6.1 Die aspektorientierte Programmierung

Größere Softwaresysteme lassen sich schwer strukturieren. Abhängig von den Strukturierungsmitteln einer Methode oder Programmiersprache wird ein größeres Problem in kleinere, überschaubare Einheiten zerlegt. Diese Einheiten sind besser zu verstehen, zu testen und zu warten, wenn sie ohne Abhängigkeiten zu anderen Komponenten formuliert sind.

Das in Kapitel 2 eingeführte Objektmodell bietet durch die Einkapselung, Modularität und Abstraktion die wesentlichen Mechanismen, um ein Problem in abgeschlossene Einheiten – in diesem Fall Objekte – zu zerlegen.

Ein aber häufig auftretender Effekt sind Vermaschungen unterschiedlicher Aspekte, die verschiedene Einheiten tangieren. Liegen solche Vermaschungen vor, ist eine isolierte Wiederverwendung einer einzelnen Einheit nicht mehr möglich. Das folgende Beispiel skizziert die Vermaschung verschiedener Einheiten.

```
1:  class Queue{
2:    ...
3:    void insert(Element e){
4:      if (head == max)
```

```
5:         P(x); // das nächste remove befreit den Semaphor
6:       content[head++] = e;     // Element einfügen
7:       V(y);    // Semaphor für remove-Blockade
8:     }          // bei leerer Queue
9:     Element remove(){
10:      if (head == 0)
11:        P(y); // warten auf insert
12:      V(x);    // evtl. wartendes insert befreien
13:      return content[head--];  // Element zurückgeben
14:    }
15: };
16:
17: int main(){
18:   int s;
19:   Queue *localQueue = new Queue;
20:   s = open(...); connect(s,..);
21:   while (1){
22:     read(s,buf, MSG_SIZE);
23:     if (!strcmp(buf,"insert")){
24:       // unmarshal parameters
25:       localQueue->insert(e);
26:     }
27:   ...
28:   }
29: }
```

In dem Beispiel ist ein Dienst skizziert, der einen Service zum Speichern von Daten (`Element`) in einer begrenzten Queue anbietet. Der Dienst ist über das Netzwerk von Klienten erreichbar. In den Zeilen 1-15 ist die Klasse des Dienstes dargestellt. Zwei Methoden zum Einfügen und Entfernen von Elementen sind in ihrer Implementierung zu sehen. Dabei werden Klienten-Zugriffe synchronisiert. Zeilen 5, 7, 11 und 12 zeigen die Synchronisation. Beim Einfügen in eine volle Queue wird der Auftrag blockiert, bis ein Entfernen aufgerufen wurde. Analog wird beim Entfernen aus einer leeren Queue gewartet, bis ein Element eingefügt wurde. Das Hauptprogramm des Dienstes (Zeilen 17-29) öffnet eine Netzwerkverbindung durch einen Socket

und wartet auf eintreffende Nachrichten von Klienten. Die Nachrichten werden nach ihrem Auftrag unterschieden und die entsprechende Methode auf dem Dienst aufgerufen. Dazu notwendig ist die Konvertierung von Parametern und Wandlung in lokale Daten.

In dem Beispiel finden sich drei Aspekte:

- **Dienst:** Der erste Aspekt ist die Implementierung des Services. Dieser ist durch die Methoden `insert` und `remove` gegeben und besteht aus dem Speichern bzw. Löschen eines Elements in der internen Datenstruktur (Zeilen 6, 13).

- **Kommunikation:** Die Kommunikation mit dem Klienten ist aus der Klasse des Dienstes in das Hauptprogramm verlagert worden. Insofern wurde hier schon eine Trennung der Aspekte bewerkstelligt. Allerdings ist die Kommunikation stark an den Dienst angepaßt, wie die Auswahl der Methode durch einen Stringvergleich (Zeile 23) zeigt.

- **Sychronisation:** Die im Beispiel realisierte Synchronisation zwischen den Aufrufen von `insert` und `remove` sichert den Klienten zu, daß die Aufrufe zum frühestmöglichen Zeitpunkt durchgeführt werden und kein Fehler – bspw. durch Einfügen in eine volle Queue – auftritt oder gesondert behandelt werden muß.

Zu diesen Aspekten kommt noch die Klienten-Seite hinzu. Klienten müssen, um den Service in Anspruch nehmen zu können, eine Verbindung zu dem Dienst aufbauen und Anfragen in dem gewünschten Format stellen. In dem Beispiel ist zu sehen, daß die einzelnen Aspekte losgelöst schwer wiederverwendbar sind, da sie syntaktisch miteinander verwoben sind.

Solche Vermaschungen zwischen verschiedenen Aspekten einer Anwendung treten unabhängig von den gewählten Abstraktionen zur Einkapselung auf. Dabei können prozedurale Abstraktion durch Module und Funktionen, objektorientierte Abstraktionen durch Einkapselung von Verhalten und Struktur oder nur Datenabstraktion zum Einsatz kommen. Die aspektorientierte Programmierung (AOP) [50] addressiert solche Effekte konzeptionell und konstruktiv.

Dazu wird in der AOP zwischen Aspekten und Komponenten unterschieden. Die Grundlage der folgenden Definitionen ist eine Mehrzweck-Programmiersprache, also eine, die nicht für einen bestimmten Anwendungszweck maßgeschneidert ist. Dabei spielt die von der Programmiersprache

zur Verfügung gestellte Abstraktion (prozedural, objektorientiert) keine Rolle.

- **Komponenten** können klar durch vorhandene Abstraktionen (Objekte, Methoden, Prozeduren) eingekapselt werden. Klar eingekapselt heißt, daß die Komponente definiert lokalisiert werden kann, einfach zugreifbar ist und mit anderen Komponenten kombiniert werden kann.

- **Aspekte** lassen sich nicht durch die vorhandenen Abstraktionen einkapseln. Sie ziehen sich somit durch mehrere eingekapselte Einheiten. Die AOP spricht dabei vom *tangling of aspects*. Typische Aspekte hängen nicht mit der funktionalen Dekomposition eines Systems zusammen, sondern spiegeln die nicht-funktionalen Eigenschaften wider. Diese können sein: Fehlerbehandlung, Leistungseigenschaften, Synchronisation etc.

Neben der Klassifikation von Aspekten versucht die AOP, Lösungsansätze für die Integration von Aspekten und Komponenten zu geben.

3.6.2 Lösungsansatz der AOP

Die Grundidee der AOP besteht darin, zwischen spezialisierten Sprachen für die Beschreibung der Aspekte und dem aus all diesen Aspekten resultierenden Programm zu trennen. Da die Aspekte unterschiedliche Eigenschaften eines Systems modellieren, ist es sinnvoll, diese in dafür geeigneten Sprachen zu formulieren. Daher führt die AOP den Begriff der *Aspektsprache* ein. Eine Aspektsprache drückt einen Teil der Gesamteigenschaften des Systems aus. Die Summe aller Eigenschaften eines Systems wird durch alle Aspekte ausgedrückt. Diese Aspekte können in verschiedenen Aspektsprachen formuliert sein.

Im folgenden sind die Aspekte aus dem eingangs diskutierten Beispiel in jeweils eigenen Aspekt-Sprachen formuliert.

```
// Service-Aspekt
class Queue {
  ...
  void insert(Element e) {
```

```
    if (head < max)
       content[head++] = e;
  }
  ...
  Element remove() {...}
};
```

Der Dienst-Aspekt wird in seine verschiedenen Aspekte aufgegliedert, und in einer eigenen Aspektsprache implementiert der Dienst den Service. Kommunikations- und Synchronisations-Aspekte werden getrennt formuliert. Im Gegensatz zu dem Beispiel ist hier nun der Klienten-Aspekt abgebildet. Dieser greift auf den Dienst zu, als ob er lokal erreichbar wäre:

```
// Klient-Aspekt
Queue *aQueue = new Queue;
Element e;
  aQueue->insert(e);
  e = aQueue->remove();
```

Der Kommunikations-Aspekt modelliert die Methoden des Dienstes, die von Klienten als Service über ein Netzwerk in Anspruch genommen werden können. Dies entspricht einer Schnittstellenbeschreibungssprache:

```
// Kommunikations-Aspekt
interface Queue {
  void insert (REF Element);
  Element remove();
};
```

Der Synchronisations-Aspekt wird wiederum in einer eigenen Sprache beschrieben. Dabei sind die beiden Synchronisationen zwischen den Einfüge- und Entfernen-Methoden jeweils getrennt modelliert und einem Semaphor zugeordnet:

```
// Synchronisations-Aspekt
sync s {
  MUTEX x {
    void Queue::insert(Element)
```

```
      REQUIRES { head<max } OR WAIT;
    void Queue::remove()
      ENABLES (Queue::insert);
  };
  MUTEX y {
    void Queue::insert(Element)
      ENABLES (Queue::remove);
    void Queue::remove()
      REQUIRES { head > 0 } OR WAIT;
  };
};
```

Ein AspectWeaver führt alle Aspekte eines Programms in eine Zielsprache bzw. eine ausführbare Datei zusammen. Abbildung 3.3 illustriert das Zusammenführen der Aspekte. Aus den einzelnen Aspektsprachen wird die Gesamtfunktionalität ohne weitere Anpassung des Programmierers generiert. In der Abbildung sind die generierten Fragmente der Klienten- und der Dienstseite zu sehen. Der Kommunikations-Aspekt ist nicht explizit dargestellt, sondern durch entsprechende Ableitung von Basisklassen angedeutet. Diese Basisklassen können, wie in Kapitel 2 beschrieben, eine Funktionalität wie das Proxy- bzw. Skeleton-Objekt des verteilten Objektmodells realisieren. *D* [58] stellt ein System dar, das die im Beispiel genannten Aspekte in ein Java-Programm zusammenführt.

Zusammengefasst besteht die Grundidee der aspektorientierten Programmierung darin, daß man für eine größere Anzahl von Problemen geeignete Aspektsprachen identifizieren und diese automatisch durch den AspectWeaver zusammenführen kann. Daher bildet AOP einen Lösungsansatz und kein universelles Paradigma. Die Wahl der Aspektsprachen und der Zielsprache erfordert große Sorgfalt, damit diese Lösung für andere Probleme wiederverwendbar ist. Die Steuerung des AspectWeavers ist von den Aspektsprachen und deren Verantwortlichkeiten abhängig, so daß auch hier keine Universalität zu erwarten ist.

Unbeschadet dessen stellt die AOP einen wichtigen Klassifizierungsansatz dar. Sind bestimmte Aufgaben einer Anwendung als Aspekte identifiziert, so kann durch geeignete Maßnahmen versucht werden, der fehlenden Einkapselung und der Vermaschung mit anderen Verantwortlichkeiten zu begegnen. Im folgenden Unterabschnitt wird die Integration von Dienst-

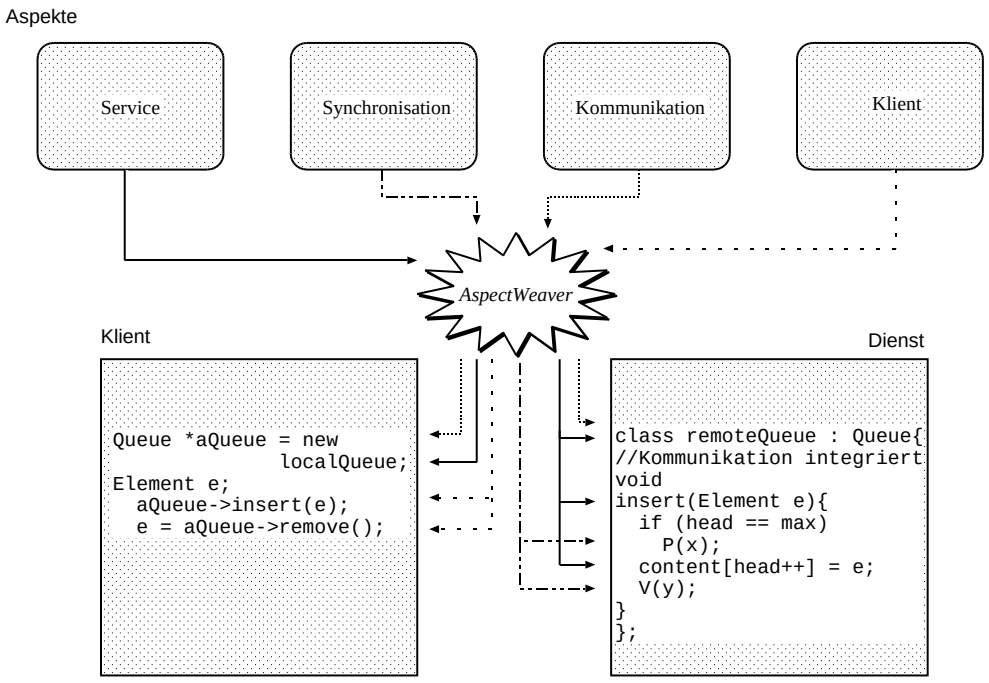


Abbildung 3.3: Aspektsprachen und Weaving

güte-Vorkehrungen in objektorientierte Anwendungen als ein Aspekt im Sinne der AOP klassifiziert.

3.6.3 Dienstgüte als Aspekt im Sinne der AOP

Abbildung 3.4 illustriert die Kommunikation eines Klienten mit einer Replikatgruppe. Diese Replikatgruppe dient zur Erhöhung der Verfügbarkeit des Service. Solange einer der Dienste in der Replikatgruppe existiert, wird ein Ergebnis an den Klienten geliefert. Somit erlaubt diese Konstruktion bei einer Anzahl von n Replikaten die Maskierung von $n-1$ Dienst-Ausfällen.

Es ist zu beachten, daß die Implementierung solcher Gruppen entsprechende Vorkehrungen auf der Transport-Ebene notwendig macht. Der Zustand der Replikate D muß für alle Replikate gleich sein, damit die von allen auf Anfragen gelieferte Antwort gleich ist. Alle Anfragen von Klienten müssen in derselben Reihenfolge an alle Replikate zugestellt werden. Notwendig hierzu ist Gruppenkommunikation mit geeigneten Protokollen für die Gruppenzugehörigkeit und die Nachrichtenweiterleitung. Weiterhin muß jedes Replikat nach Verarbeiten einer Anfrage in demselben Zustand

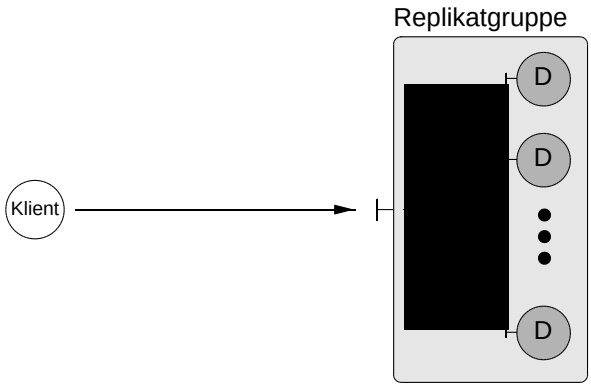


Abbildung 3.4: Verfügbarkeit durch Gruppenkommunikation

sein (Determinismus der Replikate). ISIS [15] und HORUS [97] stellen Vertreter solcher Infrastrukturen dar. Das Protokoll für die Gruppenzugehörigkeit stellt sicher, daß neu gestartete Replikate denselben Zustand erhalten, wie die in der Gruppe befindlichen Replikate. Das Protokoll für die Nachrichtenweiterleitung ist für die Auslieferung aller Klientenanfragen an die Replikate in derselben Reihenfolge verantwortlich. Ein solcher nicht-zentralisierter Ansatz wird als als State-Machine-Ansatz [87] bezeichnet.

Abbildung 3.5 zeigt eine mögliche Realisierung einer Gruppenorganisation. Diese Realisierung dient nur der Motivation der auftretenden Probleme und ist stark vereinfacht. Beispielsweise lassen sich so Zugriffe mehrerer Klienten auf dieselbe Gruppe nicht ohne weitere Vorkehrungen lösen. Ein ausgezeichneter Dienst ist mit dem Klienten kollokiert. Dieser stellt einen *Sequencer* dar, dessen Verantwortlichkeit in der Verwaltung der Gruppe und der Weiterleitung der Aufrufe liegt. Damit der Klient nicht durch die Dienstimplementierung belastet wird, realisiert der Sequencer nicht den eigentlichen Dienst, sondern stellt nur eine transparente Einkapselung der Replikatgruppe dar. In dem Modell eines verteilten Methodenaufrufs (vgl. Abbildung 2.3) käme dies einem erweiterten Proxy-Objekt gleich. Somit ist es möglich, für den Klienten transparent eine Einkapselung der Dienstgüte Verfügbarkeit zu realisieren.

Die Implementierung der einzelnen Dienste aber stellt ein Problem dar. Wird ein Dienst D im Sinne des Objektmodells als Objekt betrachtet, das Struktur und Verhalten im Sinne einer Teilverantwortlichkeit eines Systems einkapselt, ist die Erweiterung um Dienstgüte schwer ohne einen Bruch dieser Modellierung zu bewerkstelligen. Eine auf Replikatgruppen basierende

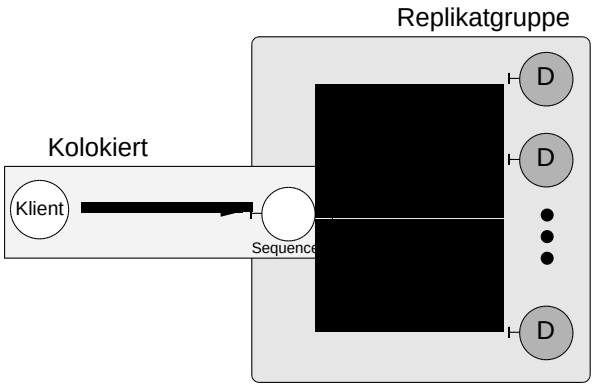


Abbildung 3.5: Transparenz von Verfügbarkeit

Dienstgüte-Realisierung von Verfügbarkeit muß Zugriff auf den Zustand des Dienstes haben, der durch die Schnittstelle des Objektes eingekapselt ist. Drei mögliche Realisierungsansätze sind:

1. *Erweiterung der Dienstimplementierung:* Die Dienstgüte-Realisierung wird explizit im Dienst integriert. Als Folge ist der Dienst vermascht mit nicht-funktionalem Verhalten, was ein Bruch der Modellierung nach Verantwortlichkeit ist. Zusätzlich finden sich verschiedene Abstraktionen in der Dienstimplementierung: die Realisierung des funktionalen Verhaltens, das durch Infrastruktur von der Kommunikation getrennt wird, und kommunikationsorientiertes Verhalten für die Gruppenkommunikation. Dadurch werden beide Aspekte schwerer einzeln wiederverwendbar.

   Abbildung 3.6 illustriert einen möglichen Ansatz, die beiden Aspekte "Verfügbarkeit" und einen einfachen Dienst – hier als Namensdienst skizziert – zusammenzuführen. In der Dienstgüte-Klasse wird die Kommunikation innerhalb der Replikatgruppe und deren Konfiguration implementiert. Die Methoden zum Transfer des Zustands (`get_state` und `put_state`) sind allerdings abhängig vom Dienst und können nicht von der Dienstgüte-Implementierung bereitgestellt werden – diese bleiben abstrakt. Der Dienst wird in der Dienst-Klasse implementiert. Durch Vererbung werden beide Klassen in eine Dienstklasse mit Dienstgüte-Befähigung zusammengeführt. Diese Klasse erbt das Verhalten sowohl der Dienstgüte als auch des Dienstes. Lediglich die Anpassung an die Dienstgüte ist hier zu implementieren.

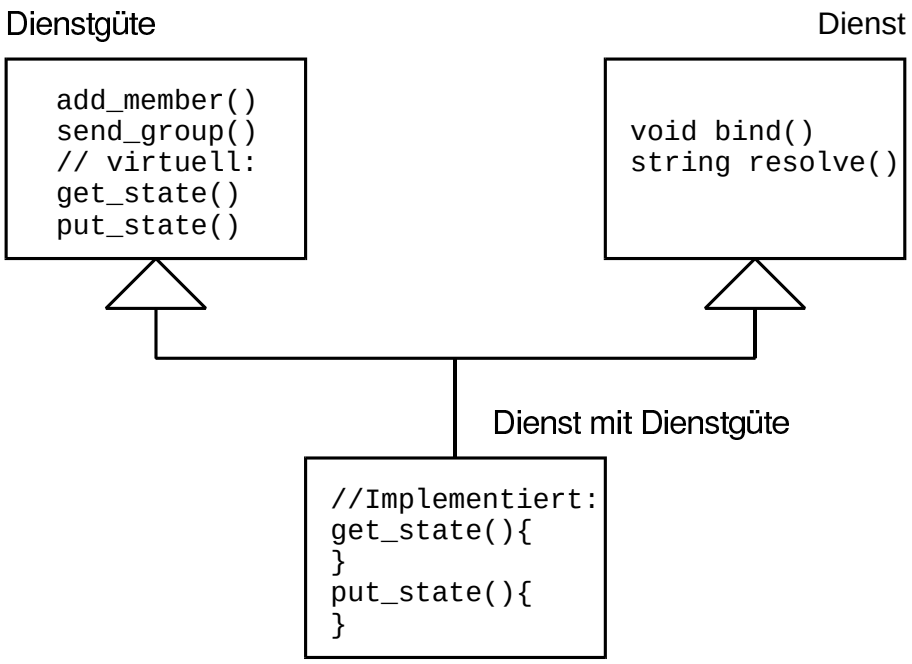


Abbildung 3.6: Trennung der Aspekte durch Vererbung

Dies entspricht einem Adapter-Entwurfsmuster [35]. Voraussetzung ist allerdings eine geeignete Implementierung der Gruppenkommunikation. Entweder wird die Transparenz des Zugriffes auf die Gruppe vom Klienten aufgegeben, und der Klient benutzt eine Methode der Dienstgüte-Implementierung (`send_group`), um die Nachrichten an die Gruppe zu senden, oder die Struktur der Dienst/Dienstgüte-Zusammenführung benötigt komplexere Strukturen wie ein Bridge-Entwurfsmuster.

2. *Implizite Trennung der Dienstgüte- und der Dienst-Realisierung:* Durch systemabhängige Lösungen, wie Präprozessoren, und Spracheigenschaften, wie Serialisierung des Objektzustands, wird das Dienstgüte-Management von der Dienstimplementierung getrennt. Der Zugriff auf den Zustand des Dienstes erfolgt hier durch Zwischenschichten, die in die Kommunikation zwischen Dienst und Klient integriert werden.

   Abbildung 3.7 zeigt eine einfache Architektur, die durch eine Multicast-Transportschicht unterhalb des ORBs Gruppenkommunikation

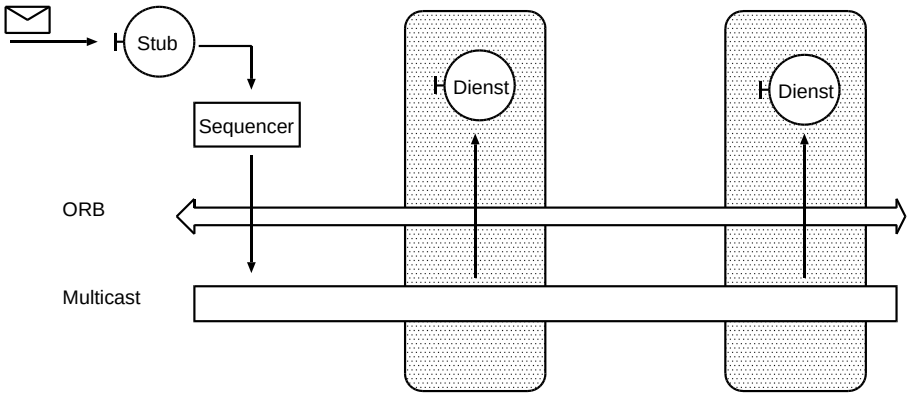


Abbildung 3.7: Implizite Trennung der Aspekte

erlaubt. Das Stub-Objekt auf der Klientenseite übergibt einen Auftrag an den Sequencer, der einen Multicast auf der Gruppe initiiert. Die Nachricht wird dann an alle Dienste der Gruppe zugestellt. Der Multicast-Transport hat nun die Möglichkeit, alle in der Gruppe verschickten Nachrichten zu protokollieren und so neu gestartete Replikate in den aktuellen Zustand zu bringen, bevor sie in die Gruppe integriert werden. Die Konfiguration des Sequencers kann von einem Administrator vorgenommen werden.

3. *Explizite Trennung der Dienstgüte und der Dienst-Realisierung:* Durch bestimmte Sprachmittel, wie Aspektsprachen oder dokumentierte Entwurfsmuster, wird die Dienstgüte-Realisierung und der Dienst durch wohldefinierte Schnittstellen gebunden, die Implementierung aber getrennt. Das in Abschnitt 3.6.2 vorgestellte Beispiel hat gezeigt, wie durch entsprechende Aspektsprachen und nachgeschaltete Verwebung durch den AspektWeaver komplexe Aspekte zusammengeführt werden können. Wenn anstelle des Sychronisations-Aspektes ein Dienstgüte-Aspekt dem Weaver zugeführt würde, könnte dieser auch eine Vermaschung ähnlich der Struktur in Abbildung 3.6 generieren. Notwendig dazu ist allerdings eine Aspektsprache, die eine Spezifikation der Dienstabhängigkeiten gegenüber der Dienstgüte-Implementierung enthält.

Alle drei Ansätze begegnen demselben Phänomen: die Dienstgüte-Charakteristik Verfügbarkeit ist ein Aspekt im Sinne der AOP, der verschiedene

Einheiten im System miteinander in Beziehung setzt. Ein aspektorientierter Ansatz für die Integration von Dienstgüte-Management in verteilte Objektsysteme ist in der Lage, komplexe Aspekte zu integrieren und somit für generische Dienstgüte-Integration geeignet. Allerdings ist bei der Gestaltung der Aspektsprachen auf eine hohe Allgemeinheit zu achten, damit der initial hohe Aufwand für die Erstellung der Aspektsprachen und des AspectWeavers von vielen Dienstgüte-Implementierungen genutzt werden kann.

Weitere Beispiele solcher Vermaschungen im Kontext des Dienstgüte-Managements sind die Verkürzung der mittleren Antwortzeit durch Lastbalancierung oder die Anwendung von bestimmten Algorithmen bei numerischerer Berechnungen. Bei der Lastbalancierung ist die Lastinformation abhängig von dem realisierten Dienst. Zwar kann auch die Auslastung der CPU in Betracht gezogen werden, aber bei sequentieller Bearbeitung von Aufträgen kann die Länge der Auftragswarteschlange ein weiteres, dienstabhängiges Kriterium sein. Die Realisierung eines Berechnungsdienstes mit verschiedenen Genauigkeiten oder Optimierungen hat Auswirkungen auf die Antwortzeit und die Qualität der Ergebnisse. Dabei ist der Aspekt der Berechnung und der Aspekt der Optimierung miteinander vermascht [50].

Beispiele für aspektorientierte Ansätze sind *D* [58], ein Rahmenwerk für Verteilungsaspekte, und Aspectix [41], das Dienstgüte als Objektfragmente unterstützt. In [81], [8], [22] findet der interessierte Leser weitere Diskussionen über Aspekte in verteilten Systemen.

3.6.4 Bewertung

In diesem Abschnitt wurden verschiedene Integrationsansätze für die explizite Integration von Dienstgüte in Objektsysteme vorgestellt. Der Schwerpunkt lag auf der Integration von Dienst und Klient als Objekte im Objektmodell; Dienstgüte-Charakteristiken wurden als Zusammenschluß der Schnittstelle der Dienstgüte-Mechanismen und der Dienstgüte-Parameter behandelt.

Dabei sind die beiden hier vorgestellten Integrationsansätze – die aspektorientierte Programmierung und Entwurfsmuster – nicht als Ausschluß zu verstehen. Die Umsetzung von Aspekten aus unterschiedlichen Aspektsprachen in eine gemeinsame Zielsprache basiert häufig auch auf Entwurfsmustern. Diese entkoppeln die entsprechenden Implementierungen. Zusam-

menfassend zeigt die Klassifikation der Repräsentierung von Dienstgüte-Charakteristiken in Objektsystemen als Aspekt im Sinne der AOP, daß eine klare Trennung der Verantwortlichkeiten von Dienstgüte-Vorkehrungen und Anwendung nicht ohne besondere Vorkehrungen etabliert werden kann.

3.7 Dienstgüte-Management

In diesem Abschnitt wird auf die Aufgaben von Dienstgüte-Architekturen zum Dienstgüte-Management eingegangen werden. Neben den Aufgaben werden die Anforderungen an das Dienstgüte-Management in verteilten Objektsystemen motiviert. Abschließend findet sich eine Diskussion des Stands der Forschung.

3.7.1 Aufgaben des Dienstgüte-Managements

Die vornehmliche Aufgabe des Dienstgüte-Managements ist die Bestimmung und Durchsetzung von Dienstgüte-Vereinbarungen zwischen Klienten und Diensten. Dabei ist die Bestimmung und Durchsetzung von Dienstgüte-Vereinbarungen abhängig von den zugrundeliegenden Eigenschaften der Dienstgüte-Architektur. Diese sind, wie eingangs des Kapitels erwähnt, Dynamik, Mehrkategorie und Generizität. Abhängig dieser drei Dimensionen finden sich unterschiedliche Ansätze, wie Dienstgüte-Management adressiert werden kann. Die Integration in eine Verteilungsinfrastruktur ist für das Dienstgüte-Management unerläßlich, da die Durchsetzung von Dienstgüte-Vereinbarungen nicht nur die nicht-funktionalen Aspekte der Dienserbringung betrachtet, sondern eng mit den funktionalen Aspekten verknüpft ist.

Die Hauptaufgaben des Dienstgüte-Managements sind:

- **Verhandlung und Anpassung:** In verteilten Systemen herrscht Konkurrenz zwischen verschiedenen Klienten/Dienst-Interaktionen um Ressourcen. Liegt keine statische Zuordnung von Ressourcen zu Klienten/Dienst-Beziehungen vor, muß vor einer Interaktion überprüft werden, ob das System die Anforderungen des Klienten erfüllen kann. Dies geschieht durch eine Verhandlung. Ändern sich die Ressourcen-Gegebenheiten während der Interaktion, bspw. durch Res-

sourcen-Entzug, kann durch eine Nachverhandlung eine Anpassung der Interaktion auf ein anderes Dienstgüte-Niveau erfolgen. Dienstgüte-Architekturen, die eine Anpassung an gegebene Ressourcen-Verhältnisse durch Verhandlung und die Sicherstellung von Ressourcen-Verfügbarkeit durch Reservierung erlauben, werden als hybride Dienstgüte-Architekturen bezeichnet [30].

- **Infrastrukturdienste:** Die Unterstützung der Interaktion zwischen Klient und Dienst mit Dienstgüte-Befähigung erfordert z. T. Erweiterungen an bestehenden Infrastrukturdiensten der Verteilungsplattform oder die Bereitstellung von zusätzlichen Diensten. Abhängig von der Auslegung können diese Dienste zusammenfallen oder aber entfallen. Beispielsweise kann die Überwachung an einen Dienst zur Ressourcen-Steuerung bzw. -Zuteilung gebunden und nicht als eigenständiger Dienst verfügbar sein. Der Dienst für die Ressourcen-Steuerung sollte wiederum vor den Anwendungsobjekten verborgen bleiben und nur für die Dienstgüte-Implementierung von Belang sein. Die folgenden Dienste sind aus Sicht der Anwendungsobjekte sinnvoll, um Dienstgüte-befähigte Interaktion zu erlauben. Es sei darauf hingewiesen, daß diese Dienste aber nicht zwingend erforderlich sind. So kann ein System die Vermittlung von Diensten an Klienten durch einen Namensdienst durchführen und nicht auf einen Trader zurückgreifen.

  - *Vermittlung:* Neben den üblichen Dienstvermittlern, die auf der Beschreibung des funktionalen Dienstverhaltens basieren, sind für die Vermittlung Dienstgüte-befähigter Dienste auch Informationen über deren unterstützte Dienstgüte-Charakteristiken nötig.
  - *Überwachung:* Ist zwischen einem Dienst und einem Klienten eine Dienstgüte-Vereinbarung erzielt, ist diese während der Interaktion zu überwachen. Das frühzeitige Erkennen von Verletzungen des vereinbarten Dienstgüte-Niveaus kann so zu einer Anpassung der Anwendung oder Zuteilung zusätzlicher Ressourcen führen.
  - *Abrechnung:* Die Verknüpfung einer bestimmten Dienstqualität mit Kosten erlaubt eine bessere Differenzierung als Pauschalpreise. So zahlen Klienten für die tatsächlich erbrachte Qualität.

Dies kommt darüber hinaus dem Gesamtsystem zugute, da Klienten mit geringerer Wahrscheinlichkeit Ressourcen anfordern, die zwar mit Mehrkosten verbunden sind, aber eine nicht benötigte Qualität ermöglichen.

Im folgenden sollen die Anforderungen an generisches Dienstgüte-Management in verteilten Objektsystemen dargestellt werden.

3.7.2 Dienstgüte-Management für verteilte Objektsysteme

Die in Kapitel 2 aufgezeigten Probleme offener verteilter Systeme werden durch unterschiedliche Vorgehensweisen (vgl. Abschnitt 3.5) adressiert. Dieser Abschnitt stellt Anforderungen an das Dienstgüte-Management in verteilten Objektsystemen vor und motiviert diese.

Architekturelle Anforderungen der Dienstgüte-Integration

Bislang sind die Anforderungen an das Dienstgüte-Management weitgehend isoliert von der zugrundeliegenden Verteilungsinfrastruktur dargestellt und diskutiert worden. Auch die in Abschnitt 3.5 betrachteten Zusammenhänge zwischen Objektsystemen und Dienstgüte-Einbettung wurden auf einer konzeptionellen Ebene erläutert. Da die Dienstgüte-Erbringung aber immer auf der Basis einer funktionalen Diensterbringung stattfindet, soll an dieser Stelle auf die Anforderungen der Dienstgüte-Integration in eine Verteilungsinfrastruktur eingegangen werden.

Als Basisarchitektur soll hier das in Kapitel 2 vorgestellte verteilte Objektmodell der Object Management Architecture (OMA) dienen. Auch andere verteilte Objektmodelle, wie das Distributed Component Object Model (DCOM), weisen eine ähnliche Struktur auf [20]. Damit lassen sich die in dieser Arbeit gewonnenen Erkenntnisse auch in anderen verteilten Objektmodellen anwenden. Die Dienstgüte-Integration sollte die Abstraktionen der Basisinfrastruktur erhalten und konform zu diesen eingebettet werden. Anwendungsentwickler erfahren so keinen Bruch in den gewohnten Abstraktionen. Desweiteren wird so die Überführung bestehender Dienste und Klienten zu Dienstgüte-befähigten erleichtert.

Im folgenden werden die aus der Sicht des Anwenders und der objektorientierten Basisinfrastruktur notwendigen Anforderungen an die Dienstgüte-Integration betrachtet.

Trennung der Verantwortlichkeiten: Ein wesentliches Prinzip des objektorientierten Entwurfs ist die Trennung der Verantwortlichkeiten (Separation of Concern). Dabei werden Einheiten (Klassen) isoliert, die eine bestimmte Verantwortlichkeit erfüllen. Diese Partitionierung eines Problems erfaßt allerdings nur die funktionale Sicht auf ein Objektsystem, also die Diensterbringung. Bei der Erweiterung um Dienstgüte-Befähigung sollte diese Trennung der Verantwortlichkeiten erhalten bleiben: Dienste, Klienten und die Dienstgüte-Mechanismen sollten soweit getrennt werden, daß diese möglichst eigenständige Einheiten sind.

Wird eine solche Trennung unterstützt, werden Dienste und Klienten getrennt von der Dienstgüte-Befähigung wiederverwendbar gehalten. Dies dient auch der späteren Integration von Dienstgüte-Befähigung von bereits existierenden Diensten und Klienten. Gerade der systemnahe Charakter von Dienstgüte-Mechanismen macht diese Trennung aus Anwendersicht wünschenswert. Damit einher geht eine weitere Rolle in dem Entwurfs- und Implementierungsprozeß. Statt wie bisher Klienten- und Dienst-Implementierer, die Anwendungsobjekte modellieren und implementieren, wird durch den systemnahen Charakter der Dienstgüte-Erbringung die Rolle des *Dienstgüte-Implementierers* induziert. Dieser ist für das Bereitstellen von Verhalten zuständig, das wegen plattformspezifischen und Dienstgütebezogenen Details von Anwendungsobjekten separiert werden sollte.

Einbettung: Die unter "Trennung der Verantwortlichkeiten" geforderte Struktur ist schwer zu realisieren, da Dienstgüte-Mechanismen bei der Integration in das Objektmodell Aspekte im Sinn der aspektorientierten Programmierung darstellen. Die Einbettung von Dienstgüte-Vorkehrungen sollte die zugrundeliegende Struktur nicht oder nur unwesentlich beeinflussen. Insbesondere sollten Klienten und Dienste im System, die keine Dienstgüte-Befähigung besitzen, keinen Änderungen unterworfen sein. Hingegen sollten die zur Nutzung von Dienstgüte-Befähigung notwendigen Änderungen möglichst angelehnt an die der Basisinfrastruktur sein.

Ein weiterer Aspekt der Einbettung betrifft die Integration von Dienstgüte-Mechanismen, die bereits auf einer Plattform existieren. Dies können

vom Betriebssystem oder Netzwerk bereitgestellte Mechanismen sein, die von einem Dienstgüte-Mechanismus in der Architektur wiederverwendbar sein sollten.

Umfassend: Die Unterstützung für das Dienstgüte-Management geht über die reine Erweiterung für die Spezifikation und Einbettung in die Objektstruktur von Anwendungen hinaus. So sind zur Laufzeit für die Bindung zwischen Klient und Dienst Verhandlungen notwendig, und auch Infrastrukturdienste sind geeignet zu modifizieren oder zu ergänzen. Eine Dienstgüte-Architektur sollte einen Grundstock solcher Vorkehrungen bereitstellen und erweiterbar gehalten sein, damit nicht aus einem solchen Mangel eine Vermischung von verschiedenen Architekturen entsteht.

3.7.3 Phasen des Dienstgüte-Managements

Die in 3.7.2 formulierten Anforderungen sollen nun anhand zweier Phasen der Lebenszeit eines verteilten Systems näher betrachtet werden. Dabei liegt der Augenmerk auf der Entwurfs- und Implementierungsphase und der Laufzeit. Diese Phasen entsprechen denen des in Kapitel 2 vorgestellten verteilten Objektmodells. Aus den hier dargelegten Aufgaben zu den ensprechenden Phasen wird in Kapitel 4 eine Rahmenwerk für das Dienstgüte-Management abgeleitet und die Unterstützung der Entwurfs- und Implementierungsphase in den folgenden Kapiteln vertieft.

Entwurfs- und Implementierungszeit

Zur Entwurfs- und Implementierungszeit werden Schnittstellen von Diensten definiert und die korrespondierenden Anwendungsobjekte, also Dienste und Klienten, implementiert. Dazu werden die Schnittstellen in der IDL definiert und durch den IDL-Compiler in eine Zielsprache übersetzt. In der Zielsprache kann das Dienst-Skelett durch den Dienstimplementierer vervollständigt werden.

Die Dienstgüte-bezogenen Aufgaben der Entwurfs- und Implementierungszeit sind die Repräsentation von Dienstgüte-Charakteristiken durch eine geeignete Spezifikation und die Integration der zugehörigen Dienstgüte-Mechanismen.

Spezifikation Es existieren verschiedene Varianten der Spezifikation von Dienstgüte-Parametern. Diese können durch Einführung einer getrennten Spezifikation, der Erweiterung der Schnittstellenbeschreibungssprache oder aber durch Abstraktionen der Zielsprache (Objekte, Prozeduren, Strukturen) realisiert werden. Dieselben Varianten gelten für die Spezifikation der Schnittstelle der Dienstgüte-Mechanismen. Aufgrund der Notwendigkeit, Schnittstellen der Dienstgüte-Mechanismen auch entfernt aufzurufen, wie auch Dienstgüte-Parametern zwischen Klient und Dienst auszutauschen, ist eine Beschreibung in IDL bzw. eine Umsetzung nach IDL notwendig.

Um die Komplexität der Beschreibung niedrig zu halten, ist es wünschenswert, eine Dienstgüte-Charakteristik im System durch ein syntaktisches Konstrukt zu spezifizieren. Dieses Konstrukt muß entsprechende Typen erlauben, um beliebige Dienstgüte-Parameter beschreiben zu können. Die Schnittstellenspezifikation der Dienstgüte-Mechanismen sollte analog flexible Spezifikationen bieten. Eine solche Dienstgüte-Spezifikation muß dann in IDL übersetzt werden, wenn keine unmittelbare Beschreibung in IDL vorliegt.

Neben der Spezifikation der Parameter und der Schnittstelle einer Dienstgüte-Charakteristik ist die Bindung einer solchen Charakteristik an einen Dienst notwendig. Dies ist eine Aufgabe zur Laufzeit. Lediglich die Zuordnung, welche Dienstgüte von einem Dienst unterstützt wird, kann bereits zur Entwurfs- und Implementierungszeit geschehen. Diese Zuordnung kann einen Dienst auf genau eine Dienstgüte-Charakteristik ($|Q| = 1$) beschränken oder aber eine beliebige Menge von Charakteristiken zulassen ($|Q| = n$).

In Kapitel 5 wird ein Überblick der verschiedenen Varianten von Dienstgüte-Spezifikationen gegeben. Diese werden bewertet und eine Erweiterung der IDL als Basis eines aspektorientierten Ansatzes gewählt und vorgestellt.

Mechanismen-Integration In Abbildung 2.3 (Seite 19) ist eine Übersicht der Einheiten während eines verteilten Methodenaufrufs im verteilten Objektmodell zu sehen. Dabei separiert der ORB Betriebssystem und Netzwerk von der Anwendung. Für die Anwendungsobjekte stellen die Stellvertreterobjekte den Zugang zum verteilten Objektsystem dar. Wird in diese Sicht die in Abbildung 3.1 dargestellte Dienstgüte-Hierachie integriert, so ergeben sich daraus drei Integrationspunkte für Dienstgüte-Mechanismen:

- **Dienstgüte-Mechanismen und Anwendungsobjekte:** Die in Kapitel 3.5 vorgestelle Klassifizierung von Dienstgüte als Aspekt im Sinne der AOP macht deutlich, daß ein Teil der Dienstgüte-Mechanismen auf der Ebene der Anwendungsobjekte existiert. Generell läßt sich diese Ebene als *Dienstgüte-Mechanismen oberhalb des ORBs* charakterisieren. Dazu gehören beispielsweise auch Mechanismen für die Multimediadaten-Übertragung [4], die typischerweise am ORB vorbeigeführt werden.

- **Dienstgüte-Mechanismen im ORB:** Aufgrund der Ende-zu-Ende-Sicht auf die Dienstgüte-Erbringung ist es notwendig, eine Hierarchie von Dienstgüte-Mechanismen zu etablieren. Dabei sind innerhalb des ORBs Vorkehrungen für das Dienstgüte-Management zu integrieren. Scheduling von Aufrufen aufgrund von Prioritäten wie auch Protokollanpassungen innerhalb des ORBs gehören hierzu.

- **Dienstgüte-Mechanismen zum Netzwerk/Betriebssystem:** Die Implementierung eines ORBs basiert auf den Schnittstellen und Mechanismen des zugrundeliegenden Betriebsystems bzw. Netzwerks. Dort bereits realisierte Dienstgüte-Vorkehrungen sollten genauso nutzbar sein wie die Verwendung von Bibliotheken, wie bspw. das ISIS-Toolkit für Gruppenkommunikation.

Die in Abbildung 3.2 dargestellte Strukturierung von Dienstgüte-Mechanismen in einer Hierarchie wird durch diese drei Schichten unterstützt. Die Integration von Dienstgüte-Mechanismen wird in Kapitel 6 vertieft.

Umsetzung der Dienstgüte-Spezifikation Die Vermaschung der funktionalen und nicht-funktionalen Aspekte sollte durch eine Umsetzung der Dienstgüte-Spezifikation in entsprechende Einheiten der Zielumgebung, d.h. der Dienstgüte-Mechanismen oberhalb des ORBs, aufgelöst werden. Die Abhängigkeit der Aspekte (Dienstgüte, Anwendung) könnte nach den Prinzipien der AOP oder durch geeignete Entwurfsmuster entkoppelt werden.

Abbildung 3.8 zeigt die drei Ebenen der Dienstgüte-Mechanismen-Integration in einer Verteilungsinfrastruktur. Dabei ist angedeutet, daß auch Interaktion zwischen Klienten und Diensten ohne Dienstgüte-Vorkehrungen parallel möglich sein soll. Neben den Dienstgüte-Mechanismen, die oberhalb

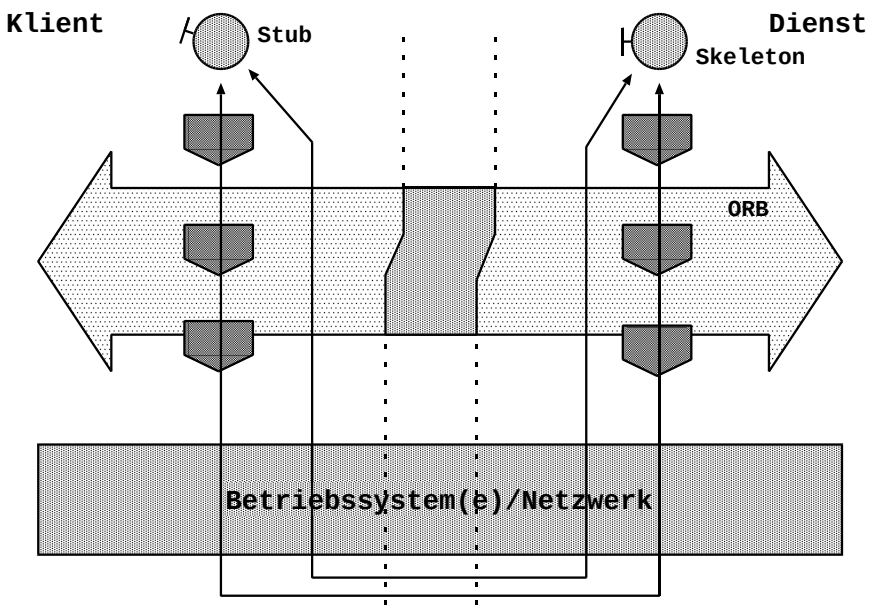


Abbildung 3.8: Mechanismen-Hierarchie

des ORBs realisiert sind, ist bei den Schichten im ORB bzw. zum Netzwerk hin darauf zu achten, daß geeignete Schnittstellen für die Integration zu schaffen sind. Gerade bei einer generischen Dienstgüte-Architektur ist sicherzustellen, daß Mechanismen beliebiger Dienstgüte-Kategorien integrierbar sind und weiterhin, daß auch verschiedene Mechanismen parallel existieren können. Auch hier tritt wieder die Dimension der Dynamik auf. Statische Architekturen könnten bei der Übersetzung des ORBs eine Menge bestimmter Dienstgüte-Mechanismen dazubinden, während dynamische Varianten die Integration zur Laufzeit zulassen. Wesentlich für die Wiederverwendbarkeit bereits implementierter Dienstgüte-Mechanismen ist deren klare Struktur entlang der Mechanismen-Hierarchie. Dies sollte durch geeignete Schnittstellen unterstützt werden.

Die Integration der Anwendungsobjekte mit den Dienstgüte-Vorkehrungen oberhalb des ORBs ist Aufgabe der Umsetzung von der Dienstgüte-Spezifikation in die jeweilige Zielsprache. Die Vermaschung der Dienstgüte-Mechanismen und Anwendungsobjekte macht hier besondere Vorkehrungen zur Trennung wünschenswert.

Die Verbindung zwischen der in Kapitel 5 vorgestellten Dienstgüte-Spezifikation und den Dienstgüte-Mechanismen wird in Kapitel 6 betrachtet.

Laufzeit

Während zur Entwurfs- und Implementierungszeit die Integration von Vorkehrungen für die Dienstgüte-Erbringung im Vordergrund stand, ist zur Laufzeit diese Dienstgüte-Erbringung zu gewährleisten. Dabei treten potentiell verschiedene Klienten und Dienste in Konkurrenz um Ressourcen zueinander. Neben der Zuteilung und Steuerung der Ressourcen, die im wesentlichen die Abstimmung der Dienstgüte-Mechanismen mit den Verwaltungseinheiten des Dienstgüte-Managements betrifft, müssen hier auch andere dynamische Fragestellung, wie das Binden von Klient und Dienst oder die Konfiguration der Dienstgüte-Mechanismen, behandelt werden.

Binden von Klient und Dienst

Der Bindevorgang zwischen Klient und Dienst mit Dienstgüte-Befähigung ist abhängig von der Spezifikation der Dienstgüte-Charakteristik und der entsprechenden Zuordnung an den Dienst. Die gütebezogenen Bestandteile des Dienstes sind dynamischer Natur und benötigen eine Verhandlung, da nicht a priori feststeht, ob diese erbracht werden können. Somit teilt sich ein Bindevorgang in Systemen mit Dienstgüte-Befähigung in Dienstvermittlung und Verhandlung auf.

Dienstvermittlung Die Erweiterung der Dienstvermittlung ist abhängig von der Repräsentation der Dienstgüte-Charakteristiken im System. Werden beispielsweise Dienstgüte-Charakteristiken durch Schnittstellen in der IDL abgebildet und einem Dienst durch Vererbung zugeführt, so ist ein herkömmlicher Vermittler ausreichend. Bei der Einführung einer Dienstgüte-Sprache ist auf eine entsprechende Abbildung der Dienstgüte-Befähigung eines Dienstes im Dienstvermittler zu achten. Dabei kann eine geeignete Erweiterung der Typspezifikation des Dienstvermittlers um die Dienstgüte-Aspekte eine Lösung darstellen. Auch die Wahl einer Spezifikation, die über die reine Typinformation hinausgeht (bspw. Konzeptgraphen [79]), kann hier Abhilfe schaffen.

Dienstgüte-Verhandlung Das Ziel einer Dienstgüte-Verhandlung ist eine Dienstgüte-Vereinbarung. Eine Dienstgüte-Vereinbarung zwischen einem Dienst D und einem Klienten K legt eine bestimmte Dienstgüte-

Charakteristik fest, mit der die Dienstleistung erbracht wird. Das Tripel $[K, D, \phi_i]$ stellt die Bindung eines Klienten und eines Dienstes dar, die durch die Einigung auf eine Dienstgüte-Charakteristik ϕ_i manifestiert wird. Der Dienst D erbringt seine Dienstleistung gegenüber dem Klienten K in dem mit $\overline{\phi}_i$ bezeichneten Dienstgüte-Niveau. Daher soll das Tripel $[K, D, \overline{\phi}_i]$ eine solche Dienstgüte-Vereinbarung bezeichnen.

An dieser Stelle sei angemerkt, daß es im Fall einer generischen Dienstgüte-Architektur ausreichend ist, den Fall $|Q| = 1$ zu betrachten. Angenommen, ein System erlaube die gleichzeitige Erbringung verschiedener Dienstgüte-Charakteristiken in der Form $[K, D, \phi_i, \phi_{i+1}, \ldots, \phi_{i+k}]$. Dann muß eine neue Dienstgüte-Charakteristik $\psi = \bigcup_{l=i}^{i+k} \phi_l$ eingeführt werden. Durch Wechsel von $Q = \{\phi_1, \ldots \phi_n\}$ zu $Q' = \{\bigcup_{l=1}^{i} \phi_l | i = 1 \ldots n\}$ besteht Q' nun aus allen Kombinationen von Dienstgüte-Mechanismen, die Q enthält. Damit ist $< D, Q' >$ nun ein Dienst, der jeweils eine Dienstgüte zur Zeit aus dieser Menge anbietet.

Für die eigentliche Verhandlung sind zwei Fälle zu unterscheiden:

1. *Einzelabstimmung:* Der Klient hat genau einen Wunsch nach einer Dienstgüte-Charakteristik ϕ. Kann die entsprechende Instanz $\overline{\phi}$ nicht geliefert werden, schlägt die Verhandlung fehl.

2. *Auswahlabstimmung:* Der Klient hat eine beliebige Instanz $\overline{\phi} \in \{\overline{\phi}_k, \ldots, \overline{\phi}_l\}$ als Wunsch. Die Verhandlung ist erfolgreich, solange eine vom Klienten gewünschte Instanz erfüllt werden kann.

Gerade in Hinblick auf adaptive Anwendungen, also solche, die differenziert auf unterschiedliche Dienstgüte-Niveaus reagieren können, ist die Auswahlabstimmung relevant. Allerdings ist es häufig notwendig, eine Präferenz des Klienten für die einzelnen Dienstgüte-Instanzen anzugeben, da nicht alle gleich präferiert werden. Zudem können ungünstig gewählte Parameter $P(\phi)$ dazu führen, daß die Aufzählungen von Klientenwünschen in der Größe zunehmen. Weiterhin ist bei Bündelungen von Abstimmungen durch Auswahlabstimmung eine scheinbare Parität zwischen den einzelnen Wünschen gegeben, die so selten vorkommt.

Da jeder Klient eine potentiell andere Bewertung geleisteter Dienstgüte besitzt, ist die bis jetzt eingeführte Abstraktion durch Zustand $P(\phi)$ und Schnittstelle $S(M_\phi)$, die für alle Klienten gleich ist, nicht ausreichend. Ein Beispiel dafür ist eine Dienstgüte, die durch zwei Parameter `kosten`

und `qualität` ausgedrückt wird. Dabei nehmen beide Parameter Werte aus {`hoch, mittel, niedrig`} an. Klienten können qualitätsorientierte, kostenorientierte oder Preis/Qualitäts-orientierte Präferenzen besitzen.

Die klientenspezifische Sicht wird durch *Klientenpräferenzen* ausgedrückt, die durch eine geeignete Spezifikation oder Verhandlungsstrategien repräsentiert werden müssen.

Dienstgüte-Mechanismen-Integration zur Laufzeit

Die Integration von Dienstgüte-Mechanismen zur Laufzeit ist abhängig von der Dimension "Dynamik". So kann eine statische Architektur sich nur auf bereits bei der Implementierungszeit in den ORB gebundene Dienstgüte-Mechanismen beschränken. Der Austausch bestehender Dienstgüte-Mechanismen zur Laufzeit ist nicht möglich.

Eine dynamische Erweiterung kann darin bestehen, daß Mechanismen mit derselben Schnittstelle ausgetauscht werden können. Somit lassen sich zumindest Verbesserungen der Mechanismen dynamisch im laufenden System realisieren.

Programmiersprachen wie Java [31] unterstützen das dynamische Nachladen von Klassen und deren Instanzen. Ein in einer solchen Sprache implementierter ORB kann die Dienstgüte-Mechanismen ensprechend nachladen.

In statisch gebundenen Programmiersprachen, wie C++ [90], läßt sich dies durch einen dynamischen Binder realisieren. Bieten alle Dienstgüte-Mechanismen dieselbe statische Schnittstelle, können diese durch einen dynamischen Binder ausgetauscht werden. Damit die spezifische Schnittstelle solcher Dienstgüte-Mechanismen genutzt werden kann, ist eine geeignete dynamische Aufrufschnittstelle zu implementieren.

Die dynamische Ersetzung bzw. Integration von Dienstgüte-Mechanismen ist für das generische Dienstgüte-Management wünschenswert. So können Objekte mit den jeweils benötigten Mechanismen versorgt werden. In Kapitel 6 wird eine Aufteilung der Schnittstelle von Dienstgüte-Mechanismen in statische und dynamische Bestandteile vorgestellt, die durch einen einfachen reflektiven Ansatz das dynamische Nachladen erlaubt.

Infrastruktur-Dienste

Neben Infrastrukturdiensten, wie die bereits angesprochene Dienstvermittlung, sind für ein umfassendes Dienstgüte-Management weitere Dienste notwendig. Es sollen hier exemplarisch drei Dienste aufgeführt werden.

Überwachung Die geleistete Dienstgüte, insbesondere hinsichtlich einer Dienstgüte-Vereinbarung, ist für den Klienten überprüfbar. Aufgrund der Ende-zu-Ende-Charakteristik von Dienstgüte ist es wünschenswert, daß Verletzungen von Dienstgüte-Vereinbarungen bereits in den unteren Schichten erkannt werden. Damit ist es möglich, entsprechende Gegenmaßnahmen einzuleiten, bevor die Anwendung betroffen ist. Dies kann beispielsweise durch eine Alternativ-Route im Netzwerk geschehen, die eine größere Bandbreite erlaubt. Die Migration eines Dienstes auf einen schnelleren Rechner bei Überlastung oder das Starten neuer Replikate bei Dienstausfällen sind Beispiele für Anpassungen der Dienstgüte-Mechanismen, die vor der Anwendung verborgen bleiben. Darüberliegende Schichten der Dienst-Erbringung und der Dienstgüte-Mechanismen können so transparent weiterarbeiten.

Ein Überwachungsdienst sollte in der Lage sein, flexible Überwachungen durchzuführen, entsprechende Überprüfungen der ermittelten Werte vorzunehmen und bei Verletzung des eingestellten Werte-Bereichs eine Benachrichtigung an ein Objekt zu senden, das entsprechende Gegenmaßnahmen einleitet.

Ressourcen-Steuerung Die Erbringung einer bestimmten Dienstgüte-Charakteristik ist mit Ressourcen des Systems verknüpft. Dabei kann "Bandbreite" oder "Rechenzeit" als Ressource aufgefaßt werden. In der Konkurrenz zwischen verschiedenen Dienst/Klienten-Beziehungen muß eine Zuteilung der Ressourcen zu den jeweiligen Klienten und Diensten erfolgen. Dieses kann nur durch entsprechende Steuerung erfolgen. Die vornehmliche Aufgabe der Ressourcen-Steuerung ist es, Ressourcen-Reservierungen von Dienstgüte-Mechanismen anzunehmen. Dabei müssen Prioritäten und Vorabreservierungen möglich sein, um systemrelevanten Diensten entsprechende Ressourcen zuteilen zu können.

Abrechnung Bei der Inanspruchnahme bestimmter Dienste und eines Dienstgüte-Niveaus können zwei Muster unterschieden werden. Zum einen kann die Dienstgüte-Erbringung als notwendige Infrastrukturleistung gesehen werden. Dies trifft beispielsweise bei "Verfügbarkeit" im Bereich von Finanzdienstleistungen zu. Eine andere Sicht der Dienstgüte-Erbringung kann als Qualitätsmerkmal eines Dienstes betrachtet werden, die für den Klienten einen bestimmten Mehrwert darstellt. Videoübertragungen mit unterschiedlichen Qualitäten – Bildgröße oder Rahmenrate – oder aber das Anbieten eines Dienstes an andere Nutzer, die für die Bereitstellung des Dienstes und evtl. bestimmte Leistungszusagen wie Antwortzeit oder eine entsprechend große Datenbasis bei einem Informationsdienst bezahlen, sind Beispiele hierfür. Erfolgt keine Differenzierung des Preises in Abhängigkeit der geleisteten Dienstgüte, ist es wahrscheinlich, daß Klienten höhere Qualität anfordern, als eigentlich benötigt würde. Eine qualitätsabhängige Abrechnung kommt hier sowohl Klienten wie auch der Systemumgebung zugute. Klienten erhalten die Möglichkeit, in Abhängigkeit der geforderten Qualität die Bepreisung der Dienstleistung zu steuern. Im Gegenzug ist es wahrscheinlich, daß unnötige Ressourcen-Monopolisierungen verhindert werden, da auch für nicht genutzte bzw. nicht benötigte Dienstqualität gezahlt werden muß.

3.8 Stand der Forschung

Mit der zunehmenden Verbreitung objektorientierter Verteilungsinfrastrukturen ist auch das Bedürfnis nach Dienstgüte-Erbringung durch solche Plattformen gestiegen. Dabei haben sich recht früh zwei Dienstgüte-Kategorien ausgebildet, die besonderes Interesse gefunden haben: Echtzeit und Fehlertoleranz.

Echtzeit Die Advanced Network Systems Architecture (ANSA) [1] stellt einen der ersten Vertreter objektorientierter Verteilungsinfrastrukturen dar. Bereits recht früh wurde untersucht, wie Dienstgüte-Charakteristiken in der Schnittstellenbeschreibung und in der Zielsprache eingebettet werden können [55]. Dabei wurde hauptsächlich der Echtzeitbereich adressiert, dieser aber auf die Übertragung kontinuierlicher Medien beschränkt. Die Darstellung von Dienstgüte-Mechanismen durch Pseudo-Objekte, wie sie ANSA

eingeführt hat, findet sich in aktuellen Ansätzen der OMG wieder. Mit der geringen Verbreitung von ANSA ist allerdings auch der vorgeschlagene Weg der Dienstgüte-Integration wenig beachtet worden.

Die Telecommunications Information Networking Architecture (TINA) [49] stellt eine Referenzplattform für verteilte Objektsysteme dar. Das Bestreben der TINA war es, die Bedürfnisse hinsichtlich unternehmesweiten und -übergreifenden Einsatz von verteilten Objektsystemen in der Telekommunikation geeignet zu unterstützen. So wurden auch früh Spezifikationen für Fehlertoleranz [94] und Echtzeit [95] formuliert. Dabei wurde die TINA allerdings häufig nur als Referenzmodell betrachtet und für Implementierungen auf CORBA als Plattform zurückgegriffen. Das ReTINA-Projekt [2] stellt eine Evaluierungsimplementierung der TINA dar, die insbesondere auch Dienstgüte-Management berücksichtigt. Dabei ist ReTINA allerdings auf den Echtzeitbereich beschränkt.

Mit der Verbreitung von CORBA als Verteilungsinfrastruktur sind auch hier recht früh Betrebungen zur Integration von Dienstgüte-Vorkehrungen der Kategorie Echtzeit zu sehen gewesen. The ACE ORB (TAO) [54], [85] stellt einen Ansatz von Echtzeit-Vorkehrungen in verteilten Objektsystemen dar. Dabei basiert TAO auf einem Entwurfsmuster-orientierten Design [86]. Dienstgüte-Spezifikationen werden durch eine Struktur in IDL ausgedrückt. Vor der Laufzeit werden die Abhängigkeiten geprüft und ein Schedule erzeugt, das die Interaktionen zur Laufzeit steuert.

Mittlerweile hat auch die OMG dem Bedarf an Echtzeit-Unterstützung durch eine Spezifikation Rechnung getragen [74]. Dabei sind wesentlich Erfahrungen aus der TAO-Gruppe eingeflossen und mit Neuerungen der CORBA 2.2 und 2.3 Spezifikation verschmolzen worden. Wesentlich ist bei diesem Ansatz die Bemühung, die CORBA nicht durch IDL-Änderungen zu modifizieren, sondern nur durch Richtlinien-Objekte (engl.: Policy Objects) die Konfiguration der Dienstgüte-Mechanismen zu erlauben.

Eine Unterstützung der Übertragung kontinuierlicher Medien aus dem Multimedia-Bereich adressiert eine Spezifikation der OMG-Telekom-Domäne [73]. Diese regelt im wesentlichen die Repräsentation von Dienstgüte-Parametern durch den Property-Service [68] und den Verbindungsaufbau durch einen Satz von Management-Funktionen. Auf die Implementierung der Mechanismen und die resultierende Anwendungsstruktur wird in der Spezifikation nicht eingegangen [46].

Fehlertoleranz Ein früher Vertreter einer CORBA-Implementierung mit Dienstgüte-Befähigung ist Electra [61]. Die CORBA wird um Methoden erweitert, mit der Objekte zur Laufzeit Objektgruppen bilden können [62]. Die Antworten einer Gruppe können für einen Mehrheitsentscheid genutzt werden. Wird nur eine Antwort benutzt, fungiert der Rest der Gruppe als Redundanz für Verfügbarkeit. Electra verwendet einen hierarchisch organisierten ORB-Kern, der die Integration verschiedener Multicast-Protokolle, wie ISIS [15] oder Horus [97], erlaubt. Electra bietet keine explizite Spezifikation für Dienstgüte-Parameter. Das Bilden und die Benutzung von Objektgruppen erfolgt durch Methoden, um die der ORB und der Objekt Adapter erweitert wurden.

Fehlertoleranz kann auch auf andere Art und Weise in CORBA integriert werden. Zwei Beispiele, die über die Einführung eines Dienstes gelöst werden, sind der Object Group Service (OGS) [29] und Maestro [99]. Während der OGS einen herkömmlichen CORBA-basierten Ansatz darstellt, greift Maestro auf die Interoperabilität von CORBA über IIOP (Internet Inter ORB Protocol) zurück. Aus Sicht der Klienten stellt sich eine Anfrage über IIOP wie ein herkömmlicher Methodenaufruf dar. Während Klienten so von einem beliebigen ORB Aufrufe an eine Objektgruppe absetzen können, benötigen Dienste eine entsprechende Unterstützung des ORBs und Beachtung des Dienstimplementierers. Auch in Maestro existiert keine explizite Spezifikation von Dienstgüte-Charakteristiken. Der OGS benötigt im Gegensatz zu Maestro, das auf der Dienstseite einen angepaßten ORB nutzt, keine Erweiterung der zugrundeliegenden Plattform. Die Komplexität der Dienstgüte-Implementierung wird in Objekt-Dienste verschoben, die Gruppenkommunikation, Fehlererkennung und Gruppenkonfiguration anbieten. Anwendungsobjekte nutzen diese Dienste, um Objektgruppen zu konfigurieren und mit diesen zu kommunizieren.

Das Eternal-Projekt [65] ermöglicht die transparente Nutzung von Dienstgüte-Mechanismen aus der Fehlertoleranz-Kategorie sowohl für Klienten als auch für Dienste. Dazu werden die Aufrufe von Klienten beim ORB-Eintritt abgefangen und entsprechend modifiziert, so daß ein zugrundeliegendes Multicast-Protokoll diese an eine Objektgruppe verteilt. Auf der Dienstseite wird durch ähnliche Mechanismen eine Verteilung des Dienstzustandes erreicht. Die Konfiguration erfolgt durch einen Administrator. Klienten und Dienste sind sich der Dienstgüte-Erbringung nicht bewußt.

Die CORBA-Messaging-Spezifikation [72] behandelt hauptsächlich Synchronisationsaspekte bei Methodenaufrufen. Klienten können steuern, wann sie einen Auftrag als zugestellt betrachten. Dabei können neben den Sychronisationspunkten (Annahme des Auftrags durch das Netzwerk, den ORB, den Objekt-Adapter oder den Dienst) auch weitere Informationen, wie die maximale Anzahl von Zwischenknoten oder Timeouts angegeben werden. Die Spezifikation der Dienstgüte-Parameter erfolgt durch Richtlinien-Objekte. Die Anbindung in die Anwendung geschieht durch das Asynchronous Messaging Interface, das Anwendungen erlaubt, auf Nachrichten aktiv zu warten (pollen) oder aber asynchron bei Vorliegen des Ergebnisses benachrichtigt zu werden (Callback). Die weitergehende Unterstützung von Fehlertoleranz in CORBA ist das Ziel weiterer Spezifikationen, die allerdings zur Zeit noch in den Anfängen der Standardisierung stehen.

Generische Dienstgüte-Architekturen

Im Gegensatz zu den bislang vorgestellten Ansätzen für Dienstgüte-Management in Objektsystemen existieren mittlerweile unterschiedliche Varianten, die generisches Dienstgüte-Management erlauben.

Adaptive Middleware stellt ein Konzept dar, das die Flexibilität der ORB-Architektur gegenüber Änderungen adressiert. Da für das Dienstgüte-Management eine solche Flexibilität unabdingbar ist, haben sich solche Ansätze auch in Gruppen entwickelt, die sich mit dem Dienstgüte-Management beschäftigen.

Ein die Prinzipien der AOP in die Struktur der Middleware integrierender Ansatz sind die "Composition Filters" [13]. Dabei modelliert ein Composition Filter jeweils einen Aspekt eines Objekts. Objekte können durch den Zusammenschluß mehrerer Composition Filter gebildet werden. Composition Filter stellen einen durchgängigen Mechanismus für die Dienstgüte-Integration von der Anwendungsschnittstelle durch die Middleware dar.

Meta-Spaces [16] bilden einen Raum, der die Schnittstelle eines Objektes in die Verantwortlichkeiten (Concerns) aufteilt. Diese Verantwortlichkeiten werden durch Objekte realisiert. Durch die Komposition eines Objektes aus den verschiedenen Verantwortlichkeiten ergibt sich das für Anwender sichtbare Objekt. Dieses stellt eine Schnittstelle nach außen dar, die auf die jeweiligen Verantwortlichkeiten, respektive deren Objekte, delegiert. Eine

entsprechende Schnittstelle erlaubt die Konstruktion solcher Objekte aus unterschiedlichen Verantwortlichkeiten. Meta-Spaces können auch als ein auf eine Aufgabe angepaßtes Komponentenmodell aufgefaßt werden.

Beiden Ansätzen ist gemeinsam, daß der Aspekt-Charakter von Dienstgüte im Sinne der AOP adressiert wird. Es werden meta-sprachliche Konstrukte zur Zusammenführung der Aspekte bereitgestellt.

Dienstgüte-Rahmenwerke Bislang wurde für Infrastrukturen, die Dienstgüte-Management ermöglichen, der Begriff der Dienstgüte-Architektur benutzt. Die softwaretechnische Struktur einer Anwendung wird als Architektur bezeichnet. Die Freiheitsgrade einer Architektur gegenüber Erweiterungen durch Anwendungs- oder Systementwickler sind mitunter gering. Gerade in Hinblick auf die Erweiterbarkeit und Wiederverwendbarkeit sind allgemeinere Strukturen, aus denen Anwendungsarchitekturen gebildet werden können, wünschenswert. Ansätze, die auf einer bestehenden Verteilungsinfrastruktur wie CORBA basieren, stellen im Sinne von [35] Rahmenwerke (engl.: Frameworks) dar:

A set of cooperating classes that makes up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes.

Somit existiert eine Wechselwirkung zwischen Rahmenwerk und Architektur. Ein Rahmenwerk diktiert eine Anwendungsarchitektur. Die grundlegende Struktur durch Partitionierung in Klassen und deren Abhängigkeiten wird bereits durch das Rahmenwerk vorgenommen. Rahmenwerke stellen also allgemeinere Strukturen als Architekturen dar.

Streng genommen stellt die CORBA an sich schon ein Rahmenwerk dar, das durch die Schnittstellenbeschreibung in den Zielsprachen durch den IDL-Compiler instantiiert wird. Aus der Schnittstellenbeschreibung werden korrespondierende Klassen in den Zielsprachen erstellt. Diese Klassen – in Kombination mit der Funktionalität des ORBs – bilden das Rahmenwerk einer Anwendung. Die Anwendungsentwickler vervollständigen in diesem Rahmenwerk die Anwendungslogik und erhalten die Möglichkeit der entfernten Kommunikation durch das Rahmenwerk.

Wird eine solche Verteilungsinfrastruktur als Grundlage für das Dienstgüte-Management benutzt und um Spezifikationen für Dienstgüte-Charakteristiken und die Integration von Dienstgüte-Mechanismen erweitert, bleibt der Charakter eines Rahmenwerkes bestehen. Ein Dienstgüte-Rahmenwerk kann durch entsprechende Generierung aus der IDL und der Spezifikation von Dienstgüte-Charakteristiken in den Zielsprachen die Trennung zwischen Anwendungs- und Dienstgüte-bezogenem Verhalten ermöglichen.

An den HP-Labs in Palo Alto ist eine Reihe von Komponenten für ein CORBA-basiertes Dienstgüte-Management-Rahmenwerk entstanden. Die QML (QoS Modeling Language) [34] stellt dabei das Fundament der Komponenten dar. QML ist sowohl in der Analyse und im Design (vgl. [52], als auch zur Generierung entsprechender Objekte zur Durchsetzung der Dienstgüte-Charakteristiken [33] einsetzbar. Dabei stellt QML eine generische Spezifikation von Dienstgüte-Charakteristiken dar. Zur Verhandlung wird jedoch eine getrennte Beschreibung verwendet [53].

Ein auf QML basierender Ansatz sind die Reflective-Reconfigurable Interconnectable Objects (R-RIO) [91]. Die Definitionen – in CBabel, einer QML-Untermenge – werden in Vorlagen der Zielsprache übersetzt. Dabei wird von R-RIO explizit generisches Mehrkategorie Dienstgüte-Management unterstützt.

Ein weiterer generischer Ansatz für Dienstgüte-Management verwendet eine in XML spezifizierte Dienstgüte-Spezifikation. Diese QoS Description Language (QDL) [24] wird in ein Rahmenwerk übersetzt [23], das entsprechend der in diesem Kapitel vorgestellten Anforderungen Unterstützung zu beiden Phasen des Dienstgüte-Managements anbietet.

Aspectix [41] ist eine in Entwicklung befindliche Arbeit. Die unterschiedlichen Verantwortlichkeiten werden in sogenannte Fragmente unterteilt, die zusammengesetzt die Objektverantwortlichkeiten realisieren. Dabei folgt Aspectix dem Leitbild der aspektorientierten Programmierung.

Das Projekt Quality Objects (QuO) [103] stellt einen weiteren aspektorientierten Ansatz dar. Die Unterstützung durch Dienstgüte-Mechanismen (Delegates) oberhalb eines modifizierten CORBA-konformen ORB-Kerns bildet die Laufzeitunterstützung von QuO [98]. Drei Aspektsprachen, für Ressourcen, Kontrakte und Struktur [59], werden durch eine Umsetzung in die Zielsprache mit den Anwendungsobjekten verwoben [60]. Den Kern der Spezifikation bildet die Kontraktbeschreibung. Ein Kontrakt legt die

Dienstgüte-Anforderung einer Verbindung zwischen Klient und Dienst fest. Diese werden durch Regionen beschrieben. Eine Region legt den dabei gültigen Wertebereich der Dienstgüte-Parameter fest.

Zusammenfassende Bewertung

Obwohl Dienstgüte-Befähigung in verteilten Objektsystemen als Bedürfnis erkannt ist, existieren hauptsächlich Speziallösungen, die jeweils eine Dienstgüte-Kategorie adressieren. Das "Green Paper on Quality of Service" [71] hat zwar versucht, in der OMG einen integrativen Ansatz zu etablieren, ist aber in der Entwicklung aktueller Spezifikationen nicht berücksichtigt worden. Stattdessen existieren für verschiedene Bereiche, wie Echtzeit und Multimedia, getrennte Spezifikationen, die nicht aufeinander zurückgreifen. Die Kategorie "Fehlertoleranz" ist durch einen Aufruf zur Einreichung von Spezifikationen als wichtig erkannt worden. Das aktuelle Stadium läßt noch keine Beurteilung zu.

Aus dem Bereich der Forschung dominieren spezialisierte Architekturen für bestimmte Dienstgüte-Kategorien. Erste Bemühungen in Richtung generischen Mehrkategorie-Dienstgüte-Managements sind zu sehen. Die umfassendste Lösung stellt bislang QuO dar. Die Spezifikation der Dienstgüte-Charakteristiken durch drei zusätzliche Beschreibungssprachen ist recht komplex.

Bislang adressieren wenige Ansätze die Spezifikation von Klientenpräferenzen, um abgestufte Dienstgüte-Verhandlungen zu ermöglichen. Im Multimedia-Bereich gibt es zwei Ansätze, die dies durch einen Dienstvermittler und die Einbettung der Dienstgüte-Parameter in einen Vektorraum lösen [27],[56]. Dabei lassen sich aber einige Anomalien, wie sie in [51] beschrieben sind, nicht auflösen. Mathematisch anspruchsvollere Verfahren, wie sie in [53] beschrieben werden, sind für Endanwender zu komplex (vgl. [11]).

Abrechnung von Dienstleistung mit dem Einbezug der Qualität ist eine im Bereich der Internet-Dienstgüte aktuelle Diskussion [47]. In die Dienstgüte-Architekturen und Rahmenwerke für Objektsysteme hat diese Diskussion bislang kaum Eingang gefunden.

Im weiteren Verlauf dieser Arbeit wird ein Rahmenwerk für das Dienstgüte-Management in offenen verteilten Objektsystemen vorgestellt werden. Der Schwerpunkt wird dabei auf die Unterstützung der Entwurfs- und Implementierungsphase gelegt.

Kapitel 4

Ein Dienstgüte-Rahmenwerk

Dieses Kapitel stellt ein Rahmenwerk für das generische Dienstgüte-Management in verteilten Objektsystemen vor. Das hier vorgestellte Rahmenwerk wurde im Rahmen des Projektes "Management Architecture for Quality of Service" (MAQS) [6], [10] entwickelt.

4.1 Ziele des Rahmenwerks

Bei der Unterstützung von generischem Dienstgüte-Management ist es wünschenswert, daß bereits existierende Dienstgüte-Mechanismen des Netzwerks, Betriebssystems oder aber angepaßter Bibliotheken, wie für Gruppenkommunikation, eingebettet werden können. Für manche Dienstgüte-Charakteristiken ist es unabdingbar, auf Dienstgüte-Vorkehrungen tieferer Schichten, wie Bandbreite-Reservierung des Netzwerks, zuzugreifen. Daher muß ein generisches Dienstgüte-Rahmenwerk die Integration solcher Mechanismen unterstützen. Weiterhin ist die Strukturierung der Dienstgüte-Mechanismen von Bedeutung, damit entsprechend der Ausführungen in Kapitel 3 Bestandteile der Dienstgüte-Mechanismen-Hierarchie wiederverwendbar sind. Dies erlaubt zudem eine einfachere Einbettung der zugrundeliegenden Dienstgüte-Mechanismen.

Die Spezifikation der Dienstgüte-Parameter und der Schnittstellen der Dienstgüte-Mechanismen soll trotz großer Freiheit einfach nutzbar sein. Nicht Dienstgüte-befähigte Dienste und Klienten sollten durch die Einbettung des Dienstgüte-Managements in die zugrundeliegende Verteilungsinfrastruktur keinen Änderungen unterworfen sein. Das bedingt, daß die für die Dienstgüte-Nutzung bereitgestellten Konstrukte im Einklang mit den

Prinzipien der zugrundeliegenden Verteilungsinfrastruktur stehen. Die dynamische Erweiterung der Dienstgüte-Mechanismen bei bereits laufendem ORB kann bei Migration von Anwendungsobjekten mit Dienstgüte-Anforderungen notwendig sein. Somit ist ein Mindestmaß an Reflektivität der Verteilungsinfrastruktur zu ermöglichen.

Die generische Dienstgüte-Unterstützung bedingt explizite Dienstgüte-Integration. Das Rahmenwerk unterstützt diese durch eine geeignete Spezifikation von Dienstgüte-Charakteristiken. Die Umsetzung der Dienstgüte-Spezifikation folgt dem Leitbild der aspektorientierten Programmierung. Dies erlaubt die Trennung der Verantwortlichkeiten von Anwendung und Dienstgüte-Vorkehrungen.

Während der Laufzeit ist die Etablierung von Dienstgüte-Vereinbarungen durch eine Verhandlung zu unterstützen. Weiterhin sind Dienste für die Dienstvermittlung, Ressourcen-Steuerung, Überwachung und Abrechnung bereitzustellen.

Im weiteren Verlauf dieses Kapitels werden die Kernelemente des Rahmenwerkes in der Entwurfs- und Implementierungsphase sowie der Laufzeit vorgestellt. Dabei wird an dieser Stelle die Entwurfs- und Implementierungsphase kürzer behandelt und dient nur dem Überblick, da diese in den folgenden Kapiteln vertieft wird.

4.2 Entwurfs- und Implementierungsphase

4.2.1 Dienstgüte-Spezifikation

Die Entkopplung der Aspekte (Klient, Dienst und Dienstgüte) wird konzeptionell durch die Spezifikation der Dienstgüte-Charakteristiken in einer erweiterten Schnittstellenbeschreibungssprache ermöglicht. Die Umsetzung der IDL in die jeweilige Zielsprache kann aus den Informationen entsprechende Entkopplungen der Implementierungen erzeugen. Dabei fungiert der erweiterte IDL-Compiler als AspectWeaver.

Die erweiterte IDL (Quality of Service IDL, QIDL) stellt zwei Grundelemente als Erweiterung zur Verfügung:

- **Dienstgüte-Schnittstelle:** Jede Dienstgüte-Charakteristik wird im Rahmenwerk durch eine QIDL-Spezifikation – die Dienstgüte-Schnittstelle – definiert. Bestandteile der Dienstgüte-Spezifikation sind die

von der Dienstgüte-Implementierung angebotenen Verantwortlichkeiten, also die Schnittstelle der entsprechenden Dienstgüte-Mechanismen. Weiterhin sind die Dienstgüte-Parameter Bestandteil der Dienstgüte-Spezifikation.

- **Dienstgüte-Zuordnung:** Die Zuordnung, welche Dienstgüte-Charakteristik ein Dienst unterstützt, erfolgt zur Entwurfs- und Implementierungszeit in der QIDL. Ein dynamischere Variante ist durch die Vermaschung zwischen Anwendungs- und Dienstgüte-Aspekten bei generischem Dienstgüte-Management nicht möglich (vgl. AOP-Diskussion in Abschnitt 3.5). Dabei kann ein Dienst D eine beliebige Menge an Dienstgüte-Charakteristiken $\phi_1, \ldots, \phi_n$ unterstützen. Diese können auch aus unterschiedlichen Dienstgüte-Kategorien stammen. Zur Laufzeit wird aus diesen Dienstgüte-Charakteristiken durch eine Verhandlung eine Dienstgüte-Vereinbarung bestimmt.

4.2.2 Verweben der Aspekte

Die auf der Schnittstellenebene noch getrennten Aspekte werden in die jeweilige Zielsprache übersetzt. Abhängig von den in den Zielsprachen zur Verfügung stehenden Strukturierungsmechanismen (Vererbung, Delegation) kann durch die Umsetzung eine Trennung der Verantwortlichkeiten durch entsprechende Entwurfsmuster generiert werden. Dabei reichen die Informationen in der QIDL aus, um entsprechende Vorlagen zu erzeugen. Die Anomalien der Vermaschung zwischen den Aspekten, wie Zugriff auf den Zustand eines Dienstes, werden durch eine dedizierte Schnittstelle zwischen den Aspekten auflösbar.

In der Umsetzung werden aus der Dienstgüte-Schnittstelle zwei Arten von Einheiten in der Zielsprache generiert. Die Dienstgüte-Parameter werden in Einheiten, wie Verbunde (struct, record), abgebildet, damit diese durch den ORB transportiert werden können. Die Schnittstelle der Mechanismen wird auf Vorlagen der Dienst- und Klientenseite abgebildet, die der Dienstgüte-Implementierer realisiert. Dabei sind diese Vorlagen durch Entwurfsmuster mit den Anwendungsobjekten verwoben.

Weiterhin erlaubt der QIDL-Compiler die Generierung von Vorlagen für eine einfache Verhandlung sowie zur Konfiguration der Dienstgüte-Mechanismen.

4.2.3 Dienstgüte-Mechanismen-Integration

Die in Abbildung 3.8 (Seite 58) dargestellten Schichten werden in dem Rahmenwerk unterschiedlich unterstützt. Die Ebene der anwendungsbezogenen Dienstgüte-Mechanismen (Dienstgüte-Mechanismen oberhalb des ORBs) wird durch die vom QIDL-Compiler generierten Vorlagen der Dienstgüte-Mechanismen abgebildet.

Die Ebene der Dienstgüte-Mechanismen im und unterhalb des ORBs wird im Rahmenwerk durch zwei Konzepte ermöglicht. Die im Rahmen des Prototypen genutzte Infrastruktur bietet einen Mikrokern-Ansatz [83]. Damit reduziert sich der ORB auf eine Aufruf- und eine Weiterleitungsschnittstelle. Durch die Zuordnung von verschiedenen Aufruf- bzw. Weiterleitungsschnittstellen zu Objektreferenzen kann hier differenziertes Verhalten in Bezug auf das Schedule von Aufträgen oder Zugriff auf das Netzwerk geschehen.

Das Rahmenwerk unterstützt für entfernte Weiterleitung, also solche, die den Adreßraum verläßt, eine konfigurierbare Weiterleitungsschnittstelle. Diese läßt sich um Benutzer-definiertes Verhalten erweitern. Dies wird in Kapitel 6 vertieft behandelt.

4.3 Laufzeit-Unterstützung

4.3.1 Verhandlung

Das Rahmenwerk generiert aus den Definitionen der QIDL neben den Implementierungsvorlagen und deren Vermaschung auch unterstützender Code. Dazu gehören eine Menge von Methoden, die von jedem Dienstgüte-befähigten Dienst angeboten werden. Klienten können die möglichen Dienstgüte-Charakteristiken abfragen und eine einfache Verhandlung initiieren. Hierzu wird der Dienstgüte-Parameter $P(\phi)$ einer Dienstgüte-Charakteristik ϕ mit den gewünschten Zielwerten $\overline{\phi}$ gefüllt und dem Dienst übermittelt. Dieser prüft, ob die Anfrage erfüllbar ist und liefert eine entsprechende Bestätigung. Diese einfache Verhandlung stößt allerdings recht schnell an ihre Grenzen. Zum einen kann der Dienst aufgrund der klientenspezifischen Bewertung der Dienstgüte nicht die aus Klientensicht nächstbeste Instanz liefern (vgl. 3.7.3). Zum anderen kann der Kommunikationsaufwand

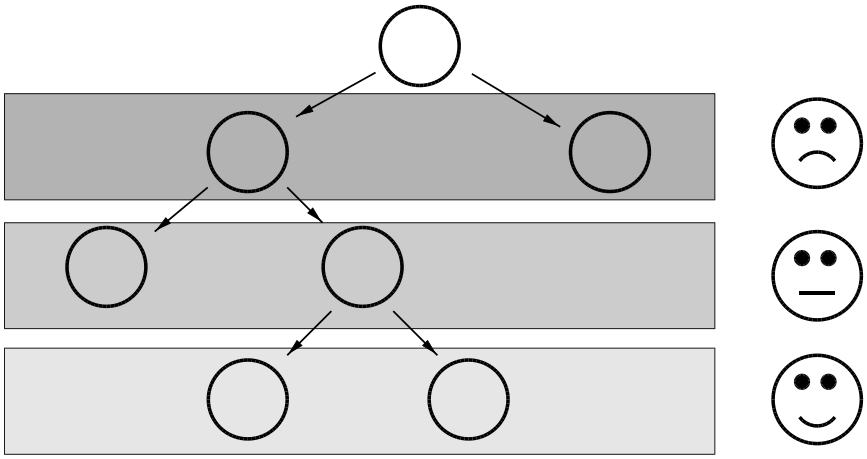


Abbildung 4.1: Präferenz-Hierarchie

recht hoch werden, wenn der Klient alle Kombinationen der Zustände eines Dienstgüte-Parameters sukzessive übermittelt.

Eine Abhilfe bieten Klientenpräferenzen. Dabei übermittelt der Klient in einer geeigneten Darstellung seine Präferenz der gewünschten Dienstgüte-Charakteristiken. Dies kann allerdings eine Explosion der zu übermittelnden Zustände zur Folge haben, wenn man beispielsweise bei Dienstgüte-Parametern an Werte wie Bandbreite in MB/s in einem Zahlenbereich von $2^0 - 2^{31}$ denkt. Gleichzeitig ist bei der Beschreibung solcher Präferenzen darauf zu achten, daß sie auch über verschiedene Dienstgüte-Kategorien hinweg Gruppierungen erlauben sollten. So kann eine Videokonferenz mit einer schlechten Qualität vom Klienten schlechter als eine reine Audio-Übertragung mit einer guten Qualität bewertet werden.

Ein Ansatz, den o.g. Problemen und Anforderungen zu begegnen, ist die Darstellung von Klientenpräferenzen in einer Kontrakt-Hierarchie [11], [38]. Ein Kontrakt stellt dabei eine Menge von Prädikaten dar, die gültige Zustände angeben. Die Anordnung in der Hierarchie wird durch Einschränken des durch die Prädikate vorgegeben Zustandsraums erreicht. Die Modellierung der Präferenzen geschieht nach zwei Regeln:

Regel der Präferenz: Jeder Knoten, der einen Zustand ϕ darstellt, der über Zustände $\{\psi_1, \ldots, \psi_n\}$ präferiert wird, hat in der Hierarchie eine größere Tiefe: $height(\phi) > height(\psi_i)$ für $i \in \{1 \ldots n\}$.

Regel der Indifferenz: Alle Knoten $\{\rho_1, \ldots, \rho_n\}$, die gleich präferiert werden, haben dieselbe Höhe in der Hierarchie: $height(\rho_i) = height(\rho_j)$ mit $i, j \in \{1 \ldots n\}$.

In Abbildung 4.1 werden die beiden Regeln und ihre Entsprechung in der Hierarchie illustriert. Die Modellierung der Hierarchie beginnt mit einem *Basis-Kontrakt*. Dieser spannt den Raum der möglichen Werte von Kontrakten in der Hierarchie auf. Durch Einschränken des Zustandsraums wird die Hierarchie modelliert. Jeder Kindknoten beschreibt also einen kleineren Ausschnitt des Zustandsraums des Elternknotens. Anwendungsentwickler können, ausgehend von einem vorgegebenen Basis-Kontrakt, ihre Präferenzen modellieren. Der Dienstgüte-Implementierer stellt einen entsprechenden Basis-Kontrakt und dessen Umsetzung auf die Dienstgüte-Parameter bereit. Die Beschreibung der Kontrakt-Hierarchie geschieht in QML. Durch die Umsetzung des Basis-Kontrakts auf die Dienstgüte-Repräsentation des Rahmenwerks ist eine für den Benutzer einfache Verhandlung möglich. Der Klient übergibt dem Dienst seine Präferenz. Dieser ermittelt die bestmögliche Dienstgüte-Instanz mit Bezug auf diese Präferenz und liefert einen Dienstgüte-Parameter mit der entprechenden Instanz.

Klientenpräferenzen stellen somit eine Abstraktion über den Dienstgüte-Parametern dar. Damit ist der Einbezug von weiteren Eigenschaften der Dienstgüte-Erbringung, wie der Preis eines bestimmten Dienstgüte-Niveaus, möglich. Darüber hinaus kann eine Umsetzung von anwendungsorientierten Beschreibungen der Qualität in die Dienstgüte-Parameter des Rahmenwerks für Benutzer hilfreich sein. Im folgenden Beispiel werden drei Qualitätsstufen (akzeptabel, durchschnittlich, gut) einer Videoübertragung beschrieben, die im Dienstgüte-Parameter andere Repräsentationen, wie der Rahmenrate, Farbtiefe oder Bildgröße besitzen können.

```
type video = contract {
   quality : increasing enum { acceptable, average, good }
      with order { acceptable < average, average < good };
   price : decreasing enum { low, medium, high }
      with order { low < medium, medium < high };
};
```

Der Kontrakt "video" spiegelt zwei Dimensionen einer Dienstgüte-Charateristik für die Dienstgüte-Befähigung einer Videoübertragung wider. Die

Dimension "quality" wird durch die geordnete Menge von "acceptable", "average" und "good" gebildet. Für die Dimension "price" wird eine ähnliche Menge mit einer anderen Ordnung gewählt. Die Ordnungen drücken aus, daß bei gleicher Qualität der niedrigere Preis präferiert wird. Damit ist das Tupel (`quality = average; price = average`) weniger präferiert als (`quality = average; price = low`).

Abbildung 4.2 zeigt einen aus dem obigen Vertragstyp gebildeten Präferenzbaum. Die Knoten in dem Baum entsprechen jeweils einem Tupel (`quality, price`). Die Wurzel des Baumes wird durch den Basiskontrakt gebildet, der den schlechtesten Wert der Präferenz modelliert. Durch Verfeinerung können nun Abstufungen der Klientenpräferenz gebildet werden. Dabei müssen in den verfeinerten Kontrakten nur die zusätzlichen Einschränkungen modelliert werden. Unveränderte Dimensionen geben ihre Werte an die verfeinerten Kontrakte weiter. In der Abbildung sind zwei Strategien abgebildet. Der linke Ast modelliert durchschnittliche Qualität und das Verhältnis zum Preis. So wird durchschnittliche Qualität mit niedrigem Preis über der mit mittlerem Preis präferiert. Durchschnittliche Qualität mit hohem Preis ist nicht in der Präferenz modelliert und wird somit nicht akzeptiert. Im rechten Ast ist die Präferenz bezogen auf hohe (gute) Qualität modelliert.

Ein Klient übermittelt seine so spezifizierte Präferenz an den Dienst. Der Dienst zerlegt den Baum in verschiedene Pfade entlang der Verfeinerungsäste und durchläuft die Pfade entsprechend ihrer Präferenz von den Blättern zu der Wurzel. Der Dienstgüte-Implementierer gibt den Basiskontrakt vor und implementiert eine Evaluierungsmethode, die prüft, ob ein gegebener Kontrakt erfüllbar ist. Die Evaluierung liefert den ersten erfüllbaren Kontrakt als Ergebnis. Dies entspricht aus Sicht des Klienten der bestmöglichen Dienstgüte, bezogen auf seine Präferenz.

Ein Nachteil der Offenlegung der kompletten Klientenpräferenz ist, daß Dienste ihren Klienten ein schlechteres Dienstgüte-Niveau anbieten könnten, das günstiger realisierbar ist. Dem kann dadurch begegnet werden, daß nicht initial die komplette Präferenz übertragen wird, sondern in jedem Verhandlungsschritt ein Indifferenzniveau. Innerhalb eines solchen Indifferenzniveaus kann ein Dienst solche Optimierungen vornehmen, ohne Klienten zu benachteiligen. Die recht kompakte Beschreibung der Klientenpräferenzen in QML läßt sowohl ein- wie auch mehrstufige Verhandlungen zu, ohne übermäßige Netzlast zu produzieren.

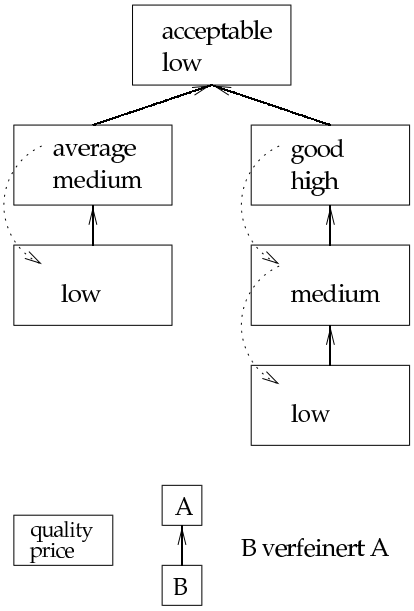


Abbildung 4.2: Präferenz-Baum

4.3.2 Infrastrukturdienste

Wie in Kapitel 3 motiviert, besteht für das Dienstgüte-Management neben der Erweiterung bestehender Dienste auch der Bedarf nach maßgeschneiderten Diensten. Da diese nicht den Schwerpunkt der Arbeit bilden, soll hier nur der Vollständigkeit halber eine Übersicht präsentiert werden.

Dienstvermittlung

Die Vermittlung von Diensten durch einen Dienstvermittler (engl. Trader, vgl. RM-ODP [43]) ist im Rahmenwerk durch einen Standard-Dienstvermittler der CORBA-Dienste [68] möglich. Die in der QIDL definierten Dienste und Dienstgüte-Charakteristiken werden in einem erweiterten Schnittstellenverzeichnis (Interface Repository) abgelegt. Ein Dienstvermittler, der dieses Schnittstellenverzeichnis verwendet, ist automatisch dazu befähigt, auf die erweiterten Beschreibungen zuzugreifen. Vermittler mit einer erweiterten Beschreibung für die Inhalte, wie der auf Konzeptgraphen beruhende AI-Trader [80], sind geeignet, auch beliebige, nicht auf einer IDL-Erweiterung beruhende Dienstgüte-Spezifikationen in den Vermittlungsvorgang zu integrieren. Allerdings ist hier auch nur der statische Teil

der Dienstgüte-Befähigung abgebildet. D.h., Klienten können Dienste aufgrund ihrer potentiellen Möglichkeit, eine bestimmte Dienstgüte-Charakteristik anzubieten, auswählen. Ob bei Zustandekommen einer Beziehung zwischen Klient und Dienst dann das vom Klienten gewünschte Dienstgüte-Niveau erbracht werden kann, muß durch eine Verhandlung herausgefunden werden. Es wäre zwar wünschenswert, bereits zum Vermittlungszeitpunkt Dienste ausschließen zu können, die eine bestimmte Mindestanforderung nicht anbieten. Dies ist jedoch nicht einfach möglich, da zum einen die Erbringung eines bestimmten Dienstgüte-Niveaus von dem jeweiligen Klienten abhängt. So kann die Netzlast unterschiedlich verteilt sein und der Dienst kann nur durch zusätzliche Informationen über den Klienten ein einhaltbares Angebot unterbreiten. Dies würde zu einer Vorverhandlung führen, die neben dem erhöhten Aufwand im Dienstvermittler, der nun einen Teil der Verhandlungskomponente enthält, auch zusätzliche Last im Netzwerk durch die Verhandlungen des Vermittlers mit den Diensten verursacht. Die Dynamik offener verteilter Systeme kann dazu führen, daß Einträge im Dienstvermittler schnell veralten, was die Vorhaltung von Qualitätszusicherungen im Vermittler erschwert. Alternativ könnte ein Dienstexport auch mit einer Gültigkeitsdauer versehen sein, die angibt, für welchen Zeitraum die angebotene Dienstgüte erbracht werden kann [53] oder einen Durchgriff auf die aktuellen Werte des Dienstes bei der Vermittlung eines Dienstes nach sich ziehen [56]. Im ersten Fall würde ein häufiger Export von Diensten das Netzwerk periodisch belasten, um aktuelle Werte im Vermittler vorzuhalten. Der zweite Fall bedeutet, daß bei einer größeren Anzahl Dienste bei der Dienstauswahl – bevor der Klient aufgrund anderer Kriterien seine Wahl einschränkt – Kommunikation mit jedem potentiellen Dienst stattfinden muß. Daher ist eine Vermittlung nur aufgrund der statischen Eigenschaften eines Dienstes und einer nachfolgenden Verhandlung zwischen Klient und Dienst sinnvoll, um aktuelle Informationen über die möglichen Dienstgüte-Niveaus anzubieten, ohne das Netzwerk unnötig zu belasten. Die Dienstvermittlung im Rahmenwerk bezieht somit nur die statischen Informationen eines Dienstes und der unterstützten Dienstgüte-Charakteristiken in den Vermittlungsvorgang ein.

Ressourcen-Steuerung

Die Erbringung eines bestimmten Dienstgüte-Niveaus ist mit der Inanspruchnahme von Ressourcen verbunden. Dabei herrscht potentiell Konkurrenz zwischen allen Klienten/Dienst-Beziehungen im System. Die reine Erbringung von "best-effort"-Dienstqualität, also ohne Verhandlung und Ressourcen-Reservierung, ist für viele Anwendungen nicht ausreichend. Daher ist eine entsprechende Vergabe von Ressourcen zu ermöglichen.

Die Realisierung durch einen Dienst liegt nahe, da die Zugriffe auf die Ressourcen-Vergabe von Klienten- wie Dienstseite erfolgen kann. Hauptaufgaben eines solchen Dienstes sind:

- **Management:** Ressourcen müssen ein- und ausgetragen werden können. Die Zuteilung von Ressourcen an eine Klienten/Dienst-Interaktion muß abfragbar sein. Dies kann für Abrechnungszwecken benutzt werden, aber auch, um Ressourcen zu befreien, die nicht ordnungsgemäß freigegeben wurden.

- **Vergaberichtlinien:** Die Vergabe von Ressourcen sollte in einer differenzierten Form erfolgen können. Dienste mit einer hohen Relevanz für das Gesamtsystem müssen benötigte Ressourcen vorgehalten bekommen. Bei der Zuteilung können wiederum Differenzierungen nach der Zeit der Nutzung und Anzahl der benötigten Ressourcen oder über den möglichen Entzug erfolgen.

Die Abrechnung von Dienstleistungen, die über eine reine Pauschalbepreisung hinausgeht, erfordert Interaktionen zwischen den von Klienten in Anspruch Ressourcen und der Abrechnungsinstanz. Bereits bei der Zuteilung von Ressourcen sind Vergaberichtlinien in Bezug auf Reservierung oder Preis einzubeziehen. In dem Netzwerkbereich des Dienstgüte-Managements finden sich unterschiedliche Varianten, die Integrated Services (RSVP) um Abrechnungsgesichtspunkte erweitern [32], [47]. Diese sind jeweils an die Ressource-Reservierungen geknüpft.

Überwachung

Eine frühzeitige Erkennung von Verletzungen des vereinbarten Dienstgüte-Niveaus kann zu einer Anpassung der Anwendung führen und somit die

Weiterarbeit erlauben. Weiterhin ist bei der Überwachung bestimmter Ressourcen auch eine Anpassung auf der Ebene der Dienstgüte-Mechanismen möglich, ohne daß die Anwendung tangiert wird. Ein Überwachungsdienst sollte konfigurierbar Objekte abfragen und einfache Überprüfungen der gelieferten Werte selbständig durchführen können. Die Benachrichtigung bei Bereichsüberschreitungen setzt zum einen die Unterstützung des Überwachungsdienstes und zum anderen ein einfaches Rahmenwerk für Callbacks voraus.

Die bei der Überwachung anfallenden Daten sollten, um unnötige Redundanz im System zu vermeiden, auch für die Abrechnung nutzbar sein. Damit muß der Abrechnungsdienst in der Lage sein, eine Vielzahl von unterschiedlichen Objekten zu überwachen. Dies kann entweder durch eine einfache Schnittstelle geschehen, die jedes Objekt anzubieten hat (LoadMonitor [36]) oder durch Ausnutzen von Plattform-spezifischen Integrationspunkten, wie Interceptors ([26], [25]).

Abrechnung

Die Abrechnung ist eng mit der Ressourcen-Steuerung verknüpft. So können zum einen preisgebundene Richtlinien für die Ressourcen-Vergabe etabliert werden und zum anderen die Abrechnung in Abhängigkeit der in Anspruch genommenen Ressourcen geschehen. Dies sichert zu, daß Klienten sich mit höherer Wahrscheinlichkeit an dem notwendigen und nicht dem verfügbaren Dienstgüte-Niveau orientieren und somit die Systemressourcen schonen [39]. Weiterhin bietet die Abrechnung die Möglichkeit, eine Statistik über die Ressourcen-Nutzung zu realisieren, die Informationen für den Ausbau des Systems liefert. Einen frühen Ansatz liefert der Accounting Service von DACNOS [40]. Dieser bettet Abrechnungsmodelle und -protokolle in eine objektorientierte Verteilungsinfrastruktur ein. Gegenüber den Abrechnungsmodellen aus dem Netzwerkbereich, die bezogen auf Ressourcen die Bepreisung etablieren, wird bei der DACNOS-Abrechnung der Service in Rechnung gestellt. Greift dieser auf zugrundeliegende Ressourcen zu, hat dies in der Preisgestaltung des Service Eingang zu finden.

Zusammenarbeit der Infrastrukturdienste

Im folgenden soll das Zusammenspiel der Dienste für Abrechnung, Ressourcen-Steuerung und Überwachung im Rahmenwerk kurz dargestellt werden.

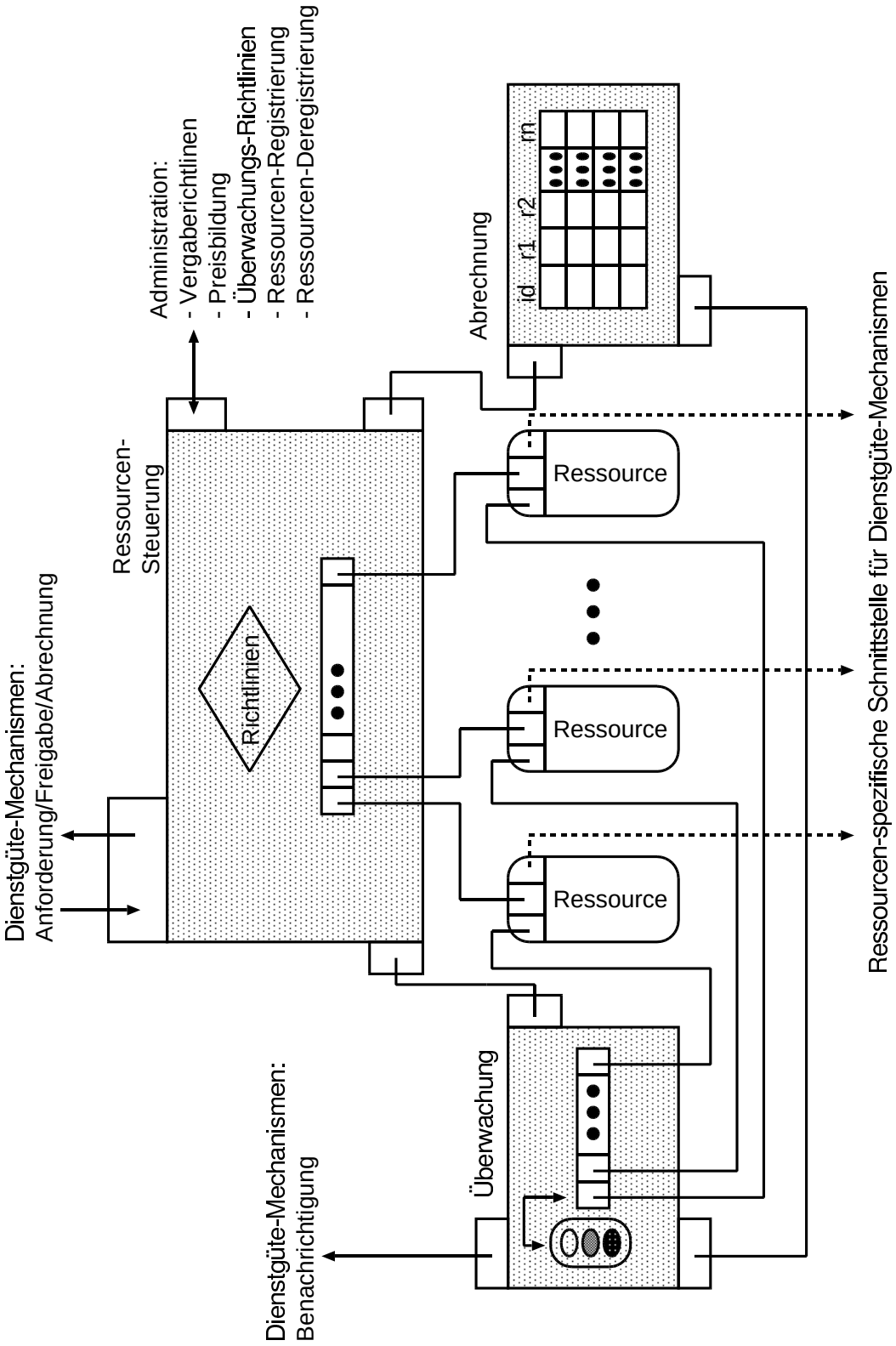


Abbildung 4.3: Überblick der Infrastrukturdienste Überwachung, Abrechnung und Ressourcen-Steuerung

Diese Dienste sind eng miteinander verknüpft. Abbildung 4.3 zeigt die schematische Interaktion zwischen den Diensten im Rahmenwerk.

Die Dienste treten nicht unmittelbar mit Anwendungsobjekten in Kontakt. Aufgabe der Dienste ist die Sicherstellung von Dienstgüte-Niveaus bei der Interaktion zwischen Klient und Dienst. Daher greifen die Dienstgüte-Mechanismen auf die Funktionalität der Dienste zu. Die Anwendung wird durch das Rahmenwerk in bestimmten Fällen, wie der Verletzung eines Dienstgüte-Niveaus, benachrichtigt.

Die Dienste sind in der Abbildung durch die drei grau unterlegten Kästen dargestellt. Die Ressourcen-Steuerung stellt im wesentlichen zwei Schnittstellen nach außen zur Verfügung:

- **Dienstgüte-Mechanismen:** Das Anfordern und Freigeben von Ressourcen wird von Dienstgüte-Mechanismen initiiert, die bestimmte Ressourcen zur Erbringung eines Dienstgüte-Niveaus benötigen. Ist die Diensterbringung beendet, werden über diese Schnittstelle die Abrechnungsinformationen an die Anwendung weitergegeben.

- **Administration:** Die Reservierung bestimmter Ressourcen für spezielle Dienste oder aber das Festlegen von Zuteilungs- bzw. Abrechnungs-Richtlinien geschieht über die Administrationsschnittstelle. Darüber hinaus werden über diese Schnittstelle dem System Ressourcen hinzugefügt bzw. entfernt.

Die Ressourcen-Steuerung nimmt Anfragen von Dienstgüte-Mechanismen entgegen und prüft aufgrund der Richtlinien, ob diese erfüllbar sind. Dazu verwaltet die Ressourcen-Steuerung alle mit der Dienstgüte-Erbringung eingetragenen Ressourcen. Ressourcen stellen dafür eine einheitliche Schnittstelle für die Verwaltung durch die Ressourcen-Steuerung zur Verfügung. Unterschiedliche Richtlinien erlauben auch den Entzug von Ressourcen, was eine Benachrichtigung der Ressource und des Dienstgüte-Mechanismus nach sich zieht. Innerhalb des Dienst-Ensembles sind die Dienste untereinander durch entsprechende Schnittstellen verknüpft. So muß die Dienstgüte-Steuerung Benachrichtigungen an Dienstgüte-Mechanismen bei Ressourcen-Entzug initiieren können und die Abrechnung informieren.

Wird eine Ressource zugeteilt, erhält der Dienstgüte-Mechanismus eine Referenz auf die entsprechende Ressource. Die Ressource stellt – neben

der Schnittstelle für die Ressourcen-Steuerung und die Überwachung – eine Ressourcen-spezifische Schnittstelle zur Verfügung. Diese Schnittstelle erlaubt den Zugriff der Dienstgüte-Mechanismen auf die Ressource. Damit sind die Ressourcen-spezifischen Funktionalitäten, wie Bandbreitenreservierung des Netzwerks oder Abarbeitungsreihenfolgen von Aufgaben für Scheduler, nutzbar.

Die Ressourcen-Steuerung erzeugt für jede Klienten/Dienst-Beziehung einen Eintrag bei der Abrechnung. Die während der Interaktion in Anspruch genommenen Ressourcen werden in der Abrechnung kumuliert und bei Beendigung in Abhängigkeit der Preisrichtlinien in Rechnung gestellt. Dabei wird eine Form von "virtueller Währung" benutzt, die in konkrete Einheiten umgerechnet werden kann.

Während der Interaktion zwischen Klient und Dienst kann durch die Überwachung die Einhaltung von Dienstgüte-Vorkehrungen der Ressourcen geprüft werden. Eine einfache Schnittstelle für die Abfrage des Zustands (gut, kritisch, verletzt) wird von jeder Ressource bereitgestellt. Die Konfiguration der Bereiche für die Zustände ist abhängig vom verhandelten Dienstgüte-Niveau und wird über die Ressourcen-Steuerung gesetzt. Bei Verletzung des Dienstgüte-Niveaus ist die Anwendung, respektive der entsprechende Dienstgüte-Mechanismus, zu verständigen. Dies geschieht über eine einfache Callback-Schnittstelle, die bei der Überwachung registriert wird.

Im weiteren Verlauf der Arbeit wird die Unterstützung des Rahmenwerks während der Entwurfs- und Implementierungsphase vertiefend dargestellt.

Kapitel 5

Spezifikation von Dienstgüte

Dieses Kapitel erläutert die Spezifikation von Dienstgüte-Charakteristiken. Nach einer Diskussion über Realisierungsalternativen und einer Bewertung wird die Schnittstellenbeschreibungssprache (Quality of Service IDL, QIDL) vorgestellt. Abschließend wird eine Umsetzung der QIDL in Zielsprachen am Beispiel einer objektorientierten Sprache (C++) präsentiert. Die IDL-Erweiterung in Kombination mit der Umsetzung realisiert einen aspektorientierten Lösungsansatz.

5.1 Alternativen der Dienstgüte-Spezifikation

Die Spezifikation von Dienstgüte-Parametern und -Mechanismen kann im verteilten Objektmodell auf unterschiedliche Art und Weise geschehen. Wir diskutieren zunächst prinzipielle Varianten [9].

5.1.1 IDL-Definitionen

Dienstgüte-Parameter und die Schnittstelle zu den Dienstgüte-Mechanismen werden in der Schnittstellenbeschreibungssprache durch adäquate Einheiten abgebildet. Dienstgüte-Parameter können so durch Verbunde (record, struct) repräsentiert werden. Dabei steht das durch die IDL angebotene Typsystem zur Verfügung. Die Übersetzung der IDL in die Zielsprache gewährleistet Sprachunabhängigkeit über Plattformgrenzen hinweg und die Nutzung des zugrundeliegenden ORBs für den Austausch der Dienstgüte-Parameter. Allerdings sind Dienstgüte-Parameter nicht von anderen Datenstrukturen unterscheidbar. Die Schnittstelle eines Dienstgüte-Mechanismus

wird durch ein Interface – wie ein Dienst auch – definiert. Die Zuweisung einer Dienstgüte-Charakteristik zu einem Dienst kann dann durch Schnittstellenvererbung geschehen.

Beipiele für eine solche Realisierung stellen der Stream-Service der OMG [73] und Quartz [88] dar, die beide auf einer Beschreibung der Dienstgüte-Parameter im Property-Service [68] beruhen. Der CORBA-Messaging Service [72] definiert Pseudo-Objekte für die Steuerung der Dienstgüte-Erbringung. Die Definition dieser Pseudo-Objekte erfolgt in IDL und die Dienstgüte-Parameter sind als Value-Types der OMG-IDL gegeben.

5.1.2 IDL-Erweiterung

Durch die Erweiterung der Schnittstellenbeschreibungssprache werden Dienstgüte-Parameter und die Schnittstellen zu den Dienstgüte-Mechanismen in entsprechenden Konstrukten mit einer Dienstgüte-bezogenen Semantik abgebildet. Dabei kann durch den IDL-Compiler zusätzlicher Code generiert werden, der sowohl Anwendungs- wie auch Dienstgüte-Entwickler unterstützt. Die Erweiterung der IDL kann auf zwei Arten geschehen:

- **implizit:** Es werden keinen neuen Schlüsselworte in der IDL eingeführt. Stattdessen werden bestimmte Bezeichner, bspw. `QoS` als Präfix von Bezeichnern, durch den IDL-Compiler gesondert behandelt. So können Dienstgüte-Spezifikationen besonders ausgezeichnet und behandelt werden.

- **explizit:** Für die Definition von Dienstgüte-Parametern und der Schnittstellen zu den Dienstgüte-Mechanismen werden neue Konstrukte in der IDL eingeführt. Die Zuweisung von Dienstgüte-Charakteristiken an Dienste kann explizit geschehen.

The Ace ORB (TAO) [85] benutzt eine implizite IDL-Erweiterung für die Darstellung der Task-Abhängigkeiten. Ein ausgezeichneter Record (struct RT_Time) trägt diese Informationen. Ein Offline-Scheduler berechnet die notwendigen Informationen, die zur Laufzeit durch den Scheduler der Laufzeitumgebung sichergestellt werden. Unser Vorschlag, die Management Architecture for Quality of Service (MAQS) [10], [7] sowie die in [23] vorgestellte Dienstgüte-Architektur erweitern die OMG-IDL um generische Dienstgüte-Spezifikationen.

5.1.3 Dienstgüte-Sprache

Die Verwendung einer eigenen Spezifikationssprache für die Dienstgüte-Parameter und ggf. die Schnittstellen der Mechanismen ermöglicht eine ähnliche Flexibilität wie die IDL-Erweiterungen. Durch die Übersetzung der Dienstgüte-Sprache entweder in die Schnittstellenbeschreibungssprache oder direkt in die Zielsprache, ist ein hoher Freiheitsgrad in der Beschreibung möglich, und gleichzeitig müssen keine Änderungen an der IDL vorgenommen werden.

Beispiele für solche Ansätze liefern die Beschreibung von Dienstgüte-Parametern im Quality-Objects-Projekt (QuO) [103] und in QML [34]. In QuO existieren zwei (ursprünglich drei) verschiedene Sprachen für die Beschreibung von Dienstgüte-Aspekten [60]. Neben der Beschreibung von Dienstgüte-Parametern können in QuO auch sog. Regionen spezifiziert werden, die Gültigkeitsbereiche der Dienstgüte-Parameter für die Laufzeit angeben. Darüber hinaus existieren spezielle Sprachen, um das Verhalten der Dienstgüte-Mechanismen zu steuern. So können Callbacks definiert werden, die aufgerufen werden, wenn sich das vereinbarte Dienstgüte-Niveau nicht mehr aufrechterhalten läßt.

Die QoS Modeling Language (QML) [34] ist eine Sprache, die für die Spezifikation von Dienstgüte-Parametern entworfen wurde. Dabei ist neben der reinen Typbeschreibung durch sog. Kontrakte eine Spezifikation von Gültigkeitsbereichen der Dienstgüte-Parameter über Profile, ähnlich der Regionen von QuO, möglich. QML ist unabhängig von einer zugrundeliegenden Architektur. Mit QRR [33] ist eine Umsetzung von QML in eine zugrundeliegenden CORBA-basierte Architektur vorgestellt worden.

5.1.4 Zielsprachen-Einbettung

Statt einer Erweiterung der Schnittstellenbeschreibungssprache oder der Verwendung dort angebotener Strukturen kann auch direkt in der Zielsprache die Spezifikation der Dienstgüte-Parameter erfolgen. Durch die fehlenden Typinformationen für das Marshalling sind solche Ansätze aber schwer generisch zu realisieren. Ein Beispiel für einen Ansatz mit Zielsprachen-Einbettung ist Electra [61], das durch eine erweiterte Schnittstelle des Objekt-Adapters und des ORBs Gruppenkommunikation erlaubt. Dienste in Objektgruppen müssen zusätzliche Methoden für die Objektserialisierung bereitstellen. Nicht objekt-orientierte Dienstgüte-Architekturen wie QoS-A

[18] verwenden häufig die Zielsprachen-Einbettung, da keine explizite Trennung zwischen Schnittstelle und Implementierung von Diensten durch eine Schnittstellenbeschreibungssprache gegeben ist.

5.1.5 Bewertung

Die hier geschilderten Integrationsansätze für die Spezifikation von Dienstgüte-Parametern und der Schnittstellen der Dienstgüte-Mechanismen spiegeln nur einen Teil der Problematik wider. So ist eine wesentliche Frage bei der Repräsentation von Dienstgüte-Parametern die der zulässigen Werte. In Programmiersprachen regeln Typsysteme, welche Werte von Daten angenommen werden können. Eine Betrachtung von Dienstgüte-Definitionen, wie sie von der ISO [44] und der ITU [96] vorgeschlagen sind, zeigt, daß für diese – hauptsächlich Verfügbarkeit und Leistungsmerkmale beschreibenden – Dienstgüte-Parameter Typsysteme herkömmlicher Programmiersprachen wie auch Schnittstellenbeschreibungssprachen ausreichend sind. Somit sind die hier genannten Integrationsstrategien gleichermaßen realisierbar. Auch komplexere Dienstgüte-Parameter, wie sie im Bereich der Zuverlässigkeit untersucht wurden [52], lassen sich mit solchen Typsystemen abbilden.

Alle Integrationsvarianten erlauben die entsprechende Repräsentation von Dienstgüte-Parametern. Die erweiterte IDL und die Dienstgüte-Sprache erlauben einen aspektorientierten Ansatz. Dies stellt keinen übermäßigen Aufwand dar, da ohnehin angepaßte Übersetzer notwendig sind. Lediglich bei der Abbildung einer Dienstgüte-Charakteristik auf die Dienstgüte-Mechanismen ist eine reine IDL-Definition mit einem deutlichen Mehraufwand verbunden. Die vom IDL-Compiler generierte Umsetzung sieht auf der Klientenseite kein benutzerdefiniertes Verhalten vor, so daß hier keine Integrationsmöglichkeit für Dienstgüte-Mechanismen ohne weitere Vorkehrungen möglich ist. Dies ist aber für die Ende-zu-Ende-Dienstgüte-Erbringung notwendig.

5.2 Bestandteile einer Dienstgüte-Spezifikation

Aufgabe einer generischen Dienstgüte-Spezifikation ist die Abbildung einer Dienstgüte-Charakteristik im System. Dabei ist der Zustand der Dienstgüte-Charakteristik wie auch die Integration der Dienstgüte-Mechanismen

durch geeignete Konstrukte abzubilden. Die Zuordnung von Dienstgüte-Charakteristiken an Dienste bzw. andere Einheiten ist durch die Dienstgüte-Spezifikation zu gewährleisten.

5.2.1 Zustand

Der Zustand einer Dienstgüte-Charakteristik wird durch die Dienstgüte-Parameter abgebildet. Eine generische Spezifikation sollte hier keine Einschränkungen durch fest vorgegebene Typen oder Restriktionen auf bestimmte Konstrukte vornehmen. Es ist bei der Wahl der Integration in die Infrastruktur darauf zu achten, daß Dienstgüte-Parameter zu verschiedenen Zwecken auch über das Netzwerk ausgetauscht werden müssen (Verhandlung, Überwachung). Um die zugrundeliegende Infrastruktur nutzen zu können, ist daher eine Definition der Dienstgüte-Parameter in IDL oder eine Übersetzung der Dienstgüte-Parameter-Definition in IDL-Konstrukte notwendig.

5.2.2 Verhalten

Das Verhalten einer Dienstgüte-Charakteristik setzt sich aus verschiedenen Facetten zusammen. Diese sind nicht alle für die Anwendung relevant, haben jedoch Einfluß auf die Spezifikation. Von den folgenden Verantwortlichkeiten ist für den Anwendungsentwickler nur die Aspekt-Integration und die Konfiguration von Belang.

- **Steuerung/Konfiguration:** Die Konfiguration des Mechanismus ist notwendig, um die Dienstgüte-Erbringung eines ausgehandelten Dienstgüte-Niveaus zu gewährleisten, wie beispielsweise durch Bandbreitenreservierung. Dabei muß entlang der Hierarchie von Dienstgüte-Mechanismen zwischen den Dienstgüte-Mechanismen kommuniziert werden. Die Schnittstelle eines Dienstgüte-Mechanismus zur Anwendung $S(M_\phi)$ bildet die entsprechende Funktionalität, damit diese die initiale Konfiguration vornehmen kann und ggf. Anpassungen auf andere Dienstgüte-Niveaus initiieren kann. Die Konfiguration kann gegenüber der Anwendung transparent erfolgen, wenn die Dienstgüte-Mechanismen der Anwendungsebene dies ohne Anwendungswissen bzw. mit den Informationen aus der Aspekt-Integration vornehmen können.

Für die Aufgaben des Managements werden Informationen der Dienstgüte-Mechanismen benötigt. Dazu können beispielsweise das Niveau der etablierten Dienstgüte-Charakteristik oder die Informationen der daran beteiligten Ressourcen angeboten werden.

- **Interne Kommunikation:** Die Etablierung eines Dienstgüte-Niveaus erfordert eine mögliche Interaktion zwischen den Dienstgüte-Mechanismen der Klienten- und der Dienstseite. Diese sollten die zugrundeliegende Infrastruktur nutzen können und müssen daher aus der Spezifikation in eine geeignete Schnittstellenbeschreibung übersetzt werden.

- **Aspekt-Integration:** Die Verantwortlichkeiten eines Dienstes gegenüber einer bestimmten Dienstgüte-Charakteristik müssen aufgrund der Aspekt-Natur der Dienstgüte explizit angegeben und behandelt werden. Die konkrete Ausgestaltung einer solchen Aspekt-Integration hängt von den Aspektsprachen und den Zielsprachen ab.

5.2.3 Zuordnung Dienstgüte/Dienst

Die Zuordnung von Dienstgüte-Charakteristiken an Dienste kann zu unterschiedlichen Zeiten und in unterschiedlichen Granularitäten erfolgen. An dieser Stelle werden unterschiedliche Varianten vorgestellt, und im nächsten Abschnitt wird diskutiert, welche Variante für generische Dienstgüte-Spezifikationen in einem verteilten Objektmodell geeignet ist.

Granularität der Zuordnung

Bei der Bewertung von Dienstgüte-Erbringung kann diese mit verschiedenen Einheiten verknüpft sein. Ein Dienst wird im verteilten Objektmodell durch seine Schnittstelle repräsentiert. Diese Schnittstelle enthält verschiedene Operationen, die mit Parametern versehen sind.

```
interface Dienst{
  long Daten();                  // aktuell
  void Buchen(in Buchungssatz); // verfügbar
};
```

Das oben stehende Beispiel zeigt einen Dienst mit zwei Operationen. Die beiden Operationen stellen jeweils andere Dienstgüte-Anforderungen. Die Daten-Operation liefert Daten, die eine bestimmte zeitliche Gültigkeit haben. Als Dienstgüte-Charakteristik kann hier die zeitliche Aktualität modelliert werden. Die Buchen-Operation soll verfügbar sein.

Bei der Zuordnung von Dienstgüte-Charakteristiken an Dienste müßte der Dienst in zwei Schnittstellen aufgeteilt werden, die jeweils die ensprechende Dienstgüte-Charakteristik der Operation zugewiesen bekommen.

Wird die Zuordnung von Dienstgüte-Charakteristiken an Operationen unterstützt, kann der oben modellierte Dienst beibehalten werden.

Die Zuordnung von Dienstgüte-Charakteristiken an Parameter von Operationen erlaubt eine noch feinere Zuordnung. So könnte die Aktualität nicht auf die Operation Daten bezogen werden, sondern auf deren Ergebnisparameter.

Zeitpunkt der Zuordnung

Die Zuordnung, welche Einheit zur Laufzeit eine bestimmte Dienstgüte-Charakteristik unterstützt, kann bereits in der Entwurfs- und Implementierungsphase erfolgen. Dazu wird in der Dienstgüte-Spezifikation die entsprechende Einheit mit einer Dienstgüte-Charakteristik verknüpft.

```
interface Dienst{
  long Daten() QOS Aktualität(0.5);
};
```

Hier wird in einer IDL-Erweiterung der Operation Daten die Dienstgüte-Charakteristik Aktualiät zugeordnet und gleichzeitig ein Wert für das Dienstgüte-Niveau – hier die maximale Verzögerungszeit – festgelegt. Bei Echtzeitanwendungen ist diese Information beispielsweise notwendig, um die Abarbeitungsreihenfolge von Aufträgen für die Laufzeit zu berechnen.

Es ist auch möglich, nur die prinzipielle Dienstgüte-Unterstützung zur Entwurfs- und Implementierungszeit festzulegen und die konkrete Ausprägung (Dienstgüte-Niveau) zur Laufzeit durch eine Verhandlung zu bestimmen. Dann würde kein Wert für das Dienstgüte-Niveau festgelegt.

Weiterhin ist die Zuordnung einer Dienstgüte-Charakteristik und die Bestimmung des Dienstgüte-Niveaus auch rein zur Laufzeit möglich. Hierzu muß das Rahmenwerk Funktionen bereitstellen, mittels der zu einer Klienten/Dienst-Beziehung ein Dienstgüte-Niveau festgelegt werden kann.

5.3 Eine generische Dienstgüte-Spezifikation

In diesem Abschnitt wird die Quality of Service IDL (QIDL) [66], eine Erweiterung der OMG-IDL um Definitionen für Dienstgüte-Schnittstellen, vorgestellt. Eine Dienstgüte-Schnittstelle spezifiziert die Dienstgüte-Parameter sowie die Schnittstellen des Dienstgüte-bezogenen Verhaltens. Zunächst wird die grundsätzliche Integration von Dienstgüte-Spezifikationen in der QIDL erläutert. Anschließend werden die Elemente der QIDL vorgestellt und die Umsetzung in die Zielsprachen präsentiert.

5.3.1 Integration

Die QIDL ist eine explizite IDL-Erweiterung. In dieser erweiterten IDL werden Konstrukte für die Definition von Dienstgüte-Parametern und die Schnittstellen zu Dienstgüte-spezifischem Verhalten eingeführt. Weiterhin ist die Zuordnung von Dienstgüte-Definitionen zu Diensten möglich.

Die explizite IDL-Erweiterung ist aus zwei Gründen gewählt worden: Vereinheitlichung der Beschreibungen eines Dienstes und Realisierung eines Ansatzes der aspektorierten Programmierung.

Vereinheitlichung der Beschreibungen

Die Schnittstelle eines Dienstes wird auch häufig als ein *Vertrag* zwischen Klient und Dienst interpretiert. Der Dienst sichert eine bestimmte Dienstleistung zu, wenn die in der Schnittstelle spezifizierten Parameter und evtl. darüber hinausgehenden Bedingungen (Vor- und Nachbedingung auf dem Zustand oder den Parametern) eingehalten werden. Die Schnittstelle kapselt die Interna des Dienstes. So können Klienten nur aufgrund der Schnittstellenbeschreibung einen Dienst in Anspruch nehmen. Im lokalen Fall ist die Schnittstellenbeschreibung durch die Klassendeklaration gegeben. Im verteilten Objektmodell wird diese explizit durch die Schnittstellenbeschreibungssprache formuliert. Die Integration der Dienstgüte-Spezifikation in die Schnittstellenbeschreibungssprache verhindert die Notwendigkeit weiterer Beschreibungssprachen. So verfügt ein Klient in einer einzigen Schnittstellenbeschreibung über die Informationen über die Dienst- und Dienstgüte-bezogenen Eigenschaften eines Dienstes.

Aspektorientierte Programmierung

Die Erweiterung der Schnittstellenbeschreibungssprache kann auch benutzt werden, um der Aspekt-Natur der Dienstgüte-Integration in verteilte Objektsysteme gerecht zu werden. Dazu ist allerdings die Einbettung in IDL-Einheiten alleine nicht ausreichend. Eine IDL-Erweiterung (implizit oder explizit) ist aus zweierlei Gründen erforderlich:

1. **Umsetzung:** Die Umsetzung der IDL in die Zielsprachen resultiert auf der Klientenseite in einem Stellvertreter-Objekt (Stub). Der Stub stellt alleine keine Möglichkeit für benutzerdefiniertes Verhalten bereit, sondern nimmt nur Aufträge an den Dienst entgegen und leitet diese weiter. Aufgrund der Ende-zu-Ende-Eigenschaft der Dienstgüte-Erbringung ist aber die Integration von Dienstgüte-spezifischem Verhalten auf der Ebene der Anwendungsobjekte auf Dienst- und Klientenseite notwendig. Um den Benutzer zu unterstützen und eine vom Rahmenwerk vorgegebene Trennung der Dienstgüte- und Anwendungsobjekte zu erreichen, ist hier eine entsprechende Code-Generierung wünschenswert.

2. **Aspekt-Integration:** Grundsätzlich ist die Integration des Dienstgüte-Verhaltens in den Ende-zu-Ende-Pfad durch Generierung entsprechender Einheiten aus der Dienstgüte-Spezifikation schon ein aspektorientierter Ansatz. Diese benötigt allerdings keine weitere Unterstützung durch den Anwendungsentwickler.

   Es existieren aber Abhängigkeiten zwischen den Anwendungsobjekten und den Dienstgüte-Mechanismen der Anwendungsebene, die nicht automatisch aufgelöst werden können, wie der Zugriff auf den Zustand von Diensten. Aus den Informationen der erweiterten Schnittstellenbeschreibung lassen sich durch eine angepaßte Umsetzung entsprechende Entwurfsmuster in der Zielsprache generieren, die eine Trennung der Implementierung solcher Aspekte unterstützt. Der Anwendungsentwickler muß für die Nutzung spezieller Dienstgüte-Charakteristiken gegebenenfalls die generierten Entwurfsmuster anpassen.

Granularität der Zuordnung

In der QIDL werden Dienstgüte-Charakteristiken durch ein eigenes Konstrukt definiert – die Dienstgüte-Schnittstelle. Die Zuordnung von Dienstgüte-Charakteristiken ist nur an Dienste möglich. Ein Dienst wird durch eine Schnittstellendefinition der OMG-IDL repräsentiert. Andere Einheiten, wie Operationen oder Parameter, wären zwar denkbar, werden aber bewußt nicht unterstützt, da die auftretenden Konflikte in keinem Verhältnis zu dem Gewinn an Aussagekraft stehen. Schnittstellen von Diensten in der IDL werden auf Objekte in den Zielsprachen umgesetzt. Dabei realisiert ein Objekt alle Operationen einer Schnittstelle. Wären für die unterschiedlichen Operationen oder gar Parameter jeweils getrennte Dienstgüte-Vereinbarungen möglich, müßten diese jeweils auf ihre Wechselwirkungen überprüft werden. Bei nicht-generischem Dienstgüte-Management ließe sich dies zur Übersetzzeit noch realisieren. Die fehlenden Informationen bei generischem Dienstgüte-Management lassen aber nur eine Überprüfung zur Laufzeit zu.

Zeitpunkt der Zuordnung

Die Erweiterung der IDL um Dienstgüte-Spezifikationen und die Zuordnung an Dienste etabliert die Bindung von Dienstgüte an Dienste schon zur Implementierungszeit. Allerdings wird in der QIDL nur die reine Befähigung eines Dienstes $< D, Q >$ vereinbart. Die Dienstgüte-Vereinbarung $[K, D, \overline{Q}]$ wird zur Laufzeit etabliert und legt fest, welche Dienstgüte-Charakteristik mit welcher Ausprägung (Dienstgüte-Niveau) geleistet wird. Sind bestimmte Vorgaben schon während der Entwurfs- und Implementierungszeit bekannt und müssen zur Laufzeit erbracht werden, kann dies durch entsprechende Ressourcen-Reservierungen sichergestellt werden.

5.3.2 Dienstgüte-Spezifikation in QIDL

Kern der Definition einer Dienstgüte-Charakteristik in QIDL ist die Dienstgüte-Schnittstelle. In QIDL wird in der Dienstgüte-Schnittstelle der Zusammenschluß des Verhaltens und der öffentlichen Struktur einer Dienstgüte-Charakteristik subsumiert. Unter der öffentlichen Struktur werden hier die Dienstgüte-Parameter verstanden. Die nicht-öffentliche Struktur, die von

der Implementierung der Dienstgüte-Charakteristik definiert wird, bleibt durch die Schnittstelle in der Implementierung verborgen.

Die Struktur einer QIDL-Dienstgüte-Definition besitzt folgenden Aufbau:

```
qos  QoS-Name                 Dienstgüte-Definition
{
    Par_1-Typ Par_1-Name;     Parameter-Definition (Zustand)
    ...
    Par_n-Typ Par_n-Name;
    interface                 Schnittstelle zum Dienstgüte-Verhalten
    {
          Operation_1-Dekl;   Operationen = Zugriff auf Verhalten
          ...
          Operation_m-Dekl;
    };
};
```

Im Anhang A.1 findet sich die Grammatik der Dienstgüte-Spezifikation in QIDL. Jede Dienstgüte-Schnittstelle wird durch das neu hinzugekommene Schlüsselwort qos eingeleitet. Damit wird ein Name an diese Definition gebunden. Die Definition besteht aus zwei Teilen:

1. **Zustandsdefinition:** In der Zustandsdefinition einer Dienstgüte-Charakteristik wird der öffentlich zugängliche Zustand vereinbart. Die Summe aller hier vereinbarten Parameter bildet als Zusammenschluß das sogenannte *Dienstgüte-Attribut*. Im Dienstgüte-Attribut sind aufgrund seiner Definition alle relevanten Parameter der Dienstgüte-Charakteristik enthalten. Jeder Parameter wird analog zu Elementen eines Records (struct) gebildet. In der Parameter-Definition steht zuerst der Typ des Parameters und dann der Name, unter dem der Parameter im Attribut erreichbar ist. Als Typen sind alle in der OMG-IDL zugelassenen Typen – auch benutzerdefinierte – erlaubt.

   Das Dienstgüte-Attribut wird in eine netzwerkweit übertragbare Struktur übersetzt. Näheres zu der Umsetzung folgt in Abschnitt 5.4.

2. **Schnittstellendefinition:** Das Verhalten einer Dienstgüte beinhaltet verschiedene Funktionalitäten: Management, Aspekt-Integration, Steuerung und interne Kommunikation der Dienstgüte-Mechanismen.

   Eine Differenzierung der Schnittstellen-Kategorien wäre im Sinne einer besseren Strukturierung wünschenswert. Da für den Dienst, respektive den Klienten, nur die Aspekt-Integration sowie die Steuerungs-Operation von Belang sind, könnte man diese durch spezielle Schlüsselwörter abheben. Um die Änderungen an der OMG-IDL klein zu halten, wurde hierauf im Prototypen verzichtet. Die bislang implementierten Dienstgüte-Charakteristiken benötigten Dienstgüte-Schnittstellen mit geringem Umfang, so daß eine weitere Strukturierung nicht zwingend erforderlich scheint.

5.3.3 Zuordnung Dienstgüte/Dienst

Einem Dienst kann eine beliebige Anzahl von Dienstgüte-Charakteristiken zugeordnet werden. Zur Laufzeit kann aber jeweils nur eine davon aktiv sein (vgl. Abschnitt 5.3.4). Diese wird durch eine Verhandlung zwischen Klient und Dienst bestimmt. In einem generischen Rahmenwerk stellt dies, wie bereits gezeigt, keine Einschränkung dar. Wird eine Kombination aus mehreren Dienstgüte-Charakteristiken benötigt, muß eine Schnittstelle dafür definiert und implementiert werden. Dabei kann in Grenzen auf die Implementierung bereits vorhandener Dienstgüte-Charakteristiken zurückgegriffen werden.

```
interface  Dienst-Name                    Dienst-Definition
        : Basis-Schnittstelle_1
        , ... , Basis-Schnittstelle_n     Schnittstellenvererbung
        withQoS  DG-Name_1
        , ... , DG-Name_m                 Dienstgüte-Zuordnung
{
         ...                              Operationen des Dienstes
};
```

Der Dienst "Dienst-Name" ist durch die Operationen der Basis-Schnittstellen und die innerhalb der Schnittstellendefinition vereinbarten Opera-

tionen definiert. Das neu eingeführte Schlüsselwort `withQoS` dient zur Zuordung von Dienstgüte-Charakteristiken an eine Dienstschnittstelle. Dabei folgt `withQoS` der Vererbungsdeklaration und kann eine beliebige Folge von Dienstgüte-Charakteristiken dem Dienst zuordnen. Eine Dienstgüte-Charakteristik ist hier durch den Namen der Dienstgüte-Schnittstelle gegeben.

5.3.4 Vererbung von Dienstgüte-Schnittstellen

Die Zuordnung mehrerer Dienstgüte-Charakteristiken an einen Dienst, von denen zur Laufzeit jeweils nur eine aktiv sein kann, stellt eine Einschränkung dar. Wird von einer Anwendung die Kombination mehrerer bestehender Dienstgüten verlangt, ist dies auf diese Weise schwer möglich. Ohne Kenntnisse der zugrundeliegenden Implementierung ist eine automatisierte Zusammenführung verschiedener Dienstgüte-Charakteristiken aber nicht sinnvoll zu realisieren.

Als Beispiel seien hier zwei Dienstgüte-Charakteristiken gegeben. Eine Dienstgüte "Verfügbarkeit", die auf einer Replikatgruppe als Mechanismus aufbaut, und eine Dienstgüte "Vertraulichkeit", die auf Verschlüsselung der übertragenen Daten beruht. Ohne genaue Kenntnisse der Implementierung der Dienstgüte-Charakteristiken kann eine kombinierte Dienstgüte "verfügbare Vertraulichkeit" nicht realisiert werden. Bei einer rein sequentiellen Ausführung der Dienstgüte-Mechanismen könnte der Aufruf des Multicast-Transports für die Verfügbarkeit vor der Verschlüsselung geschehen. Somit würden die Daten unverschlüsselt an die Replikate übergeben. Je nach Implementierung des Dienstgüte-Mechanismus für "Vertraulichkeit" könnte die Sequenz in anderer Reihenfolge das gewünschte Resultat erzielen. Im allgemeinen ist die automatisierte Zusammenführung von Dienstgüte-Mechanismen zu kombinierten Dienstgüte-Mechanismen wenig sinnvoll.

Hingegen ist auf der Schnittstellenebene ein Bezug zwischen Dienstgüte-Charakteristiken zur Klassifizierung durchaus sinnvoll. Analog zu der Schnittstellenvererbung in OMG-IDL, die keinen direkten Bezug zur Implementierung aufweist, kann so die Struktur von Dienstgüte-Charakteristiken durch Vererbung auf der Schnittstellenebene ausgedrückt werden. QIDL bietet die Möglichkeit, Schnittstellenvererbung für Dienstgüte-Charakteristiken zu vereinbaren. Die Semantik der Vererbung ist auf der Schnittstellenebene definiert. Aufgrund der oben geschilderten Probleme wird durch

die Umsetzung keine Beziehung der Dienstgüte-Mechanismen auf der Implementierungsebene erzeugt.

```
qos  QoS-A                              Dienstgüte A Definition
{
   Parameter P1;                        Parameter Definition (Zustand)
   interface                            Schnittstelle zum Dienstgüte-Verhalten
   {
         Operation O1;                  Operationen = Zugriff auf Verhalten
   };
};
qos  QoS-B                              Dienstgüte B Definition
{
   Parameter P2;                        Parameter Definition (Zustand)
   interface                            Schnittstelle zum Dienstgüte-Verhalten
   {
         Operation O2;                  Operationen = Zugriff auf Verhalten
   };
};
qos   QoS-C extends QoS-A, QoS-B        Dienstgüte C Definition
{
   Parameter P3;                        Parameter Definition (Zustand)
   interface                            Schnittstelle zum Dienstgüte-Verhalten
   {
         Operation O3;                  Operationen = Zugriff auf Verhalten
   };
};
```

Im oben stehenden Beispiel werden drei Dienstgüte-Charakteristiken (`QoS-A`, `QoS-B` und `QoS-C`) vereinbart. `QoS-A` und `QoS-B` folgen dem schon vorgestellten Schema einer Dienstgüte-Definition und fassen Zustand und Verhalten der Dienstgüte-Charakteristiken durch Parameter und Operationen zusammen. Die Dienstgüte-Charakteristik `QoS-C` stellt eine Erweiterung der Dienstgüte-Charakteristiken `QoS-A` und `QoS-B` dar. Dies wird durch das neu eingeführte Schlüsselwort `extends` ausgedrückt. Die Semantik der Vererbung erfolgt in einer Aggregation des Zustands und des Verhaltens der Basis-Dienstgüte-Charakteristiken in der Sub-Dienstgüte. Dabei ist zu beachten, daß Namenskonflikte nicht auftreten dürfen. Überladungen von Operationen und die daraus mögliche Polymorphie des Verhaltens

ist nicht zulässig. Die Auswirkung auf die Typisierung von Diensten und insbesondere die Subtyp-Relation bei Diensten mit zugeordneten Dienstgüten findet sich in Abschnitt 5.6.

5.3.5 Beispiel

An dieser Stelle soll ein einfaches Beispiel die Nutzung von QIDL zur Spezifikation von Dienstgüte-Charakteristiken auf der Schnittstellenebene verdeutlichen. Der nächste Abschnitt beschreibt dann die Umsetzung der QIDL-Definitionen in Einheiten der Zielsprache.

```
1: qos reliable
   {
2:   short server_num;

     interface {
3:     short act_group_size();

4:     string get_adr();
5:     void enter_ref(in string ref);

6:     void put_state(in any s);
       any get_state();
     };
   };

   interface naming
   {
     void bind(in string name1, in string name2);
     string resolve(in string name);
     void unbind(in string name);
   };

7: interface reliable_naming : naming withQoS reliable
   {
   };
```

Im vorliegenden Beispiel wird die Dienstgüte-Charakteristik `reliable` zur Sicherstellung der Verfügbarkeit einem Namensdienst zugeordnet.

1. Vereinbarung der Dienstgüte "reliable". Diese basiert auf einem Dienstgüte-Mechanismus, der eine Replikatgruppe benutzt.

2. Der Zustand der Dienstgüte-Charakteristik wird durch die Anzahl der Replikate in der Gruppe repräsentiert.

3. Zur Überwachung kann die Anzahl der Replikate in der Gruppe bestimmt werden.

4. `get_adr` ist eine Operation, die für die interne Abstimmung der Dienstgüte-Implementierung auf Dienst- und Klientenseite benutzt wird. Die Klient-Dienstgüte-Implementierung ruft auf der Dienstgüte-Implementierung der Dienstseite `get_adr` auf, um die Gruppenkommunikationsadresse zu erhalten.

5. Bei `enter_ref` wird der Dienstgüte-Implementierung auf der Klientenseite die Objektreferenz eines neuen Replikats übergeben.

6. Die Operationen `put_state` und `get_state` dienen zur Aspektintegration der nicht-funktionalen und funktionalen Aspekte. Beide Operationen stellen Verantwortlichkeiten des Dienstes gegenüber der Dienstgüte-Implementierung dar. Durch eine entsprechende Umsetzung müssen diese Operationen im Dienst implementiert werden. Dies wird in Abschnitt 5.4 genauer erläutert.

7. Zuordnung der Dienstgüte `reliable` an die Schnittstelle `reliable_naming`. Durch Schnittstellenvererbung von `naming` erhält die Schnittstelle das notwendige Verhalten des Namensdienstes. Durch die Vererbung bleibt `naming` als Dienst ohne Dienstgüte-Befähigung erhalten.

5.4 Generierung von Vorlagen

Aus den zuvor beschriebenen Erweiterungen der OMG-IDL für die Spezifikation von Dienstgüte-Charakteristiken müssen analog zu den Schnittstellen von Diensten Vorlagen für die Implementierung erzeugt werden. Die

Codegenerierung realisiert hierbei einen aspektorientierten Ansatz, indem Anwendungs- und Dienstgüte-spezifisches Verhalten durch generierte Entwurfsmuster getrennt wird. Zunächst werden die Ziele und Voraussetzungen der Umsetzung von QIDL in eine Zielsprache skizziert und danach die Umsetzung der Struktur und des Verhaltens der Dienstgüte-Schnittstelle beschrieben.

5.4.1 Ziele und Voraussetzungen

Die Umsetzung der QIDL basiert auf einem zugrunde liegenden verteilten Objektmodell. Dieses ist an die OMA angelehnt. Aufgrund der Ähnlichkeit anderer Ansätze lassen sich die hier gewonnen Ergebnisse auch auf andere Plattformen mit einem ähnlichen Objektmodell, wie DCOM oder Java/RMI, anwenden. Als Zielsprache soll hier eine objektorientierte Programmiersprache angenommen werden. Auch wenn – ähnlich wie bei den Umsetzungen der OMG-IDL nach C oder COBOL – Umsetzungen in nichtobjektorientierte Sprachen denkbar wären, ist dies nicht Gegenstand der vorliegenden Arbeit.

Wie schon in Kapitel 4 ausgeführt, sollte die Umsetzung der Dienstgüte-Spezifikation die Trennung der Verantwortlichkeiten von Anwendung und Dienstgüte-Vorkehrungen unterstützen. Für die Kommunikation der Dienstgüte-Mechanismen und der beteiligten Einheiten des Rahmenwerks sollte die zugrundeliegende Verteilungsinfrastruktur benutzt werden können.

Die durch das Rahmenwerk MAQS vorgegebene Bindung von Diensten und Dienstgüte-Charakteristiken impliziert ein weiteres Ziel der Umsetzung: die Auswahl einer bestimmten Dienstgüte-Charakteristik zur Laufzeit soll dazu führen, daß alle anderen dem Dienst zugeordneten Dienstgüte-Charakteristiken ausgeblendet werden. Abbildung 5.1 zeigt eine Zuordnung von Dienstgüte-Schnittstellen $\varphi_1, \ldots, \varphi_n$ an einen Dienst D. Dies entspricht der Dienstgüte-Befähigung $< D, Q >$ mit $Q = \{\varphi_1, \ldots, \varphi_n\}$. Diese Befähigung kann zur Laufzeit durch eine Verhandlung instantiiert werden. Als Resultat wird eine Dienstgüte-Vereinbarung $[K, D, \overline{\varphi_i}]$ erzielt.

Ist eine Dienst/Klienten-Beziehung durch eine solche Dienstgüte-Vereinbarung $[K, D, \overline{\varphi}]$ gebunden, sollten alle Einheiten durch Zugriff auf den Dienst – durch die Objektreferenz – auch Zugriff auf die aktuelle Dienstgüte-Charakteristik φ haben. Dies bedeutet aber nichts anderes, als den Zu-

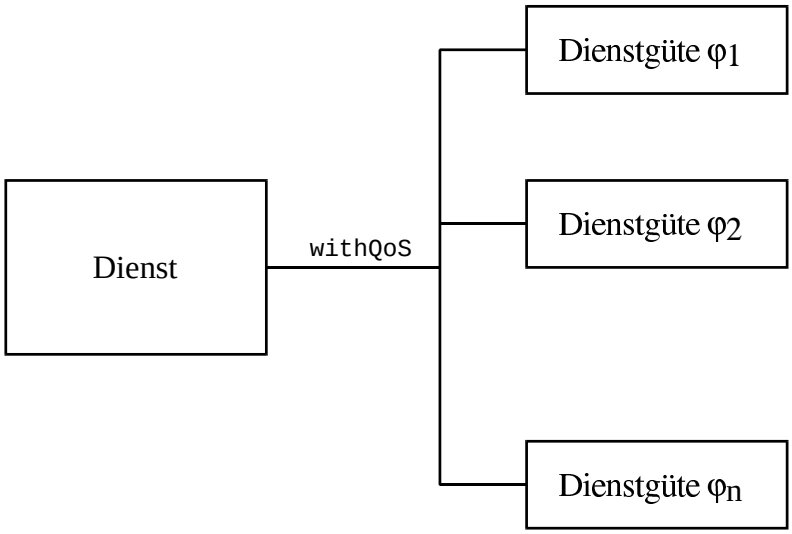


Abbildung 5.1: Dienst- und Dienstgüte-Schnittstellen in QIDL

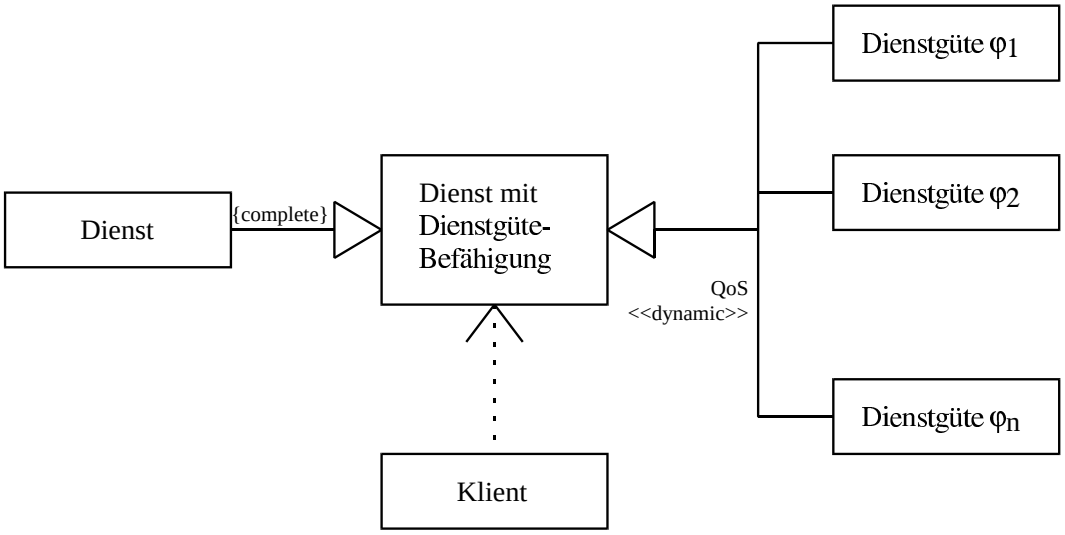
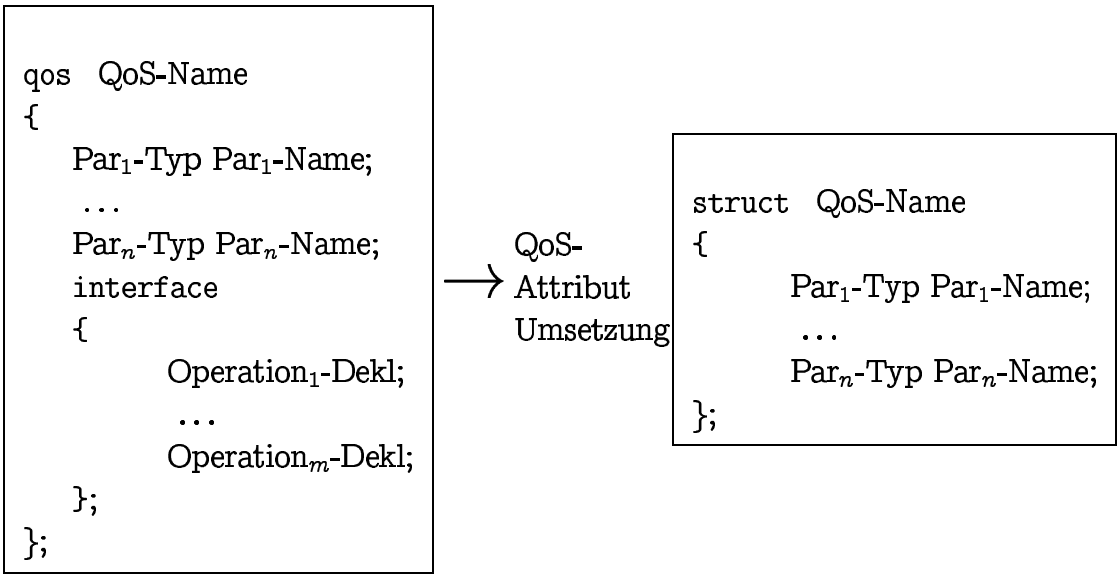


Abbildung 5.2: Dienst- und Dienstgüte-Schnittstellen zur Laufzeit

griff auf die Schnittstelle $S(M_{\varphi_i})$ des zugehörigen Dienstgüte-Mechanismus M_{φ_i} zu haben. In Abbildung 5.2 ist dieser Sachverhalt graphisch dargestellt. Über die Objektreferenz des Dienstes kann nicht nur das Dienstverhalten initiiert werden, sondern die aktuell vereinbarte Dienstgüte-Charakteristik – in diesem Fall $\varphi_i \in \{\varphi_1, \ldots \varphi_n\}$ – kann zu Zwecken der Überwachung oder Steuerung von anderen Objekten erreicht werden. Alle anderen Dienstgüte-Charakteristiken, die von einem Dienst angeboten werden, sollten nicht erreichbar sein. In der UML-Notation der Abbildung wird dies dadurch deutlich gemacht, daß der Klient von der Klasse des Dienstgüte-befähigten Dienstes abhängig ist. Diese Klasse erfüllt immer die Dienst-Rolle. Weiterhin kann diese Klasse genau eine Rolle der Dienstgüte-Schnittstellen zusätzlich erfüllen.

5.4.2 Umsetzen der Dienstgüte-Parameter

Innerhalb einer QIDL-Dienstgüte-Schnittstelle werden Zustand und Verhalten einer Dienstgüte-Charakteristik vereinbart. Dabei spiegelt der Zustand die relevanten öffentlichen Informationen einer Dienstgüte-Charakteristik wider. Diese entsprechen den Dienstgüte-Parametern. Eine naheliegende Umsetzung kann durch die folgende Umsetzung in eine IDL-Struktur geschehen.



Die Dienstgüte-Schnittstelle mit dem Namen "QoS-Name" wird in eine Struktur der OMG-IDL mit demselben Namen übersetzt. Dabei sind alle Dienstgüte-Parameter der Dienstgüte-Schnittstelle und ihre Typen in der IDL-Struktur entsprechend definiert. Für den Anwendungsentwickler stellt sich die generierte IDL-Struktur in der Zielsprache wie eine gewöhnliche IDL-Struktur dar, d.h. diese trägt Typinformationen, mit denen sie durch die Verteilungsinfrastruktur übertragen werden kann. Im Rahmenwerk wird die Bezeichnung *Dienstgüte-Attribut* für die so in die Zielsprache eingebetteten Dienstgüte-Parameter verwendet.

Die in QIDL erlaubte Schnittstellenvererbung von Dienstgüte-Schnittstellen wird für Dienstgüte-Attribute durch die Aggregation der Dienstgüte-Parameter der Basis-Dienstgüte-Schnittstellen umgesetzt. Daher sind gleiche Namen in Basis-Dienstgüte-Schnittstellen auch untersagt, da die resultierenden Namenskonflikte bei der Umsetzung schwer auflösbar wären. Das Beispiel aus dem vorangegangenen Abschnitt wird in folgendes Dienstgüte-Attribut übersetzt:

```
struct  QoS-C
{
        Par1-Typ P1;
        Par2-Typ P2;
        Par3-Typ P3;
};
```

5.4.3 Umsetzen des Dienstgüte-Verhaltens

Die Umsetzung des Dienstgüte-Zustands in Strukturen der OMG-IDL suggeriert, daß eine Umsetzung des Dienstgüte-Verhaltens in OMG-IDL-Konstrukte, wie Interfaces, auch möglich wäre. Dies ist leider nicht der Fall, da die Verwebung von Dienstgüte-Mechanismen mit Diensten besonderer Vorkehrungen bedarf und die Umsetzung durch Schnittstellenvererbung zwei für die Dienstgüte-Erbringung wichtige Einschränkungen nach sich zieht. Zum einen wird durch die Generierung des Stub-Objektes auf der Klientenseite die Integration von benutzerdefiniertem Code für das Dienstgüte-Management nicht unterstützt, und zum anderen wären durch eine Schnitt-

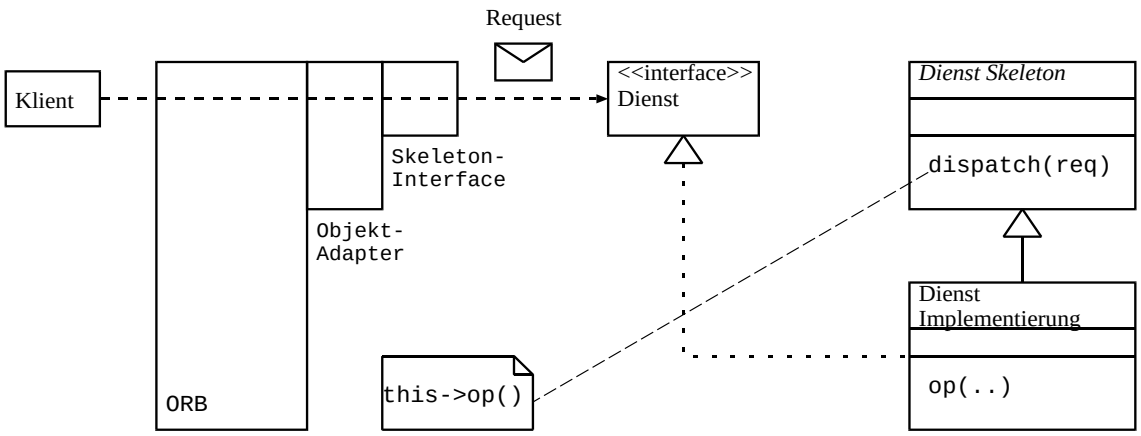


Abbildung 5.3: Dienst und Skeleton

stellenvererbung von Dienstgüte-Schnittstellen an Dienste alle Operationen der verschiedenen Dienstgüte-Charakteristiken zur Laufzeit erreichbar und nicht nur die der aktuell ausgehandelten. Schwerer wiegt jedoch, daß vor dem Eintreffen des Aufrufs bei der Zielmethode des Dienstes kein Dienstgüte-spezifisches Verhalten auf der Anwendungsebene integriert werden kann. Somit ist eine gesonderte Umsetzung der Dienstgüte-Schnittstelle erforderlich.

Im weiteren Verlauf dieses Abschnitts wird die Umsetzung des Verhaltensanteils einer Dienstgüte-Schnittstelle vorgestellt. Dabei wird zunächst der Fall mit einer dem Dienst zugeordneten Dienstgüte-Charakteristik $< D, \varphi >$ betrachtet. Die Umsetzung wird für die Dienst- und danach für die Klientenseite beschrieben. Anschließend wird die Unterstützung für die Auswahl einer Dienstgüte bei mehreren zugeordneten Dienstgüte-Schnittstellen für Klienten- und Dienstseite präsentiert.

Dienstseite

Abbildung 5.3 zeigt die Umsetzung einer Dienst-Schnittstelle in eine objektorientierte Zielsprache. Ein Auftrag eines Klienten (Request) wird an den ORB des Dienstes geschickt. Der Auftrag entspricht dem Aufruf einer Methode $C \rightarrow \mathit{Dienst.op}()$. Innerhalb des Auftrags befindet sich die Objektreferenz des Diensts sowie der Name der Methode und eventuelle Parameter. Der Objekt-Adapter nimmt den Auftrag vom ORB an und leitet diesen über das Skeleton-Interface an die Dienstimplementierung weiter.

Das Skeleton fungiert als das Bindeglied zwischen ORB, Objekt-Adapter und der Dienstimplementierung. Aus der Schnittstellenbeschreibung des Dienstes wird vom IDL-Compiler das Skeleton generiert. Dieses stellt eine abstrakte Basisklasse der Dienstimplementierung dar. Dabei ist die gesamte Funktionalität der Anmeldung und Kommunikation mit der Verteilungsinfrastruktur außer der reinen Dienstimplementierung im Skeleton realisiert. Der Dienstimplementierer erstellt eine von dem Skeleton abgeleitete Klasse und implementiert dort die Methoden des Dienstes. Bei Instantiierung der Dienstklasse wird durch die Skeleton-Basisklasse der Dienst beim Objekt-Adapter angemeldet. Eingehende Nachrichten von Klienten werden durch das Skeleton-Interface an die `dispatch`-Methode des Skeletons weitergeleitet. In dieser Methode werden der Operationsname und die Parameter aus dem Auftrag extrahiert und an die entsprechende Methode des Dienstes delegiert. Diese Struktur realisiert ein Object-Adapter-Entwurfsmuster [35].

Für die Umsetzung einer Schnittstellenbeschreibung einer Dienstgüte-Befähigung $< D, \varphi >$ in QIDL in eine objektorientierte Programmiersprache soll gewährleistet werden, daß

- Dienst und Dienstgüte-Charakteristik unter derselben Objektreferenz erreichbar sind,
- Dienst- und Dienstgüte-Implementierung separat implementiert werden können,
- die Abhängigkeiten zwischen Dienst-Aspekt und Dienstgüte-Aspekt automatisch zusammengeführt werden und
- das Dienstgüte-Verhalten vor und nach jedem Aufruf einer Operation des Dienstes eingreifen kann.

Abbildung 5.4 zeigt die von der QIDL-Umsetzung erzeugte Struktur auf der Dienstseite. Neben der Ableitung zwischen Dienst-Skeleton und Dienst-Implementierung, wie sie in Abbildung 5.3 eingeführt wurde, kommen ein Skeleton und eine Implementierung für die Dienstgüte-Schnittstelle hinzu. Das Dienstgüte-Skeleton (QoS-Skel.) wird wie ein Dienst-Skeleton aus den Definitionen der Dienstgüte-Schnittstelle generiert. Die Dienstgüte-Implementierung (QoS-Impl.) ist von diesem Skeleton abgeleitet und realisiert alle Operationen der Dienstgüte-Schnittstelle bis auf die Operationen, die mit den Dienstgüte-Mechanismen der Klientenseite und der

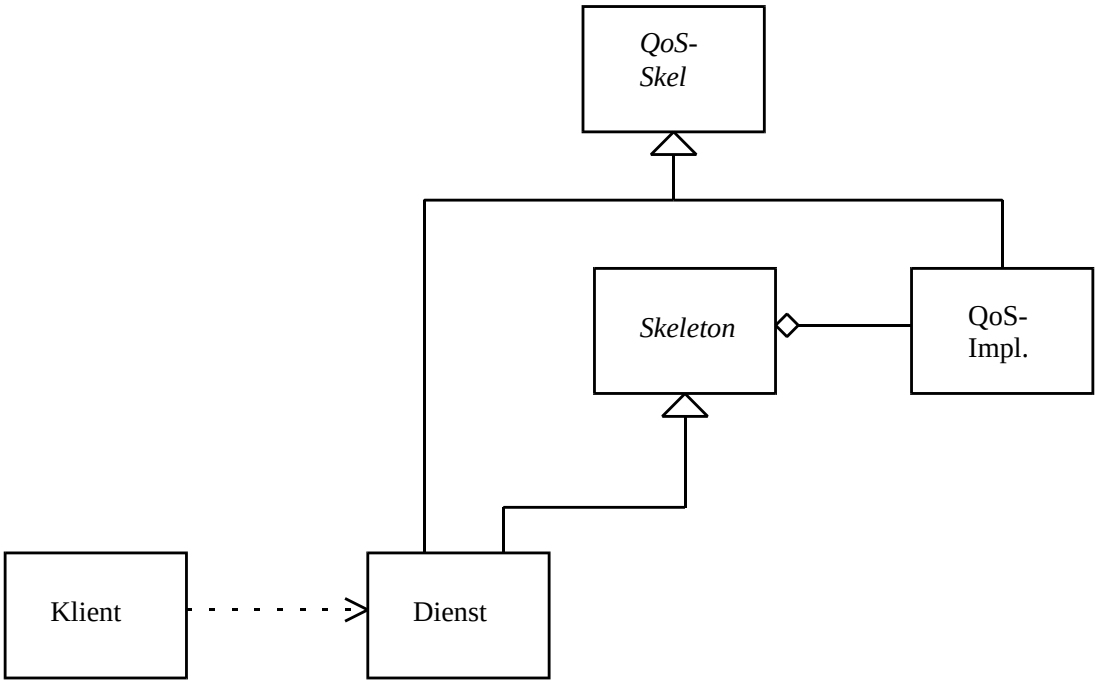


Abbildung 5.4: Dienst und Skeleton mit Dienstgüte-Befähigung

Aspekt-Integration auf der Dienstseite assoziiert sind. Das Dienst-Skeleton hält die Dienstgüte-Implementierung als Aggregrat. Der Dienst erbt nur von dem Dienstgüte-Skeleton. Das bedeutet, die Dienstgüte-Funktionalität ist nur durch das Dienst-Skeleton erreichbar. Der Dienst unterstützt aber die Dienstgüte-Schnittstelle, da er von dem Dienstgüte-Skeleton abgeleitet ist. Somit ist durch diese Struktur die Trennung zwischen der Dienst- und der Dienstgüte-Implementierung sowie die gemeinsame Erreichbarkeit durch dieselbe Objektreferenz gewährleistet. Es ist nun aber dafür Sorge zu tragen, daß vor jedem Aufruf Dienstgüte-spezifisches Verhalten aufgerufen wird, die Steuerungs- und Management-Funktionalität der Dienstgüte-Charakteristik von anderen Objekten erreicht werden kann und die Aspekt-Integration in geeigneter Art und Weise realisiert wird. Dabei ist folgendes zu beachten:

- **Dienstgüte-befähigte Operationen des Dienstes:** Nach dem Erzielen einer Dienstgüte-Vereinbarung zwischen Dienst und Klient sollen alle Interaktionen mit dem vereinbarten Dienstgüte-Niveau durchgeführt werden. Dies setzt die Steuerung der Interaktion durch die

Dienstgüte-Mechanismen voraus. Die Dienstgüte-Mechanismen auf der Anwendungsebene greifen auf den Auftrag zu, bevor dieser an die Dienstimplementierung weitergereicht wird. Nach der Dienstausführung muß eventuell für den Transport des Ergebnisses vom Dienst zum Klienten wiederum Dienstgüte-Verhalten auf den Auftrag zugreifen. Als Beispiel sei hier eine Dienstgüte-Charakteristik für "Vertraulichkeit" genannt, die Parameter verschlüsselt. Dies muß auch für out-Parameter und den Rückgabewert von Operationen geschehen. Somit ist ein Prolog und ein Epilog von Dienstgüte-Verhalten für jeden Operationsaufruf zu realisieren.

- **Dienstgüte-Verhalten:** Das Dienstgüte-Verhalten der Dienstseite dient der Überwachung, Steuerung und Konfiguration der Dienstgüte-Mechanismen und auch der Abstimmung zwischen den Dienstgüte-Mechanismen der Klienten- und Dienstseite, wie `get_adr` im Beispiel für "Verfügbarkeit" (vgl. Seite 99).

- **Aspekt-Integration:** Die Aspekte (Dienst/Dienstgüte) sind auf zweierlei Art und Weise miteinander verwoben. Zum einen ist das Einfügen von Dienstgüte-Verhalten in den Kommunikationspfad zwischen Klient und Dienst schon eine Form der aspektorientierten Programmierung. Dies kann automatisch aus den QIDL-Beschreibungen generiert werden. Darüber hinaus existieren aber auch anwendungsspezifische Abhängigkeiten zwischen Dienst und Dienstgüte-Charakteristik. Als Beispiel sei hier wieder auf die Dienstgüte-Charakteristik "Verfügbarkeit" mit dem Austausch des Dienstzustands (`get_state`, `put_state`) verwiesen. Ein weiteres Beispiel ist eine Dienstgüte-Charakteristik, die Antwortzeiten durch Lastbalancierung auf einer Server-Gruppe reduziert. Die Last der einzelnen Server ist abhängig von Server-spezifischen Faktoren (Länge der Warteschlage, CPU-Last, etc.). Somit sollte die Lastinformation durch den Server geliefert werden. Dazu notwendig ist eine definierte Schnittstelle, mit der die Dienstgüte-Charakteristik Funktionalität festlegen kann, die ein Dienst zu erbringen hat.

Abbildung 5.5 zeigt, wie das Dienstgüte-Verhalten in den Kommunikationspfad zwischen Klient und Dienst auf der Dienstseite integriert ist. Die drei beteiligten Einheiten (Dienst-Skeleton, Dienstgüte-Implementie-

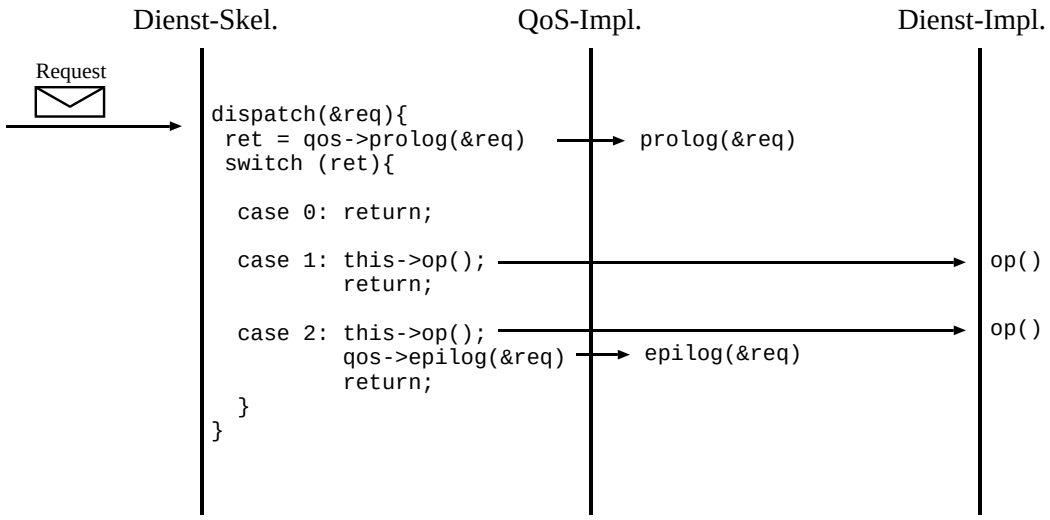


Abbildung 5.5: Dienstgüte-Verhalten bei einem Dienstaufruf

rung und Dienst-Implementierung) entsprechen denen aus Abbildung 5.4. Der Auftrag des Klienten wird im Dienst-Skeleton der `dispatch`-Methode übergeben. Diese ruft auf der aggregierten Dienstgüte-Implementierung die Methode `prolog` mit dem Auftrag als Parameter auf. Diese Methode wird vom Dienstgüte-Implementierer bereitgestellt. Durch den Rückgabewert bestimmt die `prolog`-Methode die weitere Verarbeitung des Auftrags:

- **Rückgabewert `PROLOG_ONLY` (0):** Die weitere Verarbeitung des Auftrags endet nach der Verarbeitung des Prologs. Das Dienst-Skeleton ruft keine weiteren Methoden des Dienstes auf, sondern übergibt die Kontrolle wieder an den ORB, der den Auftrag als bearbeitet an den Klienten sendet. Dies ist für Methoden der Steuerung und des Managements der Dienstgüte-Implementierung notwendig und erlaubt interne Kommunikation der Dienstgüte-Mechanismen. Aus dem Auftrag kann die `prolog`-Methode die entsprechende Operation der Dienstgüte-Schnittstelle extrahieren und bearbeiten. Da die `prolog`-Methode vor der Dienst-Implementierung aufgerufen wird, kann so verhindert werden, daß im Dienst Operationen der Dienstgüte-Schnittstelle aufgerufen werden. Dies gilt mit Ausnahme der Aspekt-Integration, die später behandelt wird.

- **Rückgabewert `PROLOG_OP` (1):** Im Gegensatz zu dem vorangegangenen Verhalten, bei dem nur Verhalten der Dienstgüte-Imple-

mentierung initiiert wurde, wird nach Bearbeitung des Dienstgüte-Prologs die Operation des Dienstes aufgerufen (`this->op()`). Dabei hat die Dienstgüte-Implementierung die Möglichkeit, auf den Auftrag zuzugreifen und geeignete Maßnahmen zu treffen, wie beispielsweise die Entschlüsselung kodierter Daten. Nach der Bearbeitung der Operation durch den Dienst wird die Kontrolle an den ORB zurückgegeben. Bei dieser Variante kann die Aspekt-Integration zwischen Dienstgüte und Dienst realisiert werden. Im Gegensatz zu den anderen Operationen der Dienstgüte-Schnittstelle werden die Operationen der Aspekt-Integration nicht von der Dienstgüte-Implementierung bearbeitet, sondern durch die Steuerung der Verarbeitung an die Dienst-Implementierung weitergeleitet. Diese muß dann die Operation bearbeiten. Es wird kein weiteres Dienstgüte-Verhalten nach der Bearbeitung der Operation durchgeführt.

- **Rückgabewert** `PROLOG_OP_EPILOG` **(2):** Die Verarbeitung bei Rückgabewert 2 entspricht der von Rückgabewert 1, nur wird zusätzlich die `epilog`-Methode der Dienstgüte-Implementierung aufgerufen. Dies erlaubt die Steuerung von Dienstgüte für die Übermittlung des Resultats an den Klienten.

Die vom QIDL-Compiler erzeugte Klassenstruktur, wie sie in Abbildung 5.4 abgebildet ist, sichert die Entkopplung von Dienstgüte- und Dienst-Aspekten zu. Bis auf die Realisierung der Aspekt-Integration erfolgt die Einbettung der Dienstgüte-Befähigung für den Dienstimplementierer transparent. Wesentlich hierfür ist die Steuerung der Verarbeitung eines Auftrages durch die erweiterte `dispatch`-Methode.

Aus den Informationen der QIDL-Beschreibung lassen sich durch den QIDL-Compiler auch Vorlagen für die `prolog`- und `epilog`-Methode der Dienstgüte-Implementierung generieren, so daß der Dienstgüte-Implementierer auch Unterstützung erfährt. Die einzelnen Implementierungen (Dienst, Dienstgüte) können einzeln erstellt und übersetzt werden. Durch den Binder werden sie zu einem ausführbaren Programm zusammengeführt.

Im folgenden wird die Umsetzung und Integration von Dienstgüte-Befähigung auf der Klientenseite vorgestellt. Zunächst wird der Fall der Dienstgüte-Befähigung mit genau einer Dienstgüte-Charakteristik betrachtet. Anschließend werden notwendige Erweiterungen für die Unterstützung und Auswahl mehrerer Dienstgüte-Charakteristiken präsentiert.

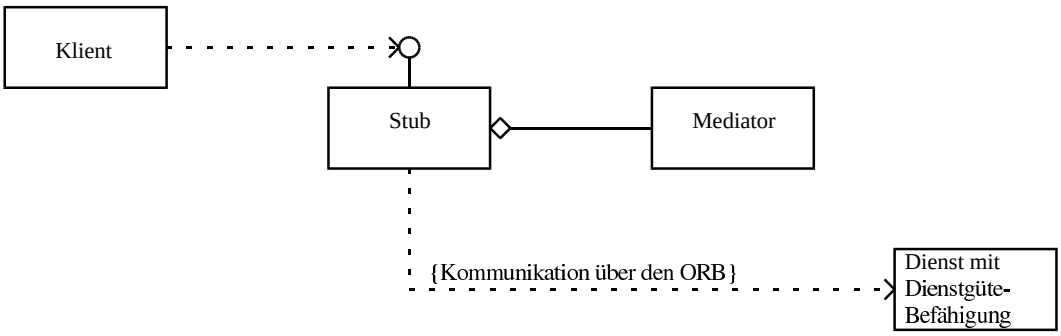


Abbildung 5.6: Dienstgüte-Verhalten auf der Klientenseite

Klientenseite

Die Integration von Dienstgüte-Verhalten auf der Klientenseite ist weniger aufwendig als auf der Dienstseite. Abbildung 5.6 zeigt die Struktur der Klientenseite. Neben dem Stub-Objekt, wie es im verteilten Objektmodell bereits existiert, kommt nur ein weiteres Objekt hinzu. Der Mediator tritt zwischen das Stub-Objekt und den ORB. Jeder Auftrag, der von einem Klient an das Stub-Objekt gestellt wird, erfährt eine Umleitung über den Mediator. Dieser ist ein Aggregrat des Stub-Objekts. Der Mediator kann auf den Inhalt des Auftrags zugreifen und Dienstgüte-bezogene Aktionen initiieren bzw. durchführen. Weiterhin ist der Mediator für die Konfiguration der Dienstgüte-Mechanismen zuständig. Der Mediator übergibt den Aufruf dann über die Aufrufschnittstelle an den ORB. Wird das Ergebnis vom ORB zurückgeliefert, nimmt der Mediator dieses vor dem Klienten an. Somit kann der Mediator wiederum entsprechende Aktionen auf dem Ergebnis, wie Entschlüsselung, vornehmen, bevor er es an den Klienten weiterleitet.

Im Gegensatz zu der Dienstseite, bei der ein Objekt eine Schnittstelle implementiert und über die Objektreferenz von anderen Objekten erreichbar ist, existiert auf der Klientenseite zunächst kein solches Objekt. Dies bedeutet, daß ein Objekt auf der Klientenseite instantiiert werden muß, damit Aufrufe der Dienstgüte-Implementierung auf der Dienstseite mit der Dienstgüte-Implementierung auf der Klientenseite kommunizieren können. Weiterhin ist für die Überwachung und Steuerung der Dienstgüte-Implementierung auf der Klientenseite Zugriff für außerhalb des Klienten-

adreßraums lokalisierte Einheiten zu schaffen. Das Stub-Objekt mit dem Mediator erlaubt nur den lokalen Zugriff auf die Dienstgüte-Implementierung.

Die Lösung dieser beiden Probleme besteht in der Erzeugung eines Stellvertreter-Objektes auf der Klientenseite, das zwei Schnittstellen anbietet: eine *Benachrichtigungs-Schnittstelle*, mit der die Klienten-Anwendung auf sich ändernden Dienstgüte-Niveaus hingewiesen werden kann, und eine *Steuerungs-Schnittstelle*, die Aufrufe an die Klienten-seitige Dienstgüte-Implementierung erlaubt.

Erweiterung um die Auswahl der Dienstgüte-Charakteristik

Bislang ist bei der Betrachtung der Umsetzung von QIDL in die Zielsprache eine Dienstgüte-Befähigung $< D, \varphi >$ zugrundegelegt gewesen. QIDL erlaubt jedoch eine beliebige Anzahl Dienstgüte-Charakteristiken bei der Zuordnung an einen Dienst. Da zur Laufzeit nur jeweils eine der zugeordneten Dienstgüte-Charakteristiken aktiv sein kann, ist durch die Umsetzung sicherzustellen, daß die entsprechenden Einheiten der jeweils aktiven Dienstgüte-Charakteristik auf Klienten- und Dienstseite installiert werden. Aufgrund der generierten Struktur auf Klienten- und Dienstseite ist dies jedoch leicht zu erweitern.

Auf der Klientenseite existiert im erweiterten Stub-Objekt eine Methode, um den aktuellen Mediator zu setzen. Ist durch eine Dienstgüte-Verhandlung eine Dienstgüte-Vereinbarung erzielt, muß der Klient nur einen entsprechenden Mediator instantiieren und an das Stub-Objekt übergeben. Durch Funktionen des Rahmenwerks wird dies vereinfacht.

Auf der Dienstseite ist der Austausch des Aggregats für die Dienstgüte-Implementierung ähnlich einfach möglich. Problematisch ist hier allerdings, daß der Dienst potentiell die Schnittstellen aller zugeordneten Dienstgüte-Schnittstellen unterstützen muß, damit der Zugriff über die Objektreferenz geschehen kann. Da die Implementierung dieser Schnittstellen aber durch Delegation an die Dienstgüte-Implementierung weitergereicht wird, kann nur die aktuell installierte Dienstgüte-Implementierung erreicht werden. Alle Aufrufe an eine nicht installierte Dienstgüte-Charakteristik führen zu einer Ausnahme-Behandlung.

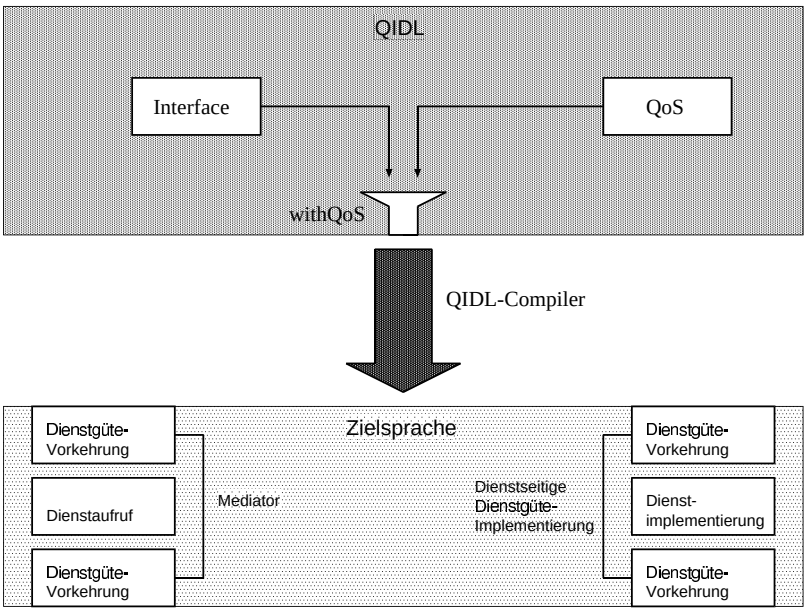


Abbildung 5.7: Aspekt-Verwebung

5.5 Einordnung des Ansatzes im Sinne der aspektorientierten Programmierung

Die in diesem Kapitel vorgestellte Spezifikation von Dienstgüte-Charakteristiken und deren Umsetzung in Zielsprachen realisiert einen aspektorientierten Ansatz. Abbildung 5.7 illustriert das Zusammenführen der Aspekte.

Dabei sind die betrachteten Aspekte die Dienstgüte-Charakteristik – gegeben durch ihre Schnittstellen-Spezifikation in QIDL – und der Dienst, der auch durch seine Schnittstellen-Definition in QIDL vorgegeben ist. Die Zuordnung einer Dienstgüte-Charakteristik zu einem Dienst mittels `withQoS` zeigt dem QIDL-Compiler die Abhängigkeit der Aspekte an. Die Verwebung der Aspekte zu einem Programm erfolgt durch den QIDL-Compiler, der hier die Rolle des AspectWeavers im Sinne der AOP übernimmt. Dabei werden Vorlagen in der Zielsprache entsprechend der in 5.4.2 beschriebenen Regeln generiert. Die Implementierung der Vorlagen wird von dem Dienstgüte- bzw. Anwendungsentwickler vorgenommen. Durch die Umsetzung wird eine weitgehende Entkopplung der Vorlagen erreicht und dennoch die Integration der Dienstgüte-Aspekte in die Anwendung gewährleistet.

Wie bei der Integration von Dienstgüte-Vorkehrungen in objektorientierte Systeme kann auch bei der Realisierung eines aspektorientierten Ansatzes zwischen der impliziten und der expliziten Dienstgüte-Integration unterschieden werden.

Wenngleich die implizite Integration wünschenswert ist, zeigen die Beispiele der Verfügbarkeit durch eine Replikatgruppe oder Lastbalancierung, daß dies im generischen Fall der Dienstgüte-Integration nicht aufrechterhalten werden kann.

Die beteiligten Aspektsprachen sind die QIDL zur Beschreibung der Schnittstellen von Dienstgüte-Charakteristiken und Diensten sowie die Implementierungssprache, in welche die QIDL-Definitionen umgesetzt werden. Die Verwebung der Aspekte, respektive deren Implementierungen, wird durch die generierten Vorlagen des QIDL-Compilers sichergestellt. Anwendungs- wie Dienstgüte-Implementierer werden keinen weiteren Sprachen oder Werkzeugen außer der IDL-Erweiterung durch die QIDL ausgesetzt.

5.6 Auswirkung auf die Basisinfrastruktur

5.6.1 Wiederverwendung von Diensten

Die hier vorgestellten Erweiterungen der OMG-IDL und die Umsetzung der Erweiterung in die Zielsprache beeinträchtigen die Definition und die Umsetzung von Diensten ohne Dienstgüte-Befähigung nicht. Allerdings ist bei dem Beispiel auf Seite 99 zu sehen, daß durchaus Abhängigkeiten zwischen Diensten mit und ohne Dienstgüte-Befähigung existieren. So sollte es einfach sein, bestehende Dienste wiederzuverwenden und mit Dienstgüte-Befähigung zu versehen. Bei dem Entwurf und der Implementierung eines Dienstes, der Dienstgüte-Befähigung erlaubt, sollte die eigentliche Dienstimplementierung auch ohne Dienstgüte-Befähigung einsetzbar sein oder aber mit anderen Dienstgüte-Charakteristiken versehen werden können. Dies kann mit der vorgestellten QIDL-Umsetzung leicht bewerkstelligt werden.

In Abbildung 5.8 ist der ursprüngliche Dienst durch die Generierung des "B-Skel"-Skeletons in der Zielsprache gegeben. Die Implementierung dieses Dienstes "B-Impl" wird an den Dienstgüte-befähigten Dienst vererbt. So wird das Dienst-spezifische Verhalten von dem Dienstgüte-befähigten Dienst vererbt. In dem Skeleton des Dienstgüte-befähigten Dienstes finden

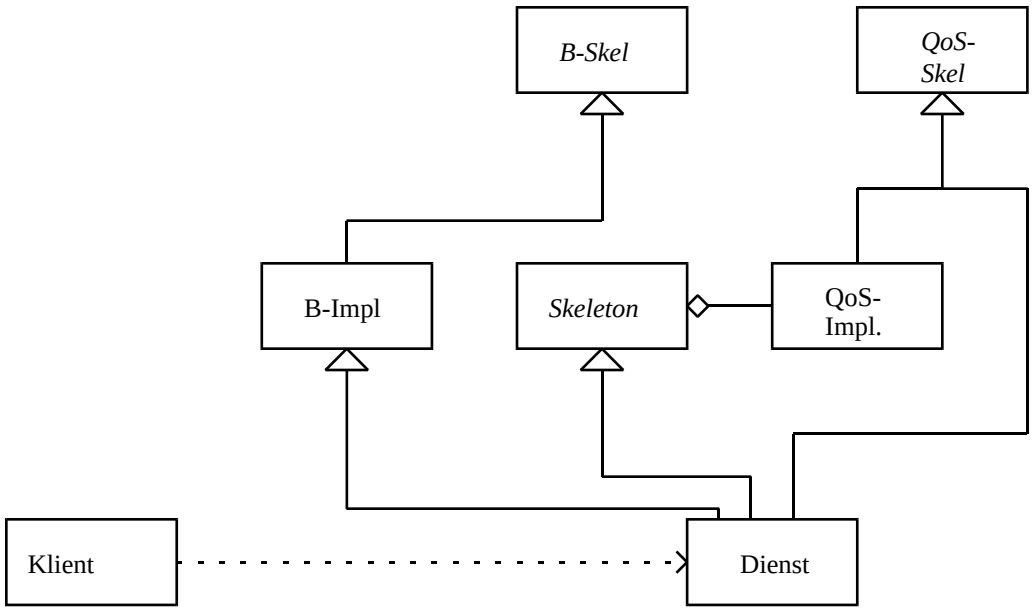


Abbildung 5.8: Wiederverwendung von Diensten bei Dienstgüte-Befähigung

sich nur noch die zusätzlichen Methoden für die Dienstgüte-Anpassung, also die Aspekt-Integration. Die vorgestellte Struktur wird durch den QIDL-Compiler in Form von Implementierungs-Vorlagen generiert und ist somit einfach vom Anwendungsentwickler nutzbar.

5.6.2 Auswirkungen auf die Subtyp-Beziehung

Die hier vorgestellte Struktur zur Sicherstellung der Wiederverwendbarkeit von Diensten wirft jedoch einige Probleme in Bezug auf das Verhältnis zwischen Diensten mit und ohne Dienstgüte-Befähigung auf. Vererbung in objekt-orientierten Programmiersprachen wird häufig zur Modellierung eine Generalisierungs/Spezialisierungs-Beziehung benutzt. Eine Basisklasse B modelliert Verhalten und Struktur, das von einer Subklasse S durch Vererbung wiederverwendet wird. Dabei kann S neues Verhalten und neue Struktur hinzufügen. In objektorientierten Sprachen werden durch solche Taxonomien von Klassen Typhierarchien gebildet [19]. Dabei wird eine Klasse als ein abstrakter Datentyp aufgefaßt. In solchen Hierarchien ist eine wesentliche Eigenschaft für die Modellierung, daß ein Subtyp an der Stelle eines Basistyps stehen kann. Für die Subtyp-Relation soll hier die Be-

zeichnung $S \leq B$ verwendet werden, um die Beziehung "S ist Subtyp von B" auszudrücken. Eine Subtyp-Relation $D' \leq D$ erlaubt die Verwendung von Objekten des Typs D' (Instanzen der Klasse D') an Orten, wo Objekte des Typs D erwartet werden. Damit soll hier – wie auch in [57] – die Anforderung nach der Ersetzbarkeit im Vordergrund stehen. Dies bedeutet für das Objekt des Subtyps, daß es sich "genau so" verhalten muß, wie ein Objekt des Basistyps sich gegenüber Benutzern des Basistyps verhält.

```
interface D' : D withQoS phi {
...
};
```

Die Dienstgüte-Bereitschaft $< D', \phi >$ repräsentiert diesen Dienst. Aus der Ableitung $D' \leq D$ die Subtyprelation für $< D', \phi > \leq D$ zu schließen ist naheliegend, da $< D', \phi >$ zunächst die Schnittstelle von D und darüber hinaus eine bestimmte Dienstgüte-Befähigung anbietet. Betrachtet man jedoch die Instanzen der Klassen, bedeutet dies konkret, daß eine Dienstgüte-Vereinbarung als Instanz der Dienstgüte-Befähigung vorliegt. Diese Dienstgüte-Vereinbarung unterliegt aber Einschränkungen, die sich aus der Aspekt-Natur der Dienstgüte-Integration ergeben. So kann zwar der Dienst-Aspekt D' durchaus anstelle von D benutzt werden, aber dies gilt nur so lange, wie keine konkrete Dienstgüte-Vereinbarung $[K, D', \overline{\phi}]$ mit einem Klienten K getroffen wurde. Sei ϕ die Dienstgüte-Charakteristik "reliable" mit einem auf einer Replikat-Gruppe basierenden Dienstgüte-Mechanismus aus dem zuvor eingeführten Beispiel. Wird explizit die Schnittstelle D der mit $[K, D', \overline{\phi}]$ assoziierten Objektreferenz benutzt, läuft die Kommunikation über das Stub-Objekt der Schnittstelle D zu dem Dienst. Dieses Stub-Objekt ist nicht in der Lage, die Dienstgüte-Erbringung zu gewährleisten. Im Sinne der Subtyp-Relation ist das auch so gewünscht, da nur das von der Basisklasse vorgegebene Verhalten realisiert werden soll. Dies hat aber für andere Benutzer der Replikatgruppe den Effekt, daß ein Replikat einen abweichenden Zustand besitzen kann. Damit ist die virtuelle Synchronität der Replikat-Gruppe nicht mehr gegeben und die Dienstgüte-Erbringung verletzt. Somit ist durch die Umsetzung der Dienstgüte-Spezifikation dafür Sorge zu tragen, daß eine solche Substitution nicht stattfinden kann. Im Rahmenwerk gilt $< D', \phi > \not\leq D$.

Die weiteren Fälle, die Beachtung finden müssen, sind zwischen Diensten mit Dienstgüte-Befähigung definiert. Diese Fälle lassen sich durch die Betrachtung der folgenden Konstellation abdecken:

```
interface D1 withQoS phi {
...
};
interface D2 withQoS psi {
...
};
interface D3 : D1, D2 {
...
};
```

Der Dienst D_3 wird durch den funktionalen Anteil der Dienste D_1 und D_3 gebildet. Die beiden Dienstgüte-Charakteristiken ψ und ϕ werden nicht zu einer Dienstgüte-Charakteristik zusammengefügt. Dies würde – wie bei der Schnittstellenvererbung durch `extends` beschrieben – eine neue Dienstgüte-Implementierung nach sich ziehen. Stattdessen werden sie dem Dienst D_3 zugeordnet. Somit ergibt sich $< D_3, \{\phi, \psi\} >$ als Dienstgüte-Befähigung von D_3. Es ist aber bei der Umsetzung sicherzustellen, daß die Aspekt-Integration der Basisklassen nicht vererbt wird. Es sei wieder das Beispiel der Dienstgüte-Charakteristik "reliable" angeführt. Sollte D_1 die Aspekt-Integration mittels `put_state` und `get_state` realisieren und dieses Verhalten von D_3 geerbt, würde nur der Zustands-Anteil von D_1 in D_3 repliziert.

Für die Substituierbarkeit im Sinne der Subtyprelation gelten die oben gemachten Betrachtungen. Auch zwischen $< D_3, \{\phi, \psi\} >$ und $< D_1, \phi >$ bzw. $< D_2, \psi >$ gilt die Subtyp-Relation nicht, da jeweils die Dienstgüte-Charakteristik der anderen Basisklasse verletzt werden könnte.

Zusammenfassend läßt sich feststellen, daß der Einbezug von Dienstgüte-Befähigung signifikante Auswirkungen auf die Typisierung von Diensten hat. Die durch die Dienstgüte-Befähigung aufgeworfenen Konflikte lassen nur eine restriktive Behandlung der Subtyp-Relation zu. Durch den QIDL-Compiler werden bei der Ableitung von Dienst- und Dienstgüte-Schnittstellen als Hilfestellung die Schnittstellen der abgeleiteten Einheiten aus denen der Basiseinheiten aggregiert. Dies dient zum einen der Erleichterung der Implementierung, und zum anderen kann so der Zugriff auf die Definitionen der Basiseinheiten durch Vererbung unterbunden werden.

5.7 Zusammenfassung und Bewertung

Der in diesem Kapitel vorgestellte Ansatz zur Spezifikation von Dienstgüte-Charakteristiken erfüllt die in Kapitel 4 beschriebenen Anforderungen. Insbesondere wird die Trennung der Verantwortlichkeiten (Anwendung/Dienstgüte) durch einen aspektorientierten Ansatz gewährleistet. Die Vermaschung der Anwendungslogik und der Dienstgüte-Implementierung erfolgt durch die Umsetzung der QIDL in die Zielsprache weitestgehend transparent für Anwendungsentwickler. Lediglich auf der Dienstseite ist durch die Aspekt-Integration eventuell Anpassung notwendig.

Durch die Erweiterung der Schnittstellenbeschreibungssprache ist keine weitere Sprache für die Dienstgüte-Spezifikation notwendig. Dies hat den Vorteil, daß in der IDL alle für die Dienstnutzung notwendigen Informationen vorliegen. Da Anwendungsentwickler ohnehin einen IDL-Compiler benutzen müssen, um die für die Interaktion notwendigen Vorlagen zu erhalten, ist die Einbettung des AspectWeavers in den IDL-Compiler naheliegend. Desweiteren sind keine zusätzlichen Aspektsprachen für die Anwendungs- und Dienstgüte-Aspekte notwendig, da beide in derselben Zielsprache implementiert werden.

Der notwendige Aufwand für die Realisierung eines aspektorientierten Ansatzes lohnt für generisches Dienstgüte-Management. In Hinblick auf flexible Erweiterbarkeit von Anwendungen kann so die Trennung der Verantwortlichkeiten etabliert werden. Für Dienstgüte-Anforderungen mit einer gewissen Orthogonalität zu der Anwendung, wie es bei systemnahen Dienstgüte-Charakteristiken der Fall ist, gelingt die Trennung der Verantwortlichkeiten durch die Aspektsprachen. Es ist zu untersuchen, inwieweit sich Dienstgüte-Charakteristiken mit einem höheren Anwendungsbezug – beispielsweise in Bezug auf die numerische Genauigkeit eines Berechnungsdiensts – geeignet in QIDL beschreiben und im Rahmenwerk implementieren lassen.

Kapitel 6

Dienstgüte-Mechanismen-Integration

Die bislang vorgestellten Komponenten des Rahmenwerks haben die Dienstgüte-Integration in die Schicht der Anwendungsobjekte adressiert. Aufgrund der Ende-zu-Ende-Eigenschaft der Dienstgüte-Erbringung ist für viele Dienstgüte-Charakteristiken Unterstützung innerhalb des ORBs und vom zugrundeliegenden Betriebssystem bzw. Netzwerk notwendig. In diesem Kapitel werden die Anforderungen an die Dienstgüte-Mechanismen-Integration in den ORB dargelegt und ein Lösungsansatz vorgestellt.

6.1 Überblick

Die Integration von Dienstgüte-Mechanismen in den Kommunikationspfad zwischen Klient und Dienst wird durch die fehlende Standardisierung des Aufbaus von Verteilungsinfrastrukturen erschwert.

Daher ist es für die Betrachtung der Dienstgüte-Mechanismen-Integration notwendig, eine bestimmte zugrundeliegende Architektur vorauszusetzen bzw. ein abstraktes Modell einzuführen, dem gängige Architekturen unterzuordnen sind. Eine Untersuchung zweier weitverbreiteter Verteilungsinfrastrukturen (DCOM und CORBA) zeigt große Ähnlichkeiten [20]. Die Integration von Dienstgüte-Mechanismen soll sich hier an einem Modell einer Verteilungsinfrastruktur orientieren. Damit ist die Übertragbarkeit in Verteilungsinfrastrukturen, die diesem Modell folgen, gewährleistet.

6.1.1 Modell einer Verteilungsinfrastruktur

Abbildung 6.1 zeigt die hierarchische Untergliederung eines Verteilungsinfrastrukturkerns (ORB) nach seinen Aufgaben.

Die Schnittstellen zu den Anwendungsobjekten werden durch die Aufruf- und Weiterleitungsschnittstelle gebildet. Diese Schnittstellen sind nicht unmittelbar für die Anwendung sichtbar. Die Aufrufschnittstelle wird von den Stellvertreter-Objekten auf der Klientenseite benutzt, um Aufträge an den Kern zu stellen. Auf der Dienstseite ist die Weiterleitungsschnittstelle für die Übermittlung der Aufträge an die Dienstobjekte zuständig. Da auf der Dienstseite typischerweise mehrere Dienstobjekte residieren können, werden diese durch eine Verwaltungseinheit – in der CORBA-Terminologie Objekt Adapter – gebündelt, die als Stellvertreter die Weiterleitungsschnittstelle bedient.

Ist ein Auftrag an den Kern übergeben worden, muß zuerst der Ort des Zielobjekts bestimmt werden. Im Auftrag ist das Zielobjekt durch die Objektreferenz bestimmt. Diese Objektreferenz enthält in geeigneter Form die Adresse des Zielobjekts, bspw. durch Angabe der IP-Adresse und der Port-Nummer an der die Verteilungsinfrastruktur des Dienstes im Netzwerk verfügbar ist. Liegt der angefragte Dienst lokal vor, kann der Aufwand für die Kommunikation durch lokale Inter- bzw. Intra-Prozeßkommunikation reduziert werden.

In der Abbildung 6.1 sind nun zwei Pfade der Aufträge erkennbar (entfernt, lokal). Diese werden in Abhängigkeit des Zielobjekt-Orts durchlaufen.

Lokal: Liegt das Zielobjekt lokal zum Klienten – also auf demselben Rechner – kann effiziente Interprozeßkommunikation, wie Shared memory oder Pipes, benutzt werden.

Der einfachste Fall liegt bei einem prozeßlokalen Dienst vor. Dann genügt ein Methodenaufruf auf dem Dienstobjekt. Damit würden allerdings auch die Weiterleitungsschnittstelle und alle anderen zwischengeschalteten Einheiten der Verteilungsinfrastruktur ausgelassen werden.

Liegt das Zielobjekt auf demselben Rechner aber nicht im selben Adreßraum wie der Klient, kann durch lokale Interprozeßkommunikation immer noch ein Geschwindigkeitsvorteil gegenüber entfernter Kommunikation erreicht werden. Die notwendigen Protokolle können beispielsweise auf Konvertierung der Daten verzichten.

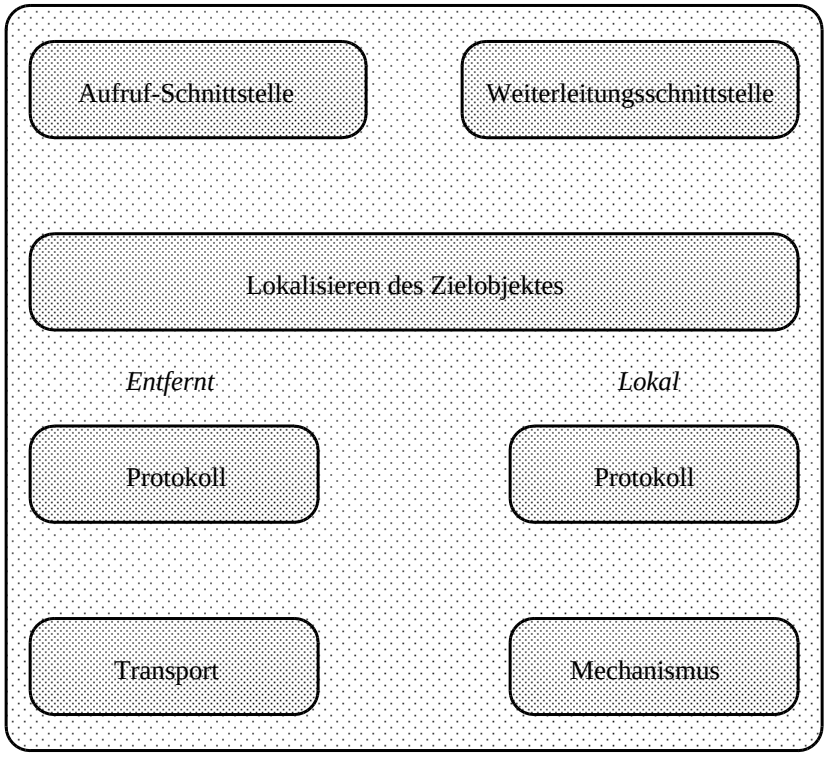


Abbildung 6.1: Modell eines Verteilungsinfrastrukturkerns

Entfernt: Bei der Kommunikation mit Diensten auf entfernten Rechnern sind für den Datenaustausch weitere Vorkehrungen notwendig. Zunächst sind die Daten in ein Format zu übertragen, das auf der Dienstseite in die entsprechenden lokalen Repräsentationen umgewandelt werden kann. Weiterhin ist in dem Protokoll dafür Sorge zu tragen, daß Aufträge sicher zugestellt werden, die Reihenfolge eingehalten wird und eventuelle Verwaltungsnachrichten etabliert werden können (Abbruch eines Auftrages, Meldung über den Ortswechsel eines Diensts, etc.).

Ähnlich wie im lokalen Fall ist auch hier eine Trennung in Protokoll- und Transport-Ebene zu sehen. Aufgabe der Protokoll-Ebene ist die Definition von Nachrichten, bspw. Aufruf oder Ergebniszustellung, und die Abbildung von Daten der Anwendungsebene auf eine netzwerkweit eindeutige Repräsentation (Marshalling). Die Transport-Ebene ist für die Abbildung der Protokoll-Ebene auf ein konkretes Netzwerk-Protokoll zuständig. Ein Beispiel einer solchen Trennung ist das General-Inter-ORB-Protocol (GIOP) der CORBA, das nur einen verläßlichen, verbindungsorientierten Transport-Dienst voraussetzt. Das Internet-Inter-ORB-Protocol (IIOP) stellt die konkrete Abbildung von GIOP auf TCP/IP dar.

6.1.2 Implementierungsalternativen

Ähnlich wie bei Betriebssystemen gibt es unterschiedliche Varianten den Kern einer Verteilungsinfrastruktur zu realisieren. Hier sollen kurz zwei Alternativen skizziert werden, die in Anlehnung an Klassifizierungen im Betriebssystementwurf [67] als Hierarchie-basierter und Mikrokern-basierter Ansatz bezeichnet werden sollen.

Mikrokern

Als ein Mikrokern wird im Bereich der Betriebssysteme ein Kern bezeichnet, der im wesentlichen die Grundaufgaben eines Betriebssystems, wie Prozeßrealisierung und Kommunikation, bereitstellt und weitere Aufgaben, wie Dateisysteme, in Dienste auslagert. Mikrokerne versprechen eine übersichtlichere Struktur und inhärente Modularität, die der Erweiterbarkeit zugute kommt.

Eine CORBA-Implementierung, die auf dem Mikrokern-Ansatz beruht, ist MICO – MICO is CORBA [83]. Der Kern von MICO verfügt nur über

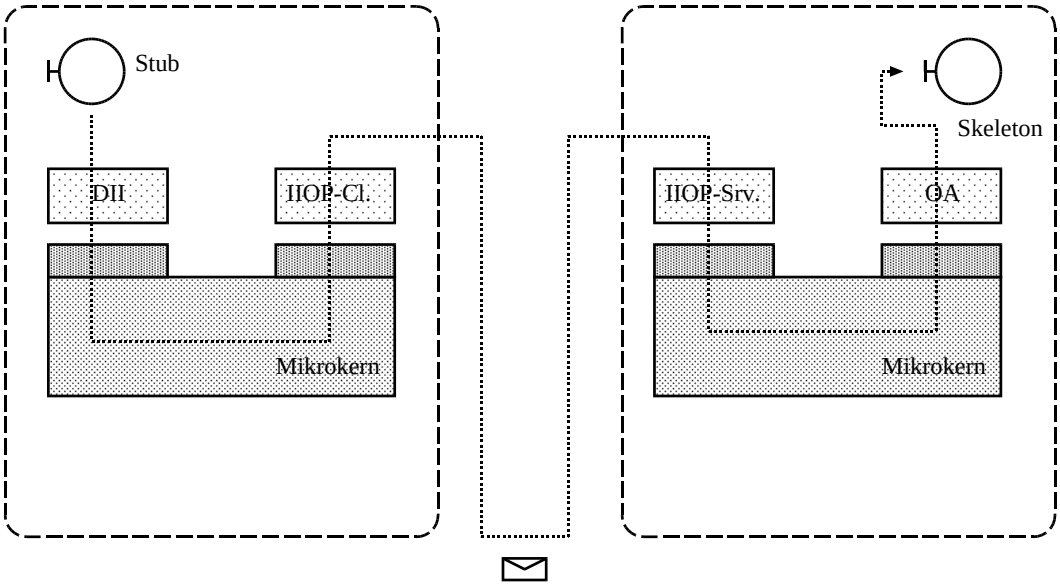


Abbildung 6.2: Mikrokern-ORB

eine einfache Aufruf- und Weiterleitungsschnittstelle. Der Mikrokern ist dafür zuständig, Anfragen der Aufrufschnittstelle an ein entsprechendes Modul an der Weiterleitungsschnittstelle zu übermitteln. Abbildung 6.2 zeigt die beteiligten Module bei rechnerübergreifender Kommunikation. Ein Stub-Objekt stellt einen Auftrag an das Dynamic Invocation Interface (DII). Das DII-Modul wandelt den Auftrag in die Mikrokern-interne Struktur und übermittelt diesen an die Aufrufschnittstelle. Der Mikrokern erkennt an der Objektreferenz, daß der angefragte Dienst entfernt ist und leitet den Auftrag an den IIOP-Klienten weiter. Dieser wandelt den Auftrag in eine IIOP-Nachricht und übermittelt ihn an den IIOP-Server des ORBs der Dienstimplementierung. Nachdem der Auftrag über IIOP die Rechnergrenzen überwunden hat und auf der Dienstseite im IIOP-Server angekommen ist, wird er dort in die Mikrokern-interne Struktur gewandelt und wiederum über die Aufrufschnittstelle dem Mikrokern der Dienstseite übergeben. Der Mikrokern stellt fest, daß der Dienst lokal existiert und leitet den Auftrag an einen Objekt-Adapter weiter. Der Objekt-Adapter kann nun über das Skeleton-Interface den Auftrag an den Dienst übermitteln.

Die Erweiterung des Mikrokerns um neue Transportprotokolle kann nun durch Hinzufügen neuer Module auf Klienten- und Dienstseite geschehen. Abhängig von der Struktur des Mikrokerns sind die Module aber nicht iso-

liert implementierbar. So kann ein Mikrokern mit ereignisbasiertem Scheduler die Interaktion der Transportmodule mit dem Kern in aufwendiger Art und Weise zur Folge haben.

Hierarchisch

Im Gegensatz zu einem Mikrokern-basierten Entwurf kann auch die Hierarchie aus Abbildung 6.1 direkt in einen hierarchischen ORB abgebildet werden. Dabei entsprechen die einzelnen Einheiten dann der Spezifikation der Verteilungsinfrastruktur. Im Falle der CORBA würde die Aufrufschnittstelle unmittelbar das Dynamic Invocation Interface bzw. das Static Invocation Interface anbieten. Die Protokollebene würde GIOP und die Transportebene IIOP realisieren. Je nach Struktur des Kerns kann auf die Protokollebene auch gänzlich verzichtet werden. Orbix [45] und ORBacus [77] sind prominente Vertreter von hierarchischen ORBs.

6.2 Anforderungen

Mit dem Hintergrund der Verantwortlichkeiten eines Verteilungsinfrastrukturkerns, wie er in Abbildung 6.1 skizziert ist, und der Betrachtung von Dienstgüte-Mechanismen (vgl. Kapitel 3 und Abbildung 3.8) werden hier die Anforderungen an die Integration von Dienstgüte-Mechanismen in den ORB beschrieben.

- **Benutzerspezifizierte Protokolle und beliebiger Transport:** Für die Einbettung von Dienstgüte-Mechanismen des Betriebssystems, Netzwerks oder bereits existierender Bibliotheken muß eine Anpassung der Auftragsübermittlung gewährleistet sein. Dies bedeutet, daß nicht nur die reine Netzwerkübertragung, sondern auch die darüber gelagerten Protokolle, wie im Beispiel CORBA das GIOP, angepaßt bzw. ersetzt werden müssen.

- **Dynamisches Nachladen:** Eine rein statische Konfiguration der Dienstgüte-Mechanismen im Kern ist wenig wünschenswert. Gerade bei der Unterstützung generischen Dienstgüte-Managements sind die Anforderungen nach Erweiterbarkeit des ORB-Kerns gegeben. Bei

statisch gebundenen Sprachen, wie C++ oder C, sind hier besondere Vorkehrungen notwendig, um die dynamische Erweiterbarkeit des ORBs zu gewährleisten.

- **Schnittstellen:** Die Dienstgüte-Mechanismen im Kern müssen bestimmte Schnittstellen unterstützen. Hinzu kommt eine Verwaltungsfunktionalität für die dynamische Integration von Dienstgüte-Mechanismen.

  - Konfiguration: Diese Schnittstellen sind für das Laden und Entfernen von Dienstgüte-Mechanismen des ORBs verantwortlich. Dazu gehört auch die Zuweisung von Dienstgüte-Mechanismen an eine Klienten/Dienst-Beziehung.
  - Einbettung: Die Einbettung von Dienstgüte-Mechanismen in den Ende-zu-Ende-Kommunikationspfad bedeutet, daß Dienstgüte-Mechanismen Anfragen entgegennehmen und mit bestimmten Dienstgüte-Zusicherungen übertragen. Darüber hinaus ist sicherzustellen, daß die Aufträge und deren Ergebnisse an die entsprechenden Einheiten – Klient und Dienst – weitergeleitet werden. Somit müssen die Dienstgüte-Mechanismen Schnittstellen für die ORB-interne Aufrufschnittstelle anbieten und die ORB-interne Weiterleitungsschnittstelle bedienen können.
  - Spezifisch: Neben den bislang beschriebenen Schnittstellen, die von allen Dienstgüte-Mechanismen angeboten werden müssen, existieren abhängig der Funktionalität der Dienstgüte-Mechanismen auch spezifische Schnittstellen eines Dienstgüte-Mechanismus. Ein Multicast-Transport muß beispielsweise die Gruppenverwaltung anbieten und ein Mechanismus für Verschlüsselung Zugriff auf die Schlüssel des Benutzers erhalten.

- **Geeignet für unterschiedliche ORB-Architekturen:** Die Struktur des ORBs kann unterschiedlich beschaffen sein. Dabei ist die den Anwendungen angebotene Schnittstelle jedoch gleich. Daher ist bei der Auslegung der Schnittstellen und der Integration von Dienstgüte-Mechanismen darauf zu achten, daß sie auch für unterschiedliche ORB-Architekturen realisierbar sind.

6.3 Eine reflektive Integration von Dienstgüte-Mechanismen

In diesem Abschnitt wird eine Integration von Dienstgüte-Mechanismen in Verteilungsinfrastrukturen vorgestellt. Der vorgeschlagene Ansatz kann in unterschiedlichen Basisinfrastrukturen zum Einsatz kommen. Die Dynamik des Systems wird durch einen reflektiven Ansatz gewährleistet.

6.3.1 Grundkonzept

Reflektive Systeme können über bestimmte Teilbereiche ihrer selbst Schlüsse ziehen und Änderungen initiieren. Man spricht in diesem Kontext auch von "open engineering". Im Kontext verteilter Objektsysteme bedeutet dies, daß Objekte nicht nur über statisch gebundene Schnittstellen miteinander kommunizieren können, sondern Informationen über andere Objekte dynamisch erhalten und Aufrufe zustellen können. In [16] findet sich ein Überblick von Reflektion in objekt-orientierten Verteilungsinfrastrukturen.

Eine reflektive Integration von Dienstgüte-Mechanismen erlaubt das nachträgliche Hinzufügen von Dienstgüte-Mechanismen. Bei der Unterstützung von statisch gebundenen Sprachen kann auf Mechanismen wie dynamisch gebundene Bibliotheken zurückgegriffen werden. Dies bedeutet, daß die Implementierung eines Objekts oder einer Funktion erst zur Laufzeit auf Anfrage – eben dynamisch – gebunden wird. Durch solche Mechanismen ist das Hinzufügen von Verhalten mit derselben Schnittstelle möglich. Allerdings ist es unwahrscheinlich, daß sich alle Dienstgüte-Mechanismen durch dieselbe Schnittstelle abbilden lassen. In 6.2 sind schon Beispiele für unterschiedliche Dienstgüte-Mechanismen-bezogene Schnittstellen vorgestellt worden.

Als Lösung bietet sich die Trennung der Schnittstellen in einen allgemeinen und einen Dienstgüte-Mechanismus-bezogenen Teil an. Der allgemeine Bestandteil kann durch eine statische Schnittstellenbeschreibung ausgedrückt werden. Für den Dienstgüte-bezogenen Bestandteil der Schnittstelle muß ein geeigneter Aufrufmechanismus bereitgestellt werden, der zur Laufzeit die entsprechenden Aufrufe weiterleitet.

Die Integration der Dienstgüte-Mechanismen kann an unterschiedlichen Stellen erfolgen. Prinzipiell könnte nach der Lokalisierung jede der zugrundeliegenden Schichten (Protokoll, Transport; vgl. Abbildung 6.1)

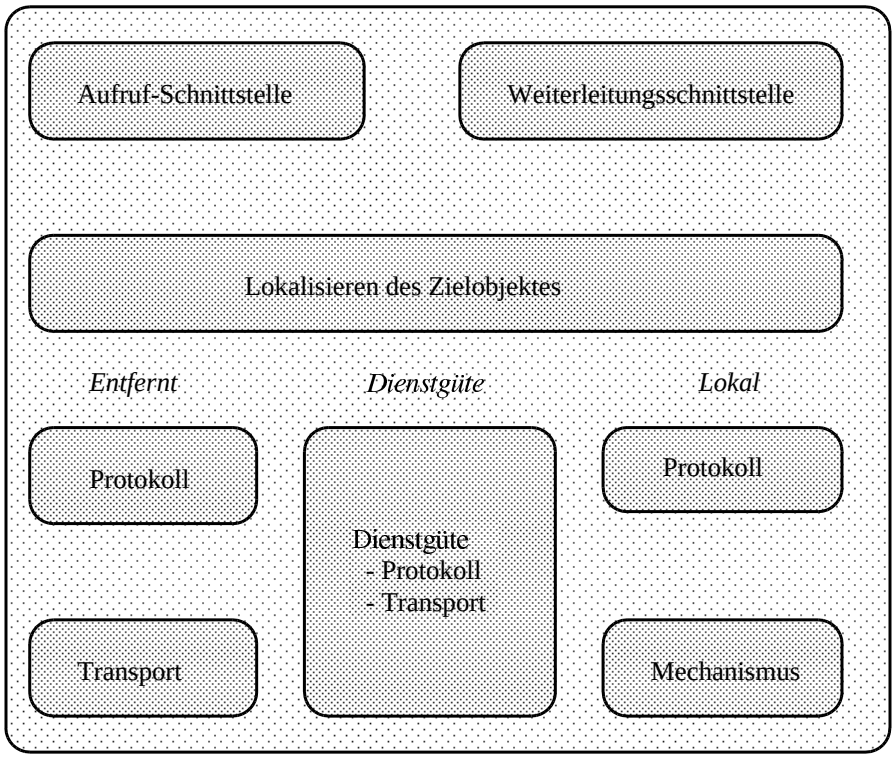


Abbildung 6.3: Verteilungsinfrastruktur-Kern mit Dienstgüte-Mechanismen

durch einen Dienstgüte-befähigten Mechanismus ersetzt werden. Da Protokoll und Transport aber typischerweise zusammenhängen, ist ein Zusammenschluß beider zu einem Dienstgüte-Mechanismus ausreichend.

Abbildung 6.3 zeigt die Erweiterung des ORB-Kerns um einen Dienstgüte-Mechanismus. Die Lokalisierung muß dahingehend erweitert werden, daß sie zwischen lokaler, entfernter und Dienstgüte-befähigter Kommunikation unterscheidet. Ist einer Klient/Dienst-Interaktion eine bestimmte Dienstgüte-Charakteristik zugeordnet, müssen die Dienstgüte-Mechanismen auf der Anwendungsebene die Dienstgüte-Mechanismen im Kern konfigurieren und für die Klient/Dienst-Beziehung einrichten. Sowohl auf der Klienten- wie auch auf der Dienstseite sind dann symmetrisch die entsprechenden Dienstgüte-Mechanismen zu integrieren und zu benutzen. Nach der Lokalisierung des Ziel-Objektes auf der Klientenseite und Weiterleitung der Anfrage an den Dienstgüte-Mechanismus ist dieser für die Realisierung des Austauschprotokolls und die Übertragung der Daten über das Netzwerk zuständig.

Auf der Dienstseite ist der zugehörige Dienstgüte-Mechanismus bereit, die Daten über das Netzwerk mit dem gegebenen Protokoll anzunehmen und dem ORB für die Weiterleitung an den Dienst zu übergeben.

Die hier beschriebene Struktur ist zunächst nur für *einen* Dienstgüte-Mechanismus tauglich. Abhängig von der Lokalisierung und der Auswahl des Übertragungsprotokolls können mehrere Dienstgüte-Mechanismen parallel integriert werden. Die Anforderungen nach dynamischen Nachladen und an die Schnittstellen der Dienstgüte-Mechanismen sind allerdings noch nicht betrachtet worden und bei dieser allgemeinen Struktur stark abhängig von der zugrundeliegenden ORB-Architektur. Der nächste Unterabschnitt bietet eine Lösung für diese Aufgaben an.

6.3.2 Bündelung der Dienstgüte-Mechanismen

Die Bündelung der Dienstgüte-Mechanismen in eine Verwaltungseinheit erlaubt die einfachere Integration in beliebige ORB-Architekturen. Die Verwaltungseinheit – Dienstgüte-Transport in Abbildung 6.4 – verwaltet alle Dienstgüte-Mechanismen und stellt eine einheitliche Schnittstelle für die Konfiguration und die Aufrufannahme/-weiterleitung bereit. So muß nur sichergestellt werden, daß alle Interaktionen zwischen Dienstgüte-befähigten Klienten und Diensten durch die Lokalisierung an diese Verwaltungseinheit weitergeleitet werden. Die Integration in einen gegebenen ORB erfolgt durch die Anpassung der Schnittstellen der Verwaltungseinheit. Die einzelnen Dienstgüte-Mechanismen und deren Verwaltung erfolgt durch die Verwaltungseinheit.

Abbildung 6.4 zeigt die Zusammenfassung der Dienstgüte-Module (QoS-Module) durch die Verwaltungseinheit. Die Verwaltungseinheit wird im Rahmenwerk als Dienstgüte-Transport bezeichnet, und es wird zwischen der Klienten- und Dienstseite unterschieden. Aufgaben des Dienstgüte-Transport sind:

- **Laden/Entfernen von Dienstgüte-Modulen:** Die Dienstgüte-Mechanismen auf der Anwendungsebene müssen bei Etablierung eines Dienstgüte-Niveaus dafür Sorge tragen, daß evtl. benötigte Dienstgüte-Mechanismen auf der Transportebene bereitgestellt werden. Dazu müssen sie Zugriff auf den Dienstgüte-Transport erhalten und entsprechend das gewünschte Dienstgüte-Modul anfordern. Nach Been-

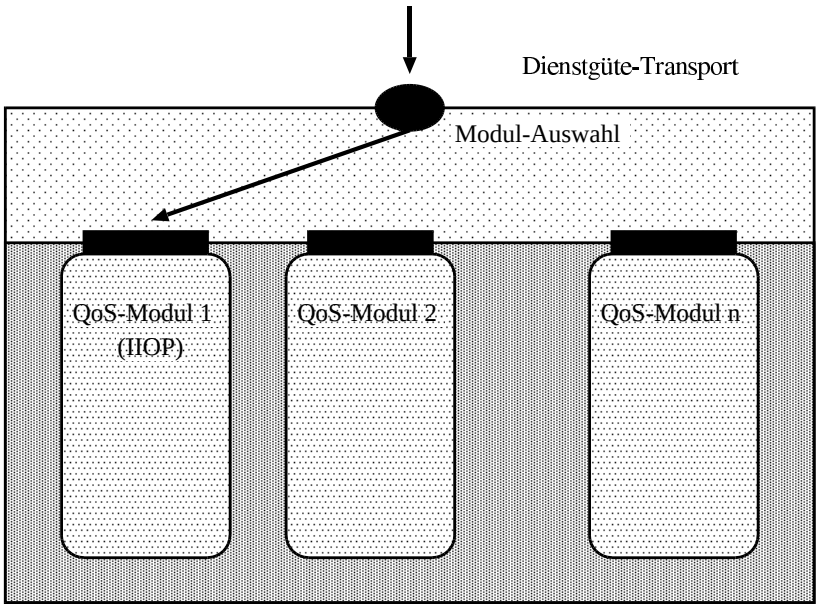


Abbildung 6.4: Verwaltung der Dienstgüte-Module

digung der Interaktion muß ein nicht mehr benötigtes Dienstgüte-Modul wieder freigegeben werden.

- **Annahme/Weiterleitung von Aufträgen mit Dienstgüte-Befähigung:** Die Aufträge zwischen Dienstgüte-befähigten Klienten und Diensten müssen auf der Klientenseite von der Lokalisierung des ORBs angenommen und entsprechend auf der Dienstseite an den ORB weitergeleitet werden, damit dieser den Auftrag an den Dienst zustellen kann. Insbesondere ist der Dienstgüte-Transport dafür zuständig, angenomme Aufträge an das entsprechende Dienstgüte-Modul zu übergeben.

- **Zuordnung Dienstgüte-Module Klient/Dienst-Beziehung:** Damit der Dienstgüte-Transport das entsprechende Dienstgüte-Modul für eine Klient/Dienst-Beziehung auswählen kann, ist dieses beim Dienstgüte-Transport von den Dienstgüte-Mechanismen der Anwendungsebene zuzuweisen. Ist kein Dienstgüte-Modul an eine Klient/Dienst-Beziehung mit Dienstgüte-Befähigung zugewiesen, wählt der Dienstgüte-Transport automatisch das IIOP-Modul (Modul 1 in Abbildung 6.4). Damit ist es überhaupt erst möglich, daß Klient

und Dienst eine Verhandlung zur Erzielung einer Dienstgüte-Vereinbarung durchführen können.

- **Weiterleiten von Konfigurations-Nachrichten an Dienstgüte-Module:** Die individuellen Schnittstellen der Dienstgüte-Module müssen von den Dienstgüte-Mechanismen der Anwendungsebene erreichbar sein. Da die Dienstgüte-Module durch den Dienstgüte-Transport gekapselt werden, muß dieser eine Möglichkeit für die Weiterleitung von Konfigurationsnachrichten an die Dienstgüte-Module bereitstellen. Diese Schnittstelle dient weiterhin dazu, die Dienstgüte-Mechanismen überwachen zu können.

- **Transport Dienstgüte-bezogener Daten an die Dienstgüte-Mechanismen der Anwendungsebene:** Die Befähigung von Dienstgüte hat auch auf die Aufträge zwischen Klient und Dienst Auswirkungen. So kann eine Dienstgruppe benutzt werden, um Diversität zu erzielen. Dabei kommen unterschiedliche Versionen eines Dienstes zum Einsatz, um Fehler durch einen Mehrheitsentscheid auszuschließen. Die Antwort zu einem Auftrag an eine solche Gruppe trägt dann nicht ein Ergebnis, sondern entsprechend viele aus der Gruppe. Daher ist die Struktur von Aufträgen entsprechend zu erweitern.

6.3.3 Involvierte Schnittstellen

In diesem Unterabschnitt wird das vorgestellte Konzept zur Dienstgüte-Mechanismen-Integration in den ORB von der Schnittstellenseite her dargestellt. Da die Dienstgüte-Implementierungen einen hohen Freiheitsgrad besitzen sollen, kann an dieser Stelle wenig mehr als deren Integration und Verwaltung erläutert werden. Abschnitt 6.4 betrachtet die aus diesem Konzept resultierende Dienstgüte-Mechanismen-Hierarchie sowie die Interaktion zwischen den beteiligten Einheiten.

Schnittstellen des Dienstgüte-Transports

Die wesentliche Aufgabe des Dienstgüte-Transports ist die Annahme und Weiterleitung von Aufträgen zwischen Dienstgüte-befähigten Klienten und Diensten an entsprechende Dienstgüte-Module. Dazu muß der Dienstgüte-

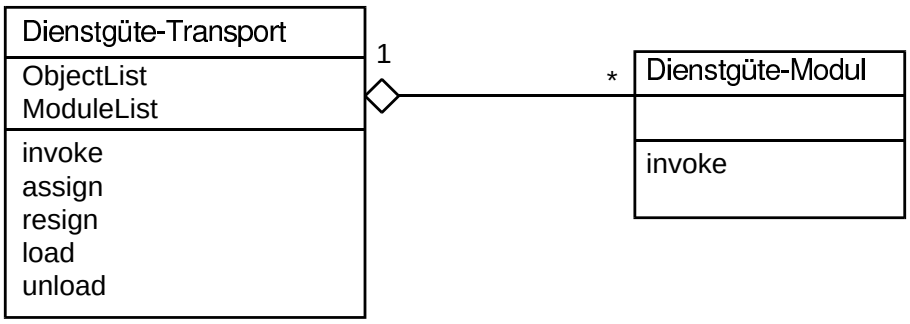


Abbildung 6.5: Dienstgüte-Transport und -Module

Transport eine geeignete Schnittstelle zu der Lokalisierungs- und Weiterleitungs-Funktionalität des ORBs bieten. Diese Schnittstelle ist von der zugrundeliegenden Architektur der Verteilungsinfrastruktur abhängig.

Die weiteren Schnittstellen des Dienstgüte-Transports sind unabhängig von dem ORB. Sie stehen nur den Dienstgüte-Mechanismen der Anwendungsebene und Verwaltungseinheiten zur Verfügung. Im wesentlichen beinhaltet diese Schnittstelle Funktionen zum Laden/Entfernen von Dienstgüte-Modulen, Zuordnung von Dienstgüte-Modulen zu einer Klienten/Dienst-Beziehung und Weiterleiten von Management-Aufträgen an Dienstgüte-Module. In Abbildung 6.5 ist die Schnittstelle des Dienstgüte-Transports skizziert.

Bis auf die Weiterleitung von Management-Aufträgen an die Dienstgüte-Module ist diese Schnittstelle statisch. Sie läßt sich somit durch ein Pseudo-Objekt realisieren. Ein Pseudo-Objekt bietet eine in der IDL definierte Schnittstelle an und läßt sich wie ein gewöhnliches Objekt ansprechen. Die Anwendungsschnittstelle von Verteilungsinfrastrukturen wird oft durch Pseudo-Objekte repräsentiert, bspw. der ORB bei CORBA.

Die Weiterleitung von Management-Aufträgen an die Dienstgüte-Module wiederum ist durch ein solches Pseudo-Objekt nicht möglich, da die Management-Aufträge nicht a priori bekannt sind. Hier muß der Dienstgüte-Transport eine dynamische Aufrufschnittstelle anbieten. Verteilungsinfrastrukturen bieten oft schon dynamische Aufrufschnittstellen, wie das Dynamic Invocation Interface (DII) in CORBA oder IDispatch in DCOM. Es ist naheliegend, die von der Infrastruktur angebotene dynamische Aufrufschnittstelle auch für die Konfiguration der Dienstgüte-Module zu verwenden. Dabei muß die Lokalisierungs- und Weiterleitungskomponente des

ORBs einen solchen Aufruf an den Dienstgüte-Transport weiterleiten, der für die Übermittlung an das entsprechende Dienstgüte-Modul zuständig ist. In Abbildung 6.5 dient die invoke-Methode des Dienstgüte-Transports zur Annahme und Weiterleitung von Aufrufen. Abschnitt 6.4.2 beschreibt die Interaktion zwischen Dienstgüte-Transport und Dienstgüte-Modulen.

Schnittstellen der Dienstgüte-Module

Dienstgüte-Module realisieren den Netzwerk-bezogenen Teil einer Dienstgüte-Charakteristik im System. Im Ende-zu-Ende-Pfad sind die Dienstgüte-Module für den Transport eines Auftrags vom Klienten zum Dienst verantwortlich. Das dabei sicherzustellende Dienstgüte-Niveau muß durch entsprechende Vorkehrungen gewährleistet werden. Diese Vorkehrungen können entweder vom Dienstgüte-Modul selbst oder durch Zugriff auf Vorkehrungen des Betriebssystems oder Netzwerks realisiert werden. Die Dienstgüte-Module kapseln solches Verhalten ein. Relevant ist für die Dienstgüte-Mechanismen der Anwendungsebene die Konfiguration der Dienstgüte-Module.

Dienstgüte-Module bieten eine gemeinsame statische Schnittstelle an, die es ermöglicht, diese als dynamisch gebundene Bibliotheken nachzuladen. Alle Dienstgüte-Module erben von derselben Basisklasse. Sie werden jeweils eindeutig gekennzeichnet, so daß die Verwaltung durch den Dienstgüte-Transport möglich ist. Die Zuordnung einer Klienten/Dienst-Beziehung zu einem Dienstgüte-Modul erfolgt über diese Kennzeichnung.

Die statische Schnittstelle enthält weiterhin eine Methode, die Aufträge als dynamischen Aufruf entgegennimmt. Der Implementierer des Dienstgüte-Moduls kann so die Modul-spezifische Funktionalität unter einer für alle Dienstgüte-Module gleichen statischen Schnittstelle realisieren. Die statische Schnittstelle dient aber in der Hauptsache dazu, Aufträge von Klienten anzunehmen und an das entsprechende Dienstgüte-Modul der Dienstseite weiterzuleiten, wo der Auftrag an den entsprechenden Dienst weitergeleitet wird.

Schnittstelle zu der Anwendung

Die bislang vorgestellten Schnittstellen der Dienstgüte-Mechanismen-Integration haben nur die Interaktion zwischen dem Dienstgüte-Transport und den Dienstgüte-Modulen betrachtet. Die Schnittstelle des Dienstgüte-

Transports zur Anwendung wurde – außer der Verwaltung der Dienstgüte-Module – nicht weiter ausgeführt.

Die Anwendungsebene benötigt zweierlei Schnittstellen zu den zugrundeliegenden Dienstgüte-Mechanismen. Neben den bislang ausgeführten Schnittstellen für die Dienstgüte-Steuerung und -Konfiguration ist das Initiieren eines Auftrages mit Dienstgüte-Befähigung das eigentliche Ziel einer Dienstgüte-Architektur. Der von einen Klienten an das Stub-Objekt gestellte Autrag muß in geeigneter Art und Weise an den Dienstgüte-Transport übergeben werden. Dazu müssen für einen Auftrag Dienstgüte-relevante Informationen mitgeführt werden.

Die im Rahmenwerk zur Anwendung kommende Lösung besteht in der Erweiterung des dynamischen Auftrags. Dabei dient der so entstandene Dienstgüte-Auftrag nicht nur zur Übermittlung des Auftrags und zusätzlicher Dienstgüte-bezogener Informationen an die Dienstgüte-Module, sondern wird für die gesamte Interaktion zwischen der Anwendung und den Dienstgüte-befähigten Komponenten des Rahmenwerks in der Dienstgüte-Mechanismen-Hierachie benutzt.

Ein dynamischer Auftrag enthält:

- **Das Ziel der Operation (Target):** Die Objektreferenz des Dienstes, der die Operation ausführen soll.

- **Den Namen der Operation:** Da ein Dienst unterschiedliche Operationen anbieten kann, muß neben dem Ziel-Objekt auch die auszuführende Operation angegeben werden.

- **Die Parameterliste der Operation:** Operationen werden durch In- und InOut-Parametern mit Informationen versorgt und liefern ihr Ergebnis durch den Ergebnis- und mögliche Out-Parameter. Die Parameterliste enhält die Parameternamen, deren Typ und im Falle von In- und InOut-Parametern den Wert des aktuellen Parameters.

- **Den Ergebnisparameter (Result):** Dieser ausgezeichnete Parameter entspricht einem reinen Out-Parameter und dient der Einbettung von Dienstaufrufen als Funktionen (R-Value) in Ausdrücken. Er wird nur durch seinen Typ qualifiziert und kann einen entsprechenden Wert vom Dienst zum Klienten transportieren.

Der Anwender wird bei der Benutzung von Verteilungsinfrastrukturen typischerweise nicht mit dynamischen Aufträgen konfrontiert. Die Anwendungsgebiete dynamischer Aufrufe beschränken sich normalerweise auf die Erstellung von Brücken (engl.: *Bridges*) zu anderen Verteilungsinfrastrukturen oder aber verzögert-synchronen Aufrufen, die mit der synchronen Auftrag/Ergebnis-Semantik nicht bewerkstelligt werden können. Die Flexibilität der dynamischen Aufrufkomposition und die bereits existierenden Schnittstellen für Aufruf und Weiterleitung prädestinieren die dynamische Aufrufschnittstelle für die Konfiguration der Dienstgüte-Module. Um die Anzahl der vom Anwendungs- bzw. Dienstgüte-Implementierer benötigten Schnittstellen gering zu halten, wurde im Rahmenwerk eine Erweiterung der dynamischen Schnittstelle für die gesamte Kommunikation zwischen den Anwendungsobjekten und den Dienstgüte-Mechanismen gewählt. Dabei kann durch vorgegebene Schnittstellen der statischen Schnittstellenelemente und generierten Stubs, die den dynamischen Auftrag erstellen, unnötige Komplexität von Anwendungsprogrammierern ferngehalten werden.

Der Dienstgüte-erweiterte dynamische Auftrag (Quality of Service Dynamic Invocation Interface, QDII) wird um folgende Elemente ergänzt:

- **QoS-Parameter-Liste:** Die Steuerung eines bestimmten Dienstgüte-Niveaus kann zusätzliche Informationen bei jedem Auftrag an die Dienstgüte-Module notwendig machen, bspw. eine Priorität des Auftrages oder die Gruppenkennung einer Multicast-Gruppe. Bestimmte Dienstgüte-Charakteristiken erfordern die Rückgabe weiterer Informationen an die Anwendung bzw. an den Mediator. Für einen Mehrheitsentscheid ist nicht ein Ergebnis des Auftrages ausreichend, sondern eine bestimmte Menge von Ergebnissen muß an die Anwendung, bzw. eine Abstimmungsinstanz, übergeben werden. So ist eine zusätzliche Möglichkeit, Dienstgüte-bezogene Parameter zu transportieren, in den Auftrag zu integrieren. Die QoS-Parameter-Liste stellt sich dabei wie eine Parameterliste von Operationen dar.

- **Auftrag-Kennung:** Die Anwendung von dynamischen Aufträgen für die Kommunikation mit den Dienstgüte-Modulen macht es notwendig, eine Kennung zu integrieren. Diese Kennung gibt an, ob der Auftrag an den Dienst oder ein Dienstgüte-Modul bzw. den Dienstgüte-Transport gerichtet ist.

- **Auftragsziel:** Das Auftragsziel gibt im Falle eines Auftrages an die Dienstgüte-Mechanismen an, welches Dienstgüte-Modul den Auftrag erhalten soll. Ist kein Dienstgüte-Modul angeben und die Auftrags-Kennung für die Weiterleitung an die Dienstgüte-Mechanismen gesetzt, nimmt der Dienstgüte-Transport den Auftrag an.

An dieser Stelle sei angemerkt, daß für die Kommunikation mit den Dienstgüte-Mechanismen kein Stub-Objekt notwendig ist. Für die Integration in das Rahmenwerk muß nur sichergestellt werden, daß QDII-Aufträge den Dienstgüte-Transport erreichen. So kann auch die dienstseitige Dienstgüte-Implementierung auf der Anwendungsebene Zugriff auf die zugrundeliegenden Dienstgüte-Mechanismen erhalten.

Die Weiterleitung der QDII-Aufträge ist Aufgabe der Lokalisierungsfunktionalität des ORBs (vgl. Abbildung 6.3). Da diese von der ORB-Architektur abhängig ist, soll an dieser Stelle auf weitere Erläuterungen verzichtet werden. Abschnitt 6.5 beschreibt die Integration des hier vorgestellten Konzepts in Mikrokern-basierte und hierarchische ORBs.

6.4 Dienstgüte-Mechanismen-Hierarchie

In diesem Abschnitt wird die resultierende Dienstgüte-Mechanismen-Hierarchie vorgestellt. Dabei wird zunächst ein Überblick gegeben und anhand zweier Beispiele erläutert, wie Dienstgüte-Mechanismen für eine Dienstgüte-Charakteristik umgesetzt werden können.

6.4.1 Überblick

Abbildung 6.6 zeigt die Dienstgüte-Mechanismen im Rahmenwerk. Oberhalb der ORBs befinden sich die in Kapitel 5 beschriebenen Dienstgüte-Mechanismen der Anwendungsebene. In der Abbildung ist jeweils die Auswahl eines Dienstgüte-Mechanismus durch eine andere Schattierung angedeutet. Die Kommunikation zwischen Klient und Dienst erfolgt bei einem ausgehandelten Dienstgüte-Niveau durch den Mediator und die Dienstgüte-Implementierung der Dienstseite. Dabei wird der entsprechende Dienstgüte-Mechanismus von Stub bzw. Skeleton automatisch aufgerufen.

Die Übergabe eines Auftrages mit Dienstgüte-Befähigung geschieht auf der Klientenseite über die erweiterte dynamische Aufrufschnittstelle (QDII).

Die Lokalisierungs- und Weiterleitungsfunktionalität des ORBs leitet alle Aufträge mit Dienstgüte-Befähigung an den Dienstgüte-Transport weiter. Dieser prüft, ob der Klienten/Dienst-Beziehung ein Dienstgüte-Modul zugeordnet ist. Falls ja, wird der Auftrag an das entsprechende Modul weitergeleitet. Das Dienstgüte-Modul ist für die Durchführung des Transports von Aufträgen zuständig. Dienstgüte-Module können hier auf Dienstgüte-Mechanismen von Kommunikationsbibliotheken, des Netzwerks oder des Betriebssystems zurückgreifen.

Ist kein Dienstgüte-Modul zugeordnet, wird das IIOP-Modul benutzt, das die Interaktion zwischen Klient und Dienst erlaubt, wenn noch keine Dienstgüte-Vereinbarung getroffen wurde (zur Verhandlung) oder aber wenn die Dienstgüte-Charakteristik nur auf der Anwendungsebene implementiert wird, wie bei Lastbalancierung. Die Dienstgüte-Mechanismen der Anwendungsebene können über das IIOP-Modul notwendige Informationen austauschen, um die Dienstgüte-Module auf der Klienten- bzw. Dienstseite zu konfigurieren. Sind diese ordnungsgemäß initialisiert, kann durch die `assign`-Methode des Dienstgüte-Transports das entsprechende Dienstgüte-Modul der Klienten/Dienst-Beziehung zugewiesen werden.

Der Auftrag ist nun über das Netzwerk geschickt worden. Das sendende Dienstgüte-Modul wird auf der Dienstseite durch ein empfangendes Pendant ergänzt. Dieses nimmt den Auftrag entgegen. Dabei benutzt das empfangende Modul dasselbe Protokoll und denselben Netzwerktransport wie das sendende. Das Dienstgüte-Modul stellt nun den Auftrag in der ORB-internen Struktur bereit und übergibt diesen der Weiterleitungsfunktionalität. Diese übergibt den Auftrag an den Objekt-Adapter (OA), der über die Skeleton-Schnittstelle den Auftrag an den Dienst weiterleitet. Das Skeleton ruft nun vor der Dienst-Operation die Dienstgüte-Implementierung auf der Dienstseite auf (vgl. Abschnitt 5.4). Schließlich erreicht der Auftrag den Dienst. Nach der Bearbeitung des Auftrags nimmt die Antwort denselben Weg durch die Schichten zurück zum Klienten.

Bislang ist außer acht gelassen worden, wie die Konfiguration der Dienstgüte-Module und das Ineinandergreifen der Dienstgüte-Mechanismen realisiert wird. Der nächste Abschnitt betrachtet anhand zweier Beispiele die Kommunikation innerhalb der Dienstgüte-Mechanismen exemplarisch auf der Klientenseite.

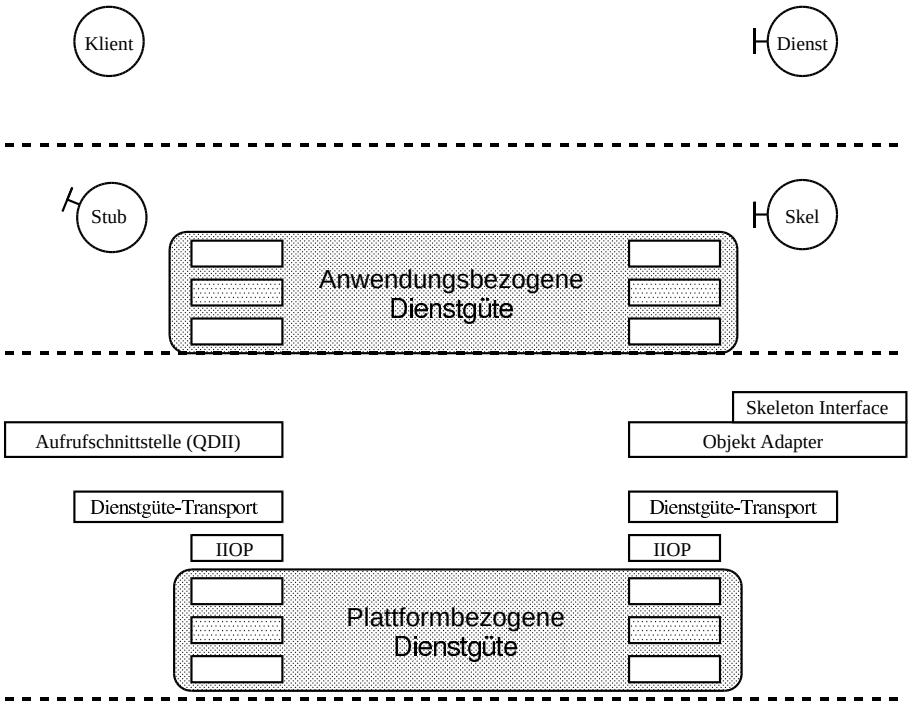


Abbildung 6.6: Dienstgüte-Mechanismen-Hierarchie im Rahmenwerk

6.4.2 Interaktion innerhalb der Dienstgüte-Mechanismen-Hierarchie

Die vorgestellten Schnittstellen des Dienstgüte-Transportes und die Konfiguration der Dienstgüte-Module sind bislang nur konzeptionell eingeführt worden. Dieser Unterabschnitt illustriert deren Nutzung und den Ablauf einer Dienstgüte-befähigten Interaktion im Rahmenwerk.

Beispielhafte Dienstgüte-Charakteristiken

Die folgenden Betrachtungen sollen in Hinblick auf zwei Dienstgüte-Charakteristiken geschehen: eine Dienstgüte-Charakteristik zur "Antwortzeit-Minimierung" durch Lastbalancierung und eine Dienstgüte-Charakteristik für "Verfügbarkeit" durch eine Replikatgruppe (Seite 99).

Lastbalancierung: Diese Dienstgüte-Charakteristik läßt sich durch Dienstgüte-Mechanismen der Anwendungsebene realisieren und benötigt kein Dienstgüte-Modul.

```
qos load
{
  // Kein Attribut

  interface{
  // Dienstgüte-Methoden der Klientenseite
    // Registrieren eines Dienstes
    void enter_server(in string ref);
    // Balancierungs-Richtlinie:
    //  bind per call: 0, initial bind: 1
    void set_policy(in short pol);
  // Dienstgüte-Methoden der Dienstseite
    // keine
  // Aspekt-Integration: Last ist dienstabhängig (aus [0,1])
    float get_load();
  };
};
```

Klienten registrieren Dienste bei der Dienstgüte-Implementierung durch die Methode `enter_server`. Die Binderichtlinie legt fest, ob bei jedem Aufruf oder nur beim ersten der Dienst mit der geringsten Last ausgewählt wird. Die Dienst/Aspekt-Integration ist hier die Lastinformation. So ist es möglich, die Dienstgüte-Implementierung für eine große Breite von Szenarien anzupassen. Der Dienst liefert die Last-Information entsprechend seiner Implementierung. Dies erlaubt eine differenzierte Implementierung und den Einbezug dienstabhängiger Informationen, wie die Länge der Auftragswarteschlange. Der Transport der Daten erfolgt über das IIOP-Modul des Dienstgüte-Transports.

Verfügbarkeit: Hier soll wieder auf das Beispiel aus Kapitel 5 zurückgegriffen werden. Dabei realisiert ein Dienstgüte-Modul einen Multicast-Algorithmus, der Reihenfolge-erhaltend Nachrichten an eine Dienstgruppe zustellt. Die Dienstgüte-Mechanismen der Anwendungsebene konfigurieren das Dienstgüte-Modul und stellen eine Verwaltungsfunktionalität bereit.

Dienstgüte-Mechanismen der Anwendungsebene

Abbildung 6.7 illustriert die Verarbeitung eines Auftrages auf der Klientenseite durch die Dienstgüte-erweiterten Einheiten. Ein Auftrag wird an das Stub-Objekt übergeben. Dieses überprüft, ob ein Mediator für eine Dienstgüte-Charakteristik installiert wurde. Ist dies nicht der Fall, wird der Auftrag direkt an den ORB weitergegeben. Der Stub verhält sich in diesem Fall wie ein Stub ohne Dienstgüte-Befähigung.

Ist ein Mediator für eine Dienstgüte-Charakteristik installiert worden, wird der Auftrag an den Mediator übergeben. Im Mediator wird geprüft, ob die aufgerufene Operation durch den Mediator bearbeitet oder aber ob der Auftrag weitergeleitet wird.

Im Falle der Dienstgüte-Charakteristik "Verfügbarkeit" existiert keine solche Operation. Alle Anfragen werden an den ORB und die Dienstgüte-Mechanismen der Transport-Ebene weitergeleitet. Hingegen wird die Lastbalancierung durch den Mediator durchgeführt. Dementsprechend sind die Konfigurationsoperationen auch vom Mediator zu bearbeiten. Die Binderichtlinie wird nicht an den ORB weitergeleitet, sondern beeinflußt die Auswahl, an welchen Dienst der Auftrag weitergeleitet wird. Die Menge der möglichen Dienste wird vom Klienten durch das Eintragen der Dienste mittels `enter_server` gebildet. Der Mediator bearbeitet den Auftrag, trägt den Dienst in seiner internen Datenstruktur ein und gibt den Auftrag als bearbeitet an den Stub und somit an den Klienten zurück. Dasselbe gilt für `set_policy`. Diese Methode regelt, wie der Mediator die Lastverteilung auf den Diensten vornimmt und hat somit für andere Einheiten keine Bedeutung. Die Dienstgüte-Vorkehrungen der Lastbalancierung werden im Mediator realisiert, ohne ein Dienstgüte-Modul im ORB zu nutzen. Die eingetragenen Dienste werden nach ihrer Last befragt und der Auftrag an den Dienst mit der geringsten Last geschickt. Dies geschieht durch Austausch des Ziel-Objektes im Auftrag. Der so modifizierte Auftrag wird dann an den ORB übergeben.

Beinhaltet der Auftrag keinen Befehl für den Mediator, sondern einen regulären Auftrag, muß der Mediator die entsprechenden Vorkehrungen für die Dienstgüte-Erbringung sicherstellen. Im Falle der "Verfügbarkeit" muß dem Dienstgüte-Transport initial mitgeteilt werden, mit welchem Dienstgüte-Modul die Klienten/Dienst-Interaktionen stattfinden. Dazu dient die `assign`-Methode des Dienstgüte-Transports. Der Mediator der Dienstgüte-

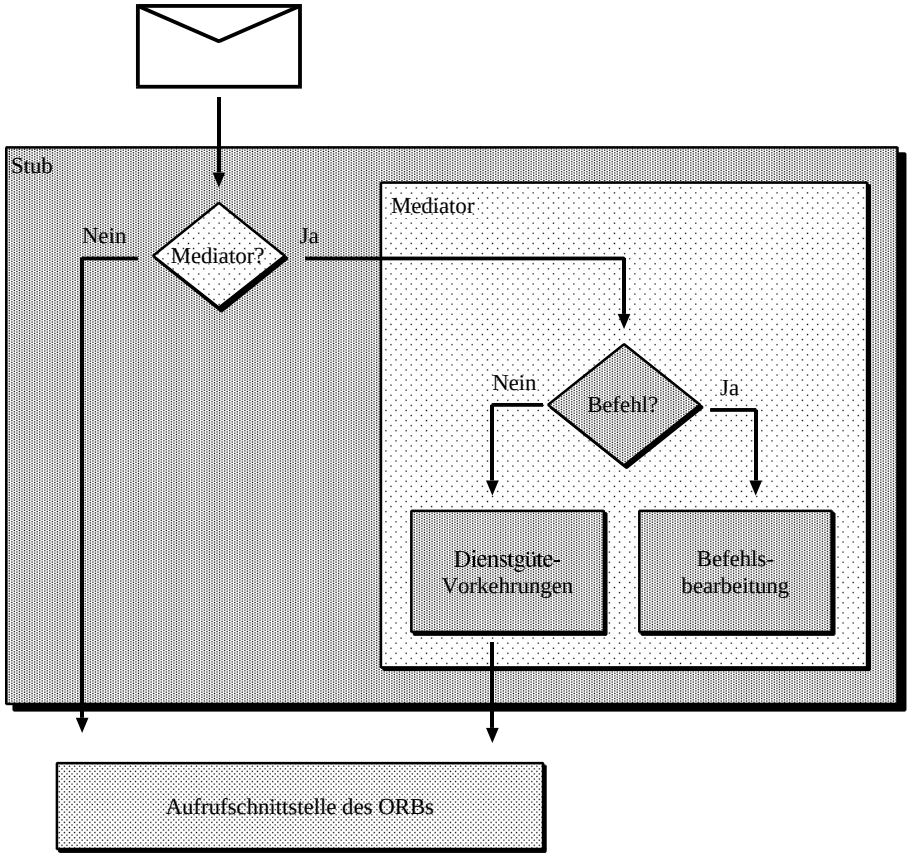


Abbildung 6.7: Durchlauf eines Auftrages durch den ORB – Stub/Mediator

Charakteristik "Verfügbarkeit" initialisiert die Verbindung, indem das entsprechende Modul geladen und dann zugewiesen wird. Das Gruppenkommunikations-Dienstgüte-Modul kann verschiedene Gruppen verwalten. Daher ist bei jedem Auftrag die entsprechende Gruppe als zusätzlicher Parameter anzugeben.

Der Ablauf eines Auftrages zwischen Dienst-Implementierung, Skeleton und der Dienstgüte-Implementierung auf der Dienstseite ist bereits in Kapitel 5 behandelt worden.

Dienstgüte-Transport

Ist der Auftrag an den ORB übergeben worden, muß er dem entsprechenden Zielobjekt zugestellt werden. Dazu ist die Lokalisierung und die Weiterleitung durch bestimmte Protokolle/Netzwerke notwendig. Die Lokalisierungseinheit des ORBs muß die Dienstgüte-befähigten Aufträge getrennt von den normalen Aufträgen behandeln. Abbildung 6.8 illustriert die beteiligten Komponenten innerhalb des ORBs.

Der Lokalisator ist in dem ersten Vergleich angedeutet. Dieser prüft, ob der übergebene Auftrag mit Dienstgüte-Befähigung versehen ist. Im wesentlichen ist dies die Hauptaufgabe der Integration des vorgestellten Modells in einen gegebenen ORB. Abschnitt 6.5 widmet sich dieser Aufgabe. Ein Auftrag ohne Dienstgüte-Befähigung wird dem GIOP/IIOP-Transport des ORBs übergeben, während ein Dienstgüte-befähigter Auftrag an den Dienstgüte-Transport weitergeleitet wird.

Der Auftrag kann auf vier verschiedene Arten interpretiert werden:

- **Auftrag ohne zugewiesenes Dienstgüte-Modul:** Der Auftrag ist an einen Dienst gerichtet, für dessen Klient/Dienst-Beziehung kein Dienstgüte-Modul im Dienstgüte-Transport zugewiesen wurde. Der Dienstgüte-Transport leitet solche Aufträge an das IIOP-Modul weiter, das den Auftrag an das entsprechende IIOP-Modul des Dienstgüte-Transports auf der Dienstseite weiterleitet. Dort wird der Auftrag entgegengenommen und an den Dienst über Objekt-Adapter und Skeleton weitergeleitet.

- **Auftrag mit zugewiesenem Dienstgüte-Modul:** Ist ein Dienstgüte-Modul zugewiesen, wird der Auftrag an das entsprechende Modul

zugestellt. Das Dienstgüte-Modul ist für Protokoll und Transport verantwortlich. An dieser Stelle erfährt der Dienstgüte-Implementierer die notwendige Freiheit, beliebige Protokolle und Transporte nutzen zu können. Der Zugriff auf zugrundeliegende Dienstgüte-Vorkehrungen, wie Bandbreitenreservierung auf der Netzwerkseite oder einer Bibliothek für Multicast-Kommunikation, ist hier möglich.

Das Dienstgüte-Modul für Verfügbarkeit sendet den Auftrag über einen Multicast an alle Dienste in der Gruppe. Dabei stellt das Multicast-Protokoll sicher, daß alle Dienste den Auftrag erhalten. Das Dienstgüte-Modul hat aber oberhalb dieses Transportes noch zwei wesentliche Aufgaben zu bewerkstelligen. Zum einen muß der Auftrag in ein geeignetes Format für die Übertragung konvertiert werden und zum anderen muß auf der Klientenseite die Abarbeitung der Aufträge bei den Diensten gewährleistet werden, bevor der Klient ein Ergebnis zugestellt erhält. Die Aufträge enthalten für das Dienstgüte-Modul relevante Informationen wie die Multicast-Gruppe in den Dienstgüte-Parametern des QDII-Auftrages.

Es ist wünschenswert, Dienstgüte-Module möglichst flexibel zu gestalten, so daß verschiedene Dienstgüte-Charakteristiken damit realisiert werden können. Im Falle des Dienstgüte-Moduls für die Verfügbarkeit durch Dienstgruppen könnte durch die zugrundeliegende Multicast-Kommunikation auch eine Dienstgüte für Diversität realisiert werden. Dabei müßten die Dienste unterschiedliche Implementierungen derselben Schnittstelle anbieten. Das Dienstgüte-Modul ist dann dafür verantwortlich, die entsprechenden Antworten an die Anwendung weiterzureichen, damit dort ein Mehrheitsentscheid – entweder im Mediator oder explizit durch den Klienten – stattfinden kann. Dazu können wiederum die Dienstgüte-Parameter des QDII-Auftrages benutzt werden.

Die hier vorgestellte Struktur unterstützt solche generischen Dienstgüte-Mechanismen auf der Dienstgüte-Modul-Ebene. Wesentlich für deren Anwendbarkeit ist eine leichte Konfiguration der Dienstgüte-Module.

- **Befehl an ein Dienstgüte-Modul:** Die Steuerung einer bestimmten Dienstgüte benötigt typischerweise eine Konfiguration, bevor die

Kommunikation stattfindet. Im Gegensatz zu den Dienstgüte-bezogenen Informationen, die im QDII-Auftrag mitgereicht werden, sind hier eigenständige Befehle für den Verbindungsaufbau notwendig. Dies kann die Reservierung von Bandbreite beim zugrundeliegenden Netzwerk-Transport im Falle einer bestimmten Durchsatzgarantie für eine Klient/Dienst-Beziehung sein oder aber die Konfiguration einer Dienstgruppe mit den Objektreferenzen der Dienste. Weiterhin sind für die Überwachung der Dienstgüte-Module bestimmte Operationen anzubieten. Da diese in einem starken Maße abhängig von der Funktionalität des Dienstgüte-Moduls sind, kann hier keine statische Schnittstelle angeboten werden.

Die Verwendung des QDII-Auftrages verspricht Abhilfe. Auf der Seite der Anwendung kann aus IDL-Definitionen ein QDII-Auftrag generiert werden. Das bedeutet, daß Anwendungen mit den Dienstgüte-Modulen interagieren, als ob es CORBA-Objekte wären. Das Dienstgüte-Modul hat Zugriff auf den Inhalt des QDII-Auftrages und kann die Operation und die Parameter extrahieren. Nach der Bearbeitung des Befehls wird der Auftrag mit eventuellen Ergebnissen wieder an den ORB zur Weiterleitung an die Anwendung übergeben.

- **Befehl an den Dienstgüte-Transport:** Die Befehle des Dienstgüte-Transportes sind fest vorgegeben (vgl. Abbildung 6.5). So könnte diese Schnittstelle durch ein Pseudo-Objekt realisiert werden. Da aber für die Weiterleitung von Befehlen an Dienstgüte-Module ohnehin der QDII-Auftrag benutzt wird, kann dieser auch für die Befehle an den Dienstgüte-Transport zum Einsatz kommen. Dazu wird ein Feld im QDII-Auftrag benutzt, welches das Ziel des Befehls angibt. Dies kann entweder eine Dienstgüte-Modul-Kennung sein oder aber leer gelassen werden und damit einen Befehl für den Dienstgüte-Transport angeben.

Die Bearbeitung von Aufträgen ist aus der Sicht des Klienten beschrieben worden. Durch die Objektreferenz kann der Lokalisator erkennen, daß der Auftrag an den Dienstgüte-Transport weiterzuleiten ist. Der Zugriff erfolgt für den Klienten über die Objektreferenz des Dienstes. Auf der Dienstseite ist dies gleichermaßen möglich. Der Dienst, bzw. die dienstseitig realisierten Dienstgüte-Implementierungen der Anwendungsebene, benötigen auch die Möglichkeit, Befehle an den Dienstgüte-Transport bzw. die

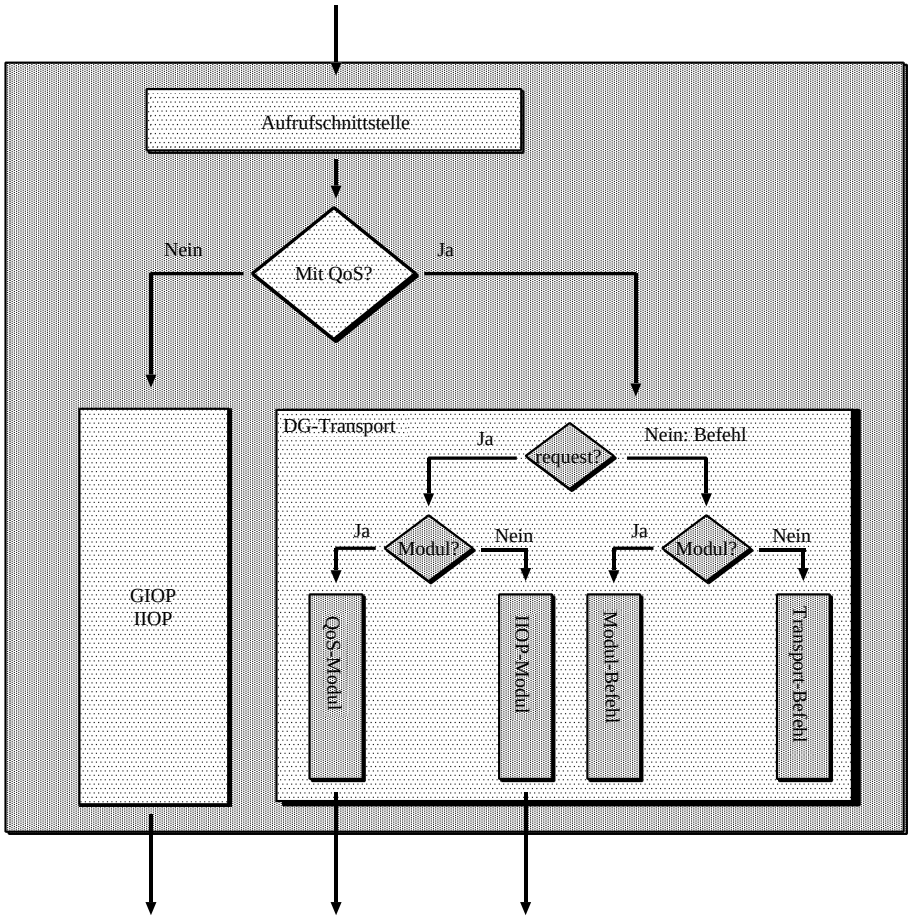


Abbildung 6.8: Durchlauf eines Auftrages durch den ORB – Dienstgüte-Transport

Dienstgüte-Module der Dienstseite abzusetzen. Dies kann einfach durch die Verwendung der Dienstobjektreferenz als Ziel des QDII-Auftrages geschehen.

Der hier vorgestellte Ansatz beruht auf der Erweiterung der dynamischen Aufrufschnittstelle. Mit derselben Abstraktion lassen sich von der Anwendung aus die unterschiedlichen Schnittstellen, wie sie in 6.2 motiviert wurden, abbilden. Der nächste Abschnitt behandelt die Integration des hier vorgestellten Konzeptes in verschiedene ORB-Architekturen.

6.5 Integration in verschiedene Architekturen

In diesem Abschnitt soll motiviert werden, wie sich das in diesem Kapitel vorgestellte Konzept zur Integration von Dienstgüte-Mechanismen zur Transportschicht in unterschiedliche ORB-Architekturen realisieren läßt. Dabei sollen die beiden in 6.1.2 vorgestellen grundsätzlichen Architekturvarianten, Mikrokern- und Hierarchie-basierte ORBs, betrachtet werden.

Die von dem Dienstgüte-Transport angebotene Schnittstelle entspricht der Annahme eines Auftrages für den Transport respektive der Weiterleitung des Auftrages an den Dienst. Typischerweise sind wenige Aussagen über den internen Aufbau einer Middleware möglich. Dies resultiert aus der Abstraktion, die eine solche Infrastruktur bereitstellt. Es sollen gerade solche Details verborgen werden und dem Implementierer der Infrastruktur wenige Vorgaben gemacht werden, damit die Architektur an die Zielplattform anpaßbar ist.

Im Kontext dieses Abschnitts steht die CORBA als Referenzinfrastruktur im Vordergrund. Auch für die CORBA gilt, daß in der Spezifikation keine Aussage über die Architektur der Implementierung gemacht wird. Die Programmierschnittstelle wird durch die CORBA-Spezifikation definiert – dies sichert die Portabilität von Anwendungen zu – und die Protokolle zwischen den ORBs sind spezifiziert – dies dient der Interoperabilität.

6.5.1 Integration in einen Mikrokern-basierten ORB

Der Prototyp des hier vorgestellten Rahmenwerkes ist auf der Basis von MICO entstanden. Der Mikrokern von MICO stellt im wesentlichen nur eine Weiterleitungsfunktionalität bereit. Dies bedeutet, der Kern bietet zwei Schnittstellen: eine Aufrufschnittstelle, um Aufträge anzunehmen, und

eine Weiterleitungsschnittstelle, um Aufträge weiterzuleiten. Dabei werden die verschiedenen Transporte (lokal, entfernt) als Dienste vom Mikrokern in Anspruch genommen. Symmetrisch zu solch einem Dienst muß auf der Partnerseite der Kommunikation eine Funktionalität zur Annahme des Auftrages und zur Weiterleitung an den lokalen ORB existieren (vgl. Abbildung 6.2).

Die Integration des Dienstgüte-Transportes ist bei einem Mikrokern-basiertem ORB recht einfach möglich. Der Dienstgüte-Transport wird als ein Transportdienst bei dem Mikrokern registriert. Die Lokalisierungs- und Weiterleitungsfunktionalität hat nur sicherzustellen, daß alle Dienstgüte-befähigten Aufträge an diesen Transport-Dienst weitergeleitet werden. Im Falle des MICO-Mikrokern basiert die Auswahl des Transport-Dienstes in Abhängigkeit von der Objektreferenz des Dienstes. Die Interoperablen Objektreferenzen (IORs) der CORBA sehen Erweiterungen vor. So können sog. Tags – also Kennzeichnungen – in einer IOR ergänzt werden. Der Dienstgüte-Transport nimmt alle IORs mit einem bestimmten Tag (`TAG_QOS`) an.

Die Integration des Dienstgüte-Transportes erfordert im Falle eines Mikrokerns keinerlei Modifikation am Kern. Lediglich die Generierung der Objektreferenzen mit der entsprechenden Kennzeichnung ist sicherzustellen. Abhängig von weiteren Implementierungsdetails des Mikrokerns sind allerdings für die Dienstgüte-Module besondere Vorkehrungen notwendig. So macht ein Mikrokern ohne die Unterstützung von Parallelität durch Threads die Integration der Dienstgüte-Module in dem Scheduler des ORBs notwendig. Ohne die Verwendung von Threads existiert eine zentrale Weiterleitung, die alle empfangsbereiten Kanäle bündelt. Im Falle von MICO bedeutet dies aber nur, daß eine Benachrichtigungsfunktion (Callback) und der Deskriptor des Empfangskanals des Dienstgüte-Moduls der Weiterleitung übergeben werden muß.

6.5.2 Integration in hierarchische ORBs

Als Beispiele für hierarchische ORBs sollen hier Orbix und ORBacus betrachtet werden. Die Flexibilität, die ein Mikrokern-ORB bietet, läßt sich auch mit hierarchischen ORBs erreichen. So bieten sowohl Orbix wie auch ORBacus die Möglichkeit, transportbezogenes Verhalten zu integrieren.

Die Orbix seit der Version Orbix 2000 zugrundeliegende Adaptive Runtime Technology (ART) [45] stellt ein Rahmenwerk für die ORB-Implementierung dar. Die notwendige Funktionalität für die Weiterleitung von Aufträgen wird durch sog. Plug-Ins realisiert. Ein Plug-In stellt dabei eine dynamisch ladbare Bibliothek dar, die durch eine wohldefinierte Schnittstelle mit dem ORB verbunden ist. Damit bieten Plug-Ins eine ähnliche Funktionalität wie die Dienstgüte-Module und der Dienstgüte-Transport. Die Integration des Dienstgüte-Transports könnte ohne größere Anpassung als Plug-In geschehen. Die notwendige Anpassung an den ORB – die Entgegennahme und Weiterleitung von Aufträgen – wird durch das Plug-In-Modell in Orbix unterstützt.

ORBacus unterstützt ein ähnliches Modell wie Orbix zur Integration von anwendungsspezifischen Transporten. Das Open Communications Interface (OCI) [76] bietet Unterstützung für Pluggable Protocols. Die Pluggable Protocols entsprechen den Plug-Ins der Orbix ART. Beliebige Protokolle können genutzt werden, nur muß das Pluggable Protocol eine übertragungssichere, verbindungsorientierte Sicht gegenüber dem ORB über der OCI-Schnittstelle anbieten. Die OCI hat Eingang in die CORBA-Realtime-Spezifikation [74] gefunden. Allerdings werden dort einige implementierungsabhängige Details der OCI von ORBacus nicht beachtet, was dazu führt, daß diese für eine konkrete Implementierung nicht ausreicht[1]. In der von ORBacus angebotenen OCI-Implementierung werden Pluggable Protocols durch einen Satz von Klassen realisiert, die durch ihre Schnittstelle gegeben sind. Ein Pluggable Protocol leitet von diesen Klassen die konkrete Implementierung ab. So ist für ein Pluggable Protocol eine Binde-Fabrik (Connector Factory), ein Binder (Connector), ein Transport und ein Akzeptor zu realisieren. Die Binde-Fabrik dient der Erzeugung der notwendigen Objekte. Binder und Akzeptor stellen die Schnittstellen gegenüber dem ORB dar. So ließe sich der Dienstgüte-Transport durch einen geeigneten Binder und Akzeptor in das OCI einbetten. Die entsprechenden Dienstgüte-Module realisieren dann die Transporte. Die Übergabe von Daten an das OCI geschieht allerdings durch Puffer, die bereits mit den konvertierten Daten eines Auftrages in Form einer Oktet-Sequenz gefüllt sind. Somit macht die Einbettung der Aufträge als Befehle an den Dienstgüte-Transport bzw. die Dienstgüte-Module eine weitere Anpassung nötig.

[1] In der Spezifikation wird auf diesen Mißstand explizit hingewiesen und auf die nächste Revision verwiesen.

Diese Anpassung kann durch sog. Interceptors geschehen, die einen Aufruf modifizieren können, bevor dieser dem ORB übergeben wird. Die Dienstgüte-Transport-Schnittstelle kann in Form eines Interceptors für die Befehle und als OCI-Schnittstelle für die Auftragsweiterleitung getrennt modelliert werden.

6.6 Zusammenfassung

Das in diesem Kapitel vorgestellte Modell zur Integration von Dienstgüte-Mechanismen in den Kern einer Verteilungsinfrastruktur bietet den Dienstgüte-Implementierungen auf der Anwendungsebene eine einheitliche Schnittstelle durch Aufträge. Dies erlaubt die Konfiguration und Überwachung der Dienstgüte-Mechanismen wie verteilte Objekte in der Verteilungsinfrastruktur.

Die Erweiterung der dynamischen Aufrufschnittstelle erlaubt einen einfachen Reflektions-Mechanismus für das dynamische Nachladen von Dienstgüte-Funktionalität durch dynamische Bibliotheken. Die Kapselung der Dienstgüte-Module durch den Dienstgüte-Transport ermöglicht die einfache Einbettung in andere ORB-Architekturen.

Die durch den Dienstgüte-Transport realisierte Einbettung von Dienstgüte-Mechanismen in den ORB unterstützt so die Ende-zu-Ende-Dienstgüte-Erbringung von der Dienstgüte-Implementierung der Anwendungsebene bis zu Dienstgüte-Mechanismen der zugrundeliegenden Plattform. Dabei bieten die Dienstgüte-Module eine Abstraktion, die es erlaubt, anwendungsorientierten Dienstgüte-Mechanismen auf unterschiedliche Implementierungen abhängig von der Plattform zuzugreifen. So kann die Bandbreiten-Reservierung auf einem ATM- und einem RSVP-Netzwerk von Dienstgüte-Modulen mit derselben Schnittstelle angeboten werden und somit die anwendungsbezogenen Dienstgüte-Implementierung ohne deren Anpassung unterstützen.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Integration von Dienstgüte-Vorkehrungen in objektorientierte Verteilungsinfrastrukturen befähigt Anwendungsentwickler, den Verteilungs-induzierten Problemen verteilter Systeme zu begegnen. Im Rahmen dieser Arbeit wurde die generische Einbettung von Dienstgüte-Vorkehrungen in verteilte Objektsysteme untersucht und ein Lösungsansatz präsentiert.

Zunächst wurde eine Analyse der für das Dienstgüte-Management notwendigen Aufgaben vorgestellt. Ausgehend von einem verteilten Objektmodell wurde untersucht, wie Dienstgüte-Vorkehrungen integriert werden können. Dienstgüte-Vorkehrungen stellen bei einem zugrundeliegenden Objektmodell nicht-einkapselbare Verantwortlichkeiten dar. Die enge Bindung der Dienstgüte-Vorkehrungen an einen Dienst führt so zu Vermaschungen in den Strukturen der Implementierung. Damit ist die getrennte Wiederverwendung beider erschwert. Zusätzlich werden unterschiedliche Abstraktionen vermischt. Die aspektorientierte Programmierung (AOP) behandelt solche Vermaschungen. Dienstgüte wurde bei der Integration in ein verteiltes Objektmodell als ein Aspekt im Sinne der AOP klassifiziert.

Ausgehend von den Anforderungen an das Dienstgüte-Management wurde ein Rahmenwerk auf Basis eines verteilten Objektmodells entworfen. Der in dieser Arbeit dargestellte Schwerpunkt liegt auf der Spezifikation von Dienstgüte-Charakteristiken und deren Umsetzung in die Implementierungssprache der Anwendungsobjekte. Für die Unterstützung der Ende-zu-Ende-Dienstgüte-Erbringung ist der Einbezug von Dienstgüte-Vorkehrungen des Netzwerks, Betriebssystems oder spezieller Bibliotheken notwendig.

Die resultierende Hierarchie von Dienstgüte-Mechanismen wird durch die vorgestellte Integration in eine Verteilungsinfrastruktur unterstützt.

Durch die Integration der Dienstgüte-Spezifikation in die Schnittstellenbeschreibungssprache erlaubt das Rahmenwerk einen aspektorientierten Ansatz ohne die Einführung weiterer Sprachen zur Spezifikation oder Implementierung. Die Spezifikation von Dienstgüte-Charakteristiken in der erweiterten IDL wird in spezielle Entwurfsmuster in der Zielsprache umgesetzt. Diese Entwurfsmuster separieren die Anwendungsobjekte weitgehend von den Dienstgüte-Vorkehrungen.

Die auf der Ebene der Anwendungsobjekte generierten Vorlagen für die Dienstgüte-Vorkehrungen können durch einen modifizierten bzw. schon dafür ausgelegten Verteilungsinfrastrukturkern in das System integriert werden. Eine einheitliche statische Schnittstelle erlaubt einen einfachen reflektiven Ansatz. So ist der Zugriff auf Dienstgüte-Vorkehrungen tieferer Schichten wie auch die Integration anwendungsspezifischer Dienstgüte-Vorkehrungen auf der Netzwerkschicht möglich.

Das Rahmenwerk bietet somit eine klare Trennung der Verantwortlichkeiten, die sowohl Anwendungsentwickler wie auch Dienstgüte-Implementierer unterstützt. Die aus der Schnittstellenbeschreibungssprache generierten Einheiten stellen für die Anwendungsobjekte eine Abstraktion dar, die sowohl die Verteilungsaspekte wie auch die Dienstgüte-Vorkehrungen einfach nutzbar anbietet und von der zugrundeliegenden Plattform isoliert.

7.2 Weitere Fragestellungen und Ausblick

Eine sich aus dieser Arbeit ergebende Fragestellung besteht in der Erweiterung und Verallgemeinerung des aspektorientierten Ansatzes. Die im Rahmen der Analyse betrachteten Dienstgüte-Charakteristiken sind aus dem systemnahen Bereich und insbesondere aus der Betrachtung typischer Probleme in verteilten Systemen und den daraus erwachsenen Anwendungsanforderungen gewonnen. Nicht-funktionale Aspekte der Dienserbringung lassen sich weiter fassen. So kann ausgehend von den bereitgestellten Abstraktionen untersucht werden, inwieweit auf Anwendungsebene nicht-funktionale Eigenschaften in ähnlicher Weise einbettbar sind. Im Rahmen dieser Arbeit wurde beispielsweise eine Dienstgüte-Charakteristik zur Parallelisierung von Berechnungen realisiert. Eine anwendungsbezogene

Dienstgüte-Charakteristik könnte numerische Optimierungen realisieren, die von den reinen mathematischen Operationen zu trennen ist. Andere Beispiele aus der Multimedia-Kategorie sind durch die Qualität einer Audio-Übertragung gegeben. So kann bei einer geringen Bandbreite durch die Kompression der Daten eine bessere Qualität der Audiowiedergabe ereicht werden, als durch Übertragung der Rohdaten. Die Kompressionsrate kann von der Anwendung isoliert und durch entsprechende Dienstgüte-Mechanismen realisiert werden. Qualitätsunterschiede ergeben sich durch mögliche verlustbehaftete Kompression und de notwendigen Anforderungen an Hardware- oder Software-Unterstützung. Andere Kriterien für die Qualität lassen sich weniger leicht vor der Anwendung verbergen. Die Wiedergabe von Stereo- oder Mono-Audiodaten erfordert entsprechende Anwendungen und auch Ausstattungen der Endgeräte.

Im Kontext dieser Arbeit wurde ein Objektmodell betrachtet, das eine starke Bindung zwischen Schnittstellen und Objekten besitzt. Insbesonderen wurde bei der Umsetzung der Schnittstellenbeschreibungssprache in die Zielsprache eine Umsetzung gewählt, die Dienste als Objekte repräsentiert. Involviert die Diensterbringung verschiedene Objekte, kann nur ein Objekt als Stellvertreter all dieser Dienste den Service anbieten. Dieses Objekt ist für die Einhaltung von Dienstgüte-Vereinbarungen mit Klienten verantwortlich. Innerhalb der Objekte, die den Service realisieren, sind für die Dienstgüte-Erbringung dann ggf. weitere interne Dienstgüte-Vorkehrungen zu etablieren. Komponentenmodelle versprechen hier einen allgemeineren Ansatz, der die Integration von Dienstgüte-Vorkehrungen lohnenswert erscheinen läßt. Zum einen unterstützen Komponentenmodelle definierte Schnittstellen zur Interaktion zwischen den beteiligten Objekten einer Komponente, und zum anderen bieten Komponenten eine über die Schnittstellenbeschreibungssprache hinausgehende Beschreibung ihrer Funktionalität in einer Komponentenspezifikation. Diese Komponentenspezifikation verspricht einen guten Ansatz, um Dienstgüte-Spezifikationen der Komponenten zu integrieren.

Neben den beiden bislang beschriebenen Forschungsrichtungen, die jeweils ein Rahmenwerk für das Dienstgüte-Management voraussetzen und darauf aufbauen, existieren innerhalb des in der Arbeit vorgestellten Rahmenwerkes weitere offene Forschungsfragen. Die Ausgestaltung von Preisen bei der Vergabe von Ressourcen und die damit verbundenen Richtlinien für die Vergabe und auch den Entzug stellen noch kein abgeschlossenes

Gebiet dar. Hier ist der Einbezug anderer Disziplinen vielversprechend. Preisrichtlinien für manche Ressourcen, die bei Nicht-Nutzung verfallen – wie Netzwerkkapazität – sind Gegenstand der Forschung in der Betriebswirtschaftslehre. Die Gestaltung von Vergaberichtlinien, insbesondere aber die Festlegung von Vergütungen bei Nichterbringung eines festgesetzten Dienstgüte-Niveaus oder Kompensationen bei dem Entzug von Ressourcen mit einer damit einhergehenden Verletzung der Dienstgüte-Vereinbarung, wirft rechtliche Fragen über die Gültigkeit solcher Richtlinien auf.

Weitere, nicht-interdisziplinäre Fragestellungen, ergeben sich aus der Frage der Wiederverwendbarkeit und Dokumentation von Dienstgüte-Vorkehrungen im Rahmenwerk. Die Erstellung eines Katalogs mit einem einheitlichen Aufbau – wie es bei Entwurfsmustern üblich ist – verspricht eine geeignete Dokumentationsform. Allerdings muß eine solche Dokumentation zwei Zielgruppen gerecht werden. Zum einen sind dies Anwendungsentwickler, die eine gegebene Dienstgüte-Implementierung anwenden wollen und Informationen für die Nutzung und Anpassung der Anwendung benötigen und zum anderen Dienstgüte-Entwickler, die auf bereits existierende transportspezifische Dienstgüte-Mechanismen aufbauen.

Für die hier skizzierten Forschungsrichtungen ist ein Rahmenwerk für das Dienstgüte-Management unerläßlich. Das in dieser Arbeit vorgestellte Rahmenwerk bietet eine gute Ausgangsbasis.

Anhang A

QIDL

A.1 Grammatik der QIDL-Definitionen

```
    <definition>    ::=     <type_dcl> ";"
                    |       <const_dcl> ";"
                    |       <except_dcl ";"
                    |       <interface> ";"
                    |       <module> ";"
1)                  |       <qos_dcl> ";"

2)  <qos_dcl>       ::=     "qos" <identifier> <qos_inheritance>
                            <qos_body>

3)  <qos_inheritance>
                    ::=     [ "extends" <qos_baseclass_list> ]

4)  <qos_baseclass_list>
                    ::=     <identifier> { "," <identifier }

5)  <qos_body>      ::=     "{" <qos_attr_list> "interface"
                                                "{" qos_op_list "}" ";"
                            "}"

6)  <qos_attr_list> ::=     <qos_attr>
                    |       <qos_attr_list> <qos_attr>
```

```
7)  <qos_op_list>    ::=     <op_dcl>*

8)  <qos_attr>       ::=     <param_type_spec> <identifier> ";"

9)  <interface_header>
                     ::=     "interface" <identifier>
                             [<inheritance_spec>] [<qos_spec>]

10) <qos_spec>       ::=     "withQoS" <identifier>
                             { "," <identifier> }
```

Die hier beschrieben Grammatik stellt nur die Änderungen an der OMG-IDL, wie sie in [69] beschrieben ist, dar.

1. **Dienstgüte-Definition:** Eine Dienstgüte-Definition kann an jeder Stelle in der IDL auftreten, an der andere Definitionen, wie die für Schnittstellen oder Module, erlaubt sind. Insbesondere lassen sich Dienstgüte-Definitionen auch in Modulen oder Schnittstellen einbetten.

2. **Dienstgüte-Deklaration:** Eine Dienstgüte-Definition wird durch das qos-Schlüsselwort eingeleitet und durch die Dienstgüte-Deklaration an einen Dienstgüte-Körper und eventuelle Basis-Schnittstellen durch die Dienstgüte-Schnittstellen-Vererbung gebunden.

3. **Dienstgüte-Schnittstellen-Vererbung:** Durch die Angabe von Basis-Dienstgüte-Schnittstellen werden Attribute und Operationen der Basis-Dienstgüten in die aktuelle Dienstgüte-Schnittstelle übernommen.

4. **Basis-Dienstgüte-Liste:** Die Basis-Dienstgüte-Liste wird für die Angabe verschiedener Dienstgüte-Bezeichner nach `extends` benötigt.

5. **Dienstgüte-Körper:** Innerhalb der Dienstgüte-Deklaration werden die Dienstgüte-Parameter und die Operationen der Dienstgüte-Mechanismen bzw. der Aspekt-Integration vereinbart.

6. **Dienstgüte-Attributliste:** Zusammenfassung aller Dienstgüte-Parameter einer Dienstgüte-Deklaration. Diese werden aus einem Typ und dem Namen gebildet (vgl. 8).

7. **Dienstgüte-Operationsliste:** Die Operationen der Dienstgüte-Deklaration werden wie die Operationen in einer anderen Schnittstelle der IDL deklariert.

8. **Dienstgüte-Attribute** bilden die Dienstgüte-Parameter ab. Es dürfen beliebige Typen der IDL – auch benutzerdefinierte – verwendet werden.

9. **Schnittstellen-Kopf:** Der Kopf einer Schnittstellen-Deklaration in der IDL wird um die Dienstgüte-Zuordnung erweitert.

10. **Dienstgüte-Zuordnung:** Durch die Dienstgüte-Zuordnung können verschiedene, bereits deklarierte Dienstgüte-Schnittstellen an eine Schnittstelle gebunden werden.

A.2 Generierte Funktionen des Rahmenwerks

Das Rahmenwerk unterstützt für die Dienstseite eine Schnittstelle, die jeder Dienstgüte-befähigte Dienst anbietet. Diese Schnittstelle erlaubt Klienten die Abfrage der unterstützten Dienstgüte-Charakteristiken und eine einfache Verhandlung. Klienten können prüfen, ob ein Dienst diese Schnittstelle anbietet und über das Rahmenwerk weitere Informationen über die angebotenen Dienstgüte-Charakteristiken einholen.

```
1: typedef Any QoSParam;
2: typedef sequence<QoSParam> QoSParams;
3: interface QoSSrvFW {
4:   QoSParams GetOfferedQoS();
5:   void Negotiate(inout QoSParam desiredQoS );
6:   bool Accept(in QoSParam offeredQoS);
7:   void Notification( in QoSParam actualQoS);
8: };
```

Basis der Dienstschnittstelle ist der Austausch von Dienstgüte-Attributen, wobei diese die Repräsentation von Dienstgüte-Parametern als Struktur in der Zielsprache darstellen. Der CORBA-Typ `Any` kann jeden anderen

Typ aufnehmen und bietet sich so als generischer Austauschcontainer an (Zeile 1, 2). Die Operation `GetOfferedQoS` liefert eine Liste von Dienstgüte-Attributen. Dabei wird für jede unterstützte Dienstgüte-Charakteristik ein Attribut zurückgegeben (Zeile 4). Klienten können ein Dienstgüte-Attribut mit den gewünschten Zielwerten füllen und über die `Negotiate`-Operation dem Dienst übergeben (Zeile 5). Der Dienst gibt denselben Wert zurück, wenn er das angeforderte Dienstgüte-Niveau erbringen kann. Durch `Accept` wird die Vereinbarung manifestiert und die entsprechenden Dienstgüte-Mechanismen auf Klienten- und Dienstseite installiert (Zeile 6). Bei Verletzung der Vereinbarung kann der Dienst mittels der `Notification`-Operation benachrichtigt werden (Zeile 7).

```
interface QoSCLFW {
  void Notification(in QoSParam actualQoS);
};
```

Auf der Klientenseite existiert nur eine Operation zur Benachrichtigung, wenn eine Dienstgüte-Vereinbarung verletzt wurde. Dies erlaubt dem Klienten die Anpassung durch eine Wiederverhandlung oder den Abbruch der Interaktion. Der QIDL-Compiler generiert für die Operationen des Rahmenwerks Implementierungsvorlagen aus den Beschreibungen der QIDL.

Anhang B

Anwendung des Rahmenwerks

In diesem Abschnitt wird eine Dienstgüte-Charakteristik zur Verbesserung der mittleren Antwortzeit durch Lastbalancierung und ihre Integration in zwei Anwendungen präsentiert.

B.1 Die Dienstgüte-Charakteristik "Load"

Die hier vorgestellte Dienstgüte-Charakteristik dient zur Reduzierung der mittleren Antwortzeit von Anfragen durch Lastbalancierung auf einem Dienst-Ensemble. Die Dienste bieten denselben Service über eine Schnittstelle an. Die Realisierung des Dienstgüte-Mechanismus erfolgt auf der Klientenseite durch den Mediator. Dort werden alle Dienste registriert. Der Dienstgüte-Mechanismus bietet zwei Balancierungsrichtlinien an. Durch einen Initial Bind wird der Dienst mit der aktuell geringsten Last an den Klienten für die gesamte Interaktion gebunden. Alternativ kann bei jedem Dienstaufruf der Dienst mit der geringsten Last ermittelt werden und der Aufruf an diesen weitergeleitet werden.

Die Lastinformation ist vom Dienst beizusteuern. Damit können verschiedene Arten von Lastinformationen integriert werden, die über eine reine Last des Rechners hinausgehen.

```
1:  qos load
2:  {
3:
4:    interface{
5:    // client qos methods
```

```
6:      // register a servers stringified IOR
7:      void enter_server(in string ref);
8:      // set policy: bind per call: 0, initial bind: 1
9:      void set_policy(in short pol);
10:   // server/qos integration methods
11:      // retrieve load information in [0,1]
12:     float get_load();
13:    };
14:  };
```

Die Dienstgüte-Schnittstelle der Dienstgüte-Charakteristik "Load" beinhaltet keine Dienstgüte-Parameter. Der Zustand wird durch die Anzahl der Dienste und die eingestellte Balancierungsrichtlinie gebildet und ist für die Anwendung nicht von Belang.

Die Operationen der Dienstgüte-Schnittstelle bestehen nur aus Lastinformation. Diese könnte entweder von der Dienstgüte-Implementierung bereitgestellt werden, dann hätte ein Dienst keine Verantwortlichkeiten zur Dienstgüte-Befähigung zu erbringen, oder aber durch den Dienst angepaßt werden und somit ein hohes Maß an unterschiedlichen Lastinformationen zulassen.

```
1:  class load_Mediator : public MAQSMediator
2:  {
3:   private:
4:    CORBA::Short load_policy; // per call oder per initial
5:    int server_selected;      // bind ist schon ein server
6:                              // gebunden? 1 ja, 0 nein
7:    CORBA::Object *target_server;
8:      // Zielserver bei initial bind
9:    vector <CORBA::Object_ptr> servers;
10:
11:   public:
12:   CORBA::ORB_ptr orb;
13:     load_Mediator(void);
14:     ~load_Mediator();
15:     virtual void invoke( MAQS::qRequest *req );
16:  };
```

Die Dienstgüte-Implementierung beschränkt sich weitestgehend auf die Klientenseite. Der `load_Mediator` ist für die Verwaltung der Dienste und die Verteilung der Aufrufe zuständig. Die vom QIDL-Compiler generierte Vorlage wird um die notwendige Verwaltungsfunktionalität erweitert. Dazu muß die Binderichtlinie (Zeile 4), die Information, ob schon ein Dienst gebunden wurde (Zeile 5), und die Liste der Dienste (Zeile 9) vorgehalten werden.

```
1:  void load_Mediator::invoke( MAQS::qRequest *req )
2:  {
3:    if( strcmp(req->operation(), "set_policy")==0 )
4:      *(req->arguments())->item( 0 )->value() >>= load_policy;
5:      if (load_policy == 1)
6:        server_selected = 0;  // Server muss bestimmt werden
7:    }
8:    else if( strcmp(req->operation(), "enter_server")==0 )
9:      // Server-Adresse aus den Paramtern extrahieren und
10:   } // in die Liste eintragen
11:   else
12:   {
13:     // Dies ist nun ein regulärer Dienstaufruf. Somit muß
14:     // entsprechend der Binderichtlinie ein Dienst ausge-
15:     // wählt und ein neuer Auftrag (new_req) erzeugt werden
16:     new_req->invoke(); // Dienstaufruf
17:   }
18: }
```

Konstruktor und Destruktor sind im Mediator entsprechend anzupassen, damit die Dienstgüte-spezifischen Einheiten ordnungsgemäß initialisiert bzw. freigegeben werden. Im Mediator wird nur eine Methode (`invoke`) für die Erbringung der Dienstgüte-Charakteristik benötigt. Diese wird vom QIDL-Compiler bereits in ihrem Gerüst erzeugt. Die `invoke`-Methode wird vom Stub aus aufgerufen, wenn ein Mediator installiert ist. Der Mediator übernimmt den Auftrag und prüft, welche Operation vorliegt. Dabei besteht nun die Möglichkeit, vor und nach jedem Aufruf bestimmte Vorkehrungen zu treffen. Im Falle der Lastbalancierung werden die Operationen für die Dienstgüte-Steuerung (`enter_server`, `set_policy`) im Mediator bearbeitet. Die Angabe einer Binderichtlinie ist in Zeile 3 zu sehen.

Der Operationsname ist im Auftrag als Text gegeben. Bei einem positiven Stringvergleich wird die Methode bearbeitet. In Zeile 4 wird das Argument – die Richtlinie – aus dem Auftrag extrahiert und in `load_policy` gespeichert. Bei einem Initial Bind wird die Variable `server_selected` auf Null gesetzt, um ein erneutes Binden zu erzwingen. Bei `enter_server` ist nur die Auswahl der Methode illustriert, um das vom QIDL-Compiler generierte Schema zu verdeutlichen. Hier muß der Dienstgüte-Implementierer entsprechend die Parameter extrahieren und in der Liste der Dienste den übergebenen Dienst speichern. Sind alle Dienstgüte-bezogenen Operationen abgearbeitet, kann davon ausgegangen werden, daß im Auftrag eine normale Dienstanfrage vorliegt (Zeile 13 ff.). Hier kann nun das Dienstgüte-spezifisches Verhalten integriert werden. Im Falle der Lastbalancierung wird nur vor dem Aufruf Verhalten benötigt, das den entsprechenden Dienst mit der geringsten Last bestimmt. Bei der Binderichtlinie Initial Bind geschieht dies nur das erste Mal. Ist der entsprechende Dienst gefunden, wird ein neuer Auftrag mit diesem Dienst als Zielobjekt erzeugt. Dieser Auftrag wird durch Aufruf der `invoke`-Methode dem ORB zur Weiterleitung übergeben (Zeile 16).

In Abbildung B.1 sind die unterschiedlichen Aufgaben bei der Entwicklung einer Anwendung mit Dienstgüte-Befähigung im MAQS-Rahmenwerk dargestellt. Im folgenden soll kurz darauf eingegangen werden, bevor die Integration der Lastbalancierung in zwei Anwendungen vorgestellt wird.

Die Aufgaben sind in drei Bereiche geteilt, die durch entsprechende Rollen adressiert werden.

Der *Client Designer* ist für Entwurf und Implementierung des Klienten eines Dienstes verantwortlich. Dabei greift dieser auf den generierten Code des QIDL-Compilers zurück und entwirft darauf aufbauende Objekte. Diese werden übersetzt und mit dem Stub-Code gebunden. Im Falle der Dienstgüte-Befähigung wird zusätzliche Funktionalität für das Rahmenwerk, sowie die Dienstgüte-Implementierung der Klientenseite gebunden.

Der *Object Designer* realisiert den Dienst. Dazu wird eine Schnittstellenbeschreibung des Dienstes erstellt und durch den QIDL-Compiler in die entsprechenden Vorlagen der Zielsprache übersetzt. Die Dienstfunktionalität wird durch die Implementierung des Skeletons auf der Dienstseite bereitgestellt.

Dies entspricht im wesentlichen der Vorgehensweise ohne Dienstgüte-Befähigung. Im Rahmenwerk wird die zusätzliche Rolle des *Dienstgüte-*

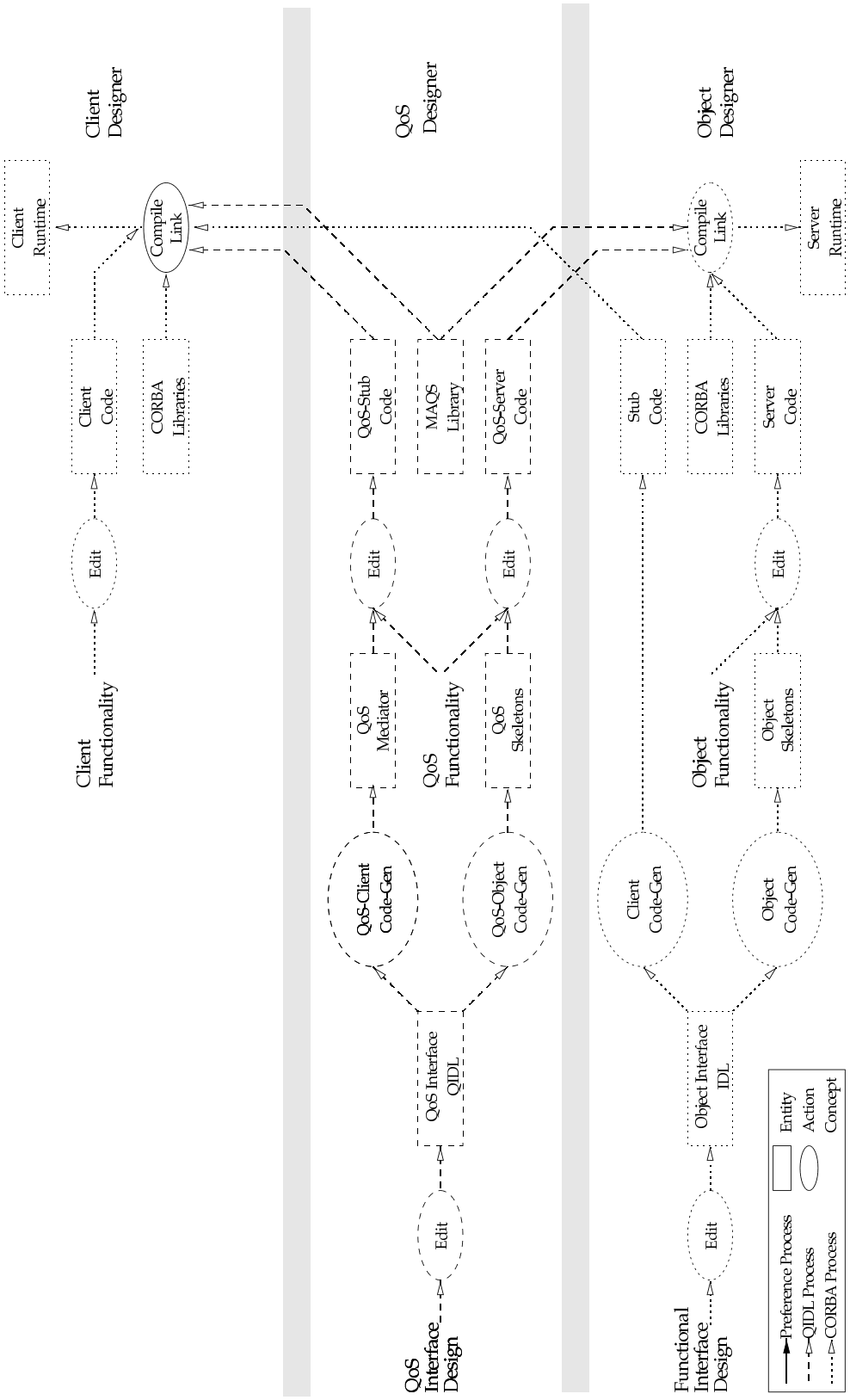


Abbildung B.1: Aufgaben während der Entwicklung

Implementierers (QoS Designer) unterstützt. Der Dienstgüte-Implementierer ist für die Schnittstelle der Dienstgüte sowie der Implementierung der klienten- und dienstseitigen Dienstgüte-Mechanismen der Anwendungsschicht verantwortlich. Dabei kann die Implementierung von Dienstgüte-Mechanismen des ORBs notwendig sein.

Die Anpassung einer Anwendung an eine Dienstgüte-Charakteristik hängt wiederum von der jeweiligen Dienstgüte-Implementierung ab. Prinzipiell könnte eine Dienstgüte-Charakteristik ohne weitere Anpassung von Klient oder Dienst realisiert sein. Oftmals ist allerdings eine Anpassung notwendig. So könnte die Konfiguration der Dienste im Lastbalancierungsbeispiel auch durch die Ressourcen-Steuerung geschehen und die Lastinformation durch die Dienstgüte-Implementierung der Dienstseite bereitgestellt werden.

Im folgenden soll anhand zweier Anwendungen die Integration der Lastbalancierung demonstriert werden.

B.2 Die Mandelbrot-Anwendung

Die in Abbildung B.2 dargestellte Mandelbrotmenge[1] wird durch eine iterative Berechnung der Konvergenz einer Funktion auf der komplexen Ebene gebildet. Da die Berechnungen zeitintensiv sind, kann eine leistungsbezogene Dienstgüte zu einer Verbesserung der Dienstqualität des Apfelmännchens führen.

Die Berechnung der Mandelbrotmenge wird durch die folgende Gleichung durchgeführt.

$z_n = {z_{n-1}}^2 - c \; ; \; z_i \in C, c \in C$ const

Diese Iteration wird für jeden Punkt in dem betrachteten Teilbereich der komplexen Ebene durchgeführt. Die Färbung der Punkte ergibt sich aus der Konvergenzgeschwindigkeit.

Der Apfelmännchen-Dienst bietet diese Berechnung durch die folgende IDL-Schnittstelle an. Die `calc`-Operation erhält den Auftrag, eine Zeile in der Menge zu berechnen. Dabei ist die Zeile durch einen Punkt in der komplexen Ebene und deren Ausdehnung gegeben. Zur Berechnung wird noch die Projektion auf die Anzahl der Punkte in der Darstellung mitgereicht.

[1] Benannt nach dem Mathematiker B. Mandelbrot. Oft wird wegen der Form auch der Begriff "Apfelmännchen" verwendet.

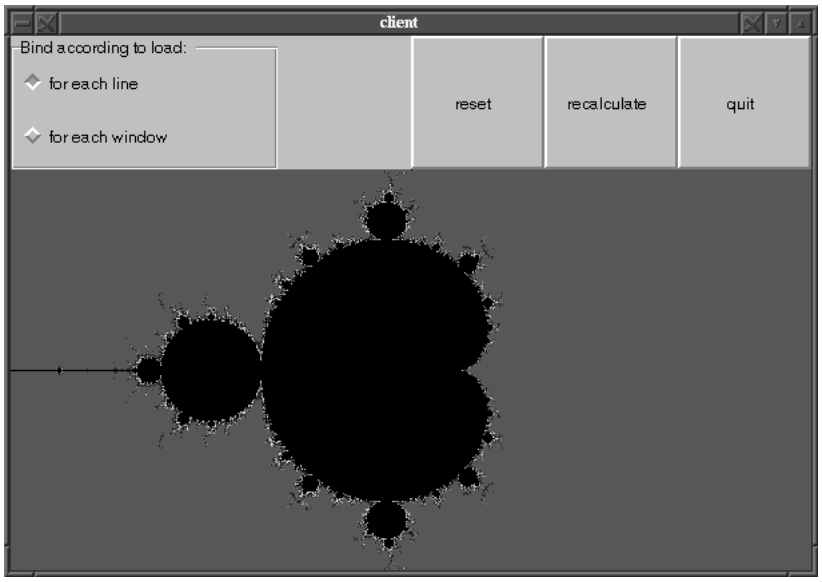


Abbildung B.2: Die Mandelbrot-Menge

Als Ausgabe wird eine Sequenz mit ganzzahligen Werten geliefert, die als Farbkodierung verwendet werden kann.

```
typedef sequence< short > apfel_line;
interface apfel
{
  void calc( in double x,         // startx
             in double y,         // starty
             in double dist,      // Länge in der Menge
             in short  points,    // Anzahl Darstellungspunkte
             out apfel_line ret); // Augabe: Farbkodierung
};
```

Die Dienstgüte-Befähigung des Dienste mit der Lastbalancierung geschieht durch die folgende QIDL-Definition. Am Anfang wird die Schnittstellenbeschreibung des Dienstes und der Dienstgüte inkludiert. Die Dienstgüte-Zuordnung geschieht bei der `load_apfel`-Schnittstelle, die von der

apfel-Schnittstelle erbt. Damit bleibt der ursprüngliche Apfeldienst erhalten.

```
#include "load.idl"
#include "apfel.idl"

interface load_apfel : apfel withQoS load
{
};
```

Aus dieser Beschreibung werden die folgende Klassen generiert: die Klasse `load_apfel` ist die gemeinsame Basisklasse für das Stub und das Skeleton. Die `load_apfel_stub`-Klasse stellt den Stub für den Dienstgüte-befähigten Dienst dar. Dieser bietet die Schnittstellen der Apfel-Schnittstelle und der Lastbalancierungs-Dienstgüte an. Das Skeleton `load_apfel_skel` stellt die Implementierungsvorlage für den Dienstgüte-befähigten Dienst dar.

```
class load_apfel :
  virtual public apfel,
  virtual public QoS_load,
  virtual public QoS::MAQSServerFramework
{
 ...
};

class load_apfel_stub :
  virtual public load_apfel,
  virtual public apfel_stub,
  virtual public QoS_load_stub,
  virtual public QoS::MAQSServerFramework_stub
{
...
};

class load_apfel_skel :
  virtual public MAQS::QOAMethodDispatcher,
  virtual public load_apfel
```

```
{
...
};
```

Für den Dienst-Implementierer ist nur die Implementierung des Dienstes von Bedeutung. Dieser kann, wie in der Deklaration der `load_apfel_impl`-Klasse zu sehen ist, die Implementierung des Dienstes von der `apfel_impl`-Klasse erben. Somit bleibt für den Dienstimplementierer nur die Implementierung der Methoden, die nicht von der Dienstgüte-Implementierung bereitgestellt werden. Dies ist in diesem Fall die Lastinformation.

```
class load_apfel_impl :
  virtual public apfel_impl,
  virtual public MAQSServerFramework_impl,
  virtual public load_apfel_skel
{
...
};

CORBA::Float
load_apfel_impl::get_load()
{
  return load_inf;
}
```

Die Implementierung der Lastinformation geschieht in der Dienstklasse `load_apfel_impl`. Die Implementierung der Operation ist aus der Dienstgüte-Schnittstelle an den Dienst delegiert worden. Ist die Lastinformation durch die Dienstgüte-Implementierung bereitgestellt, muß der Dienst keine Anpassung an die Dienstgüte vornehmen.

Auf der Klientenseite ist die Benutzung des Dienstes durch CORBA einfach möglich. Nach der Initialisierung des ORBs und Instantiierung eines Stub-Objektes kann wie mit einem lokalen Objekt kommuniziert werden. Im folgenden ist das Code-Fragment zum Anfordern einer Zeile zu sehen.

```
apfel_line_var ret;
server -> calc(startr, wi/height*y+starti, wr, width, ret);
```

Die Instantiierung eines Stub-Objektes erfordert einen kleinen Mehraufwand gegenüber der reinen CORBA-Interaktion. In der Hauptsache resultiert dies aus dem manuellen Setzen des Mediators und des Eintragens von Diensten in den Mediator.

```
1:   CORBA::Object_var obj = o->string_to_object( ref );
2:   server = load_apfel::_narrow(obj);
3:   load_Mediator *med = new load_Mediator;
4:   med->orb = o;
5:   server->_set_mediator(med);
6:   server->set_policy(0); // per call is default
7:   ifstream refstream("ref");
8:   while (refstream >> ref)
9:   {
10:    server->enter_server(CORBA::string_dup(ref));
11:  }
```

In Zeilen 1 und 2 ist die Instantiierung des Stub-Objektes für den `load_apfel`-Dienst dargestellt. In Zeile 3-5 wird der Mediator instantiiert und im Stub-Objekt des Dienstes eingetragen. In den folgenden Zeilen wird der Mediator konfiguriert. Zunächst wird die Binderichtlinie gesetzt (Zeile 6) und dann in Zeilen 7-11 die Objektreferenzen der Dienste eingetragen. Nun ist der Mediator für die Dienstgüte-Erbringung konfiguriert und die Aufrufe an den Dienst werden unter allen eingetragenen Diensten in Abhängigkeit der Last verteilt.

B.3 Die Ticker-Anwendung

In diesem Unterabschnitt wird die Integration der ursprünglich für die Mandebrotmenge entwickelten Dienstgüte-Charakteristik "Load" in eine andere Anwendung beschrieben. Das Anwendungsszenario ist in Abbildung B.3 dargestellt. Ein Ticker-Dienst stellt in periodischen Abständen Daten bereit, die an Anwender geliefert werden. Auf diesen Daten werden von den Anwendern Analysen angewendet, die von bestimmten Diensten angeboten werden. Solche Anwendungsszenarien finden sich beispielsweise in Banken, bei denen Aktienkurse bewertet werden und aufgrund von Analysen Käufe bzw. Verkäufe getätigt werden.

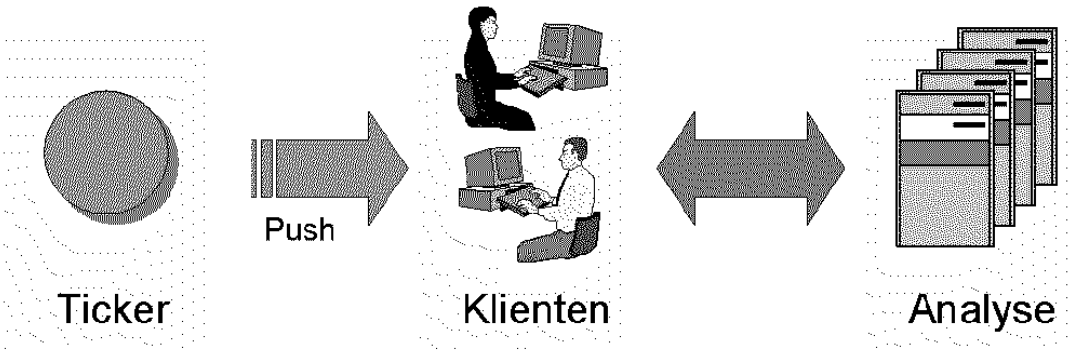


Abbildung B.3: Ticker: Anwendungsszenario

Bei aufwendigen Analysefunktionalitäten, aber gleichzeitig hohen Anforderungen an die Antwortzeit, kann eine gleichmäßige Auslastung der Dienste zu einer Verbesserung des mittleren Antwortverhaltens führen. Dies wird durch die Dienstgüte-Charakteristik "Load" bewerkstelligt.

Die in Abbildung B.3 abgebildete Anwendung erhält Börsendaten durch einen Ticker-Dienst. Der Anwender kann aus den Angeboten des Ticker-Dienstes (rechte obere Auswahlbox) bestimmte auswählen und in die Anzeigebox (linke obere Box) übernehmen. In der Anzeigebox sind aktuelle Werte und der Verlauf dargestellt. Für jeweils einen Wert läßt sich eine Analyse in dem unteren Bereich des Fensters darstellen. Dazu werden die Daten des Ticker-Dienstes an den Analyse-Dienst übergeben. Diese Analysen sind – in Anlehnung an die Realität – aufwendig, wohingegen die Entscheidungen der Anwender möglichst kurzfristig zu treffen sind. Daher ist die Interaktion mit dem Analyse-Dienst mit der Dienstgüte-Charakteristik "Load" zu versehen.

```
interface calc
{
  typedef sequence<double> doublelist;
  void simulate(in doublelist points, out doublelist lower,
        out doublelist upper, in long avglen);
};
```

Der Analyse-Dienst bietet seinen Service über die `calc`-Schnittstelle an. Die Daten werden als Sequenz von Fließkommazahlen ausgetauscht. Aufgrund der Historie bestimmt der Analyse-Dienst den Durchschnitt und die geglättete obere und untere Abschätzungen der Daten.

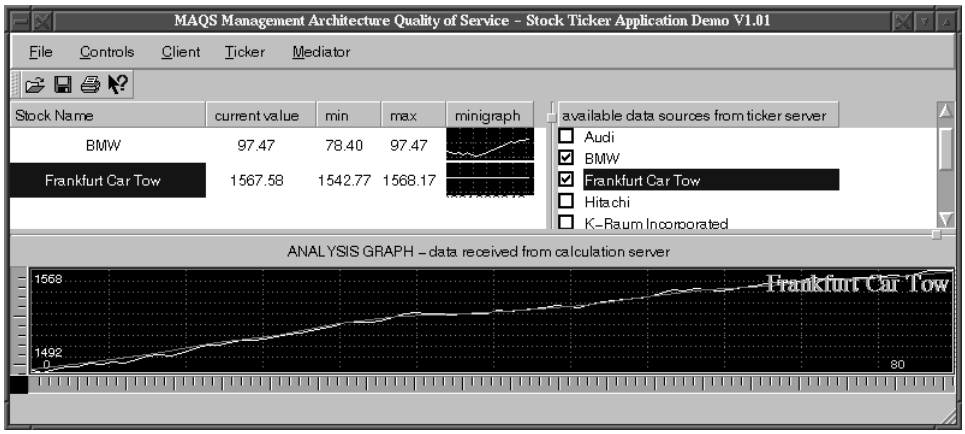


Abbildung B.4: Lastbalancierung von Analyse-Diensten

```
#include "calc.idl"
#include "qos.loadbalancing.idl"

interface calc_loadbalancing : calc withQoS load
{

};
```

Die Dienstgüte-Befähigung wird wie bei der Mandelbrot-Anwendung auch durch Ableitung von der Dienstschnittstelle und Zuweisung der Dienstgüte-Charakteristik gebildet. Die Anwendung bleibt bis auf die Initialisierung des Mediators auf der Klientenseite von Änderungen verschont. Auf der Dienstseite ist die Lastinformation bereitzustellen.

```
void CCalc_Server_App::calculate_load()
{
        m_current_load = m_calls / m_calls_per_second;
        if ( m_current_load > 1.0 )
        {
                m_calls_per_second++;
                m_current_load = 1.0;
        }
        m_calls = 0;
};
```

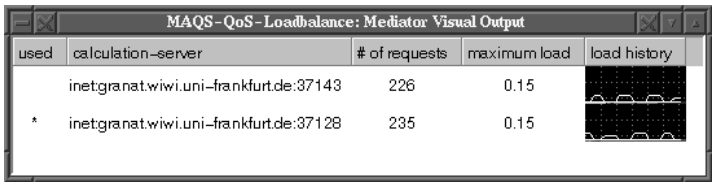


Abbildung B.5: Mediator-Visualisierung der Lastbalancierung

```
float CCalc_Server_App::get_load()
{
        return m_current_load;
};
```

Die Lastinformation wird durch die Anzahl Anfragen während einer bestimmten Zeiteinheit gebildet.

Abbildung B.5 visualisiert die Verteilung der Aufrufe auf zwei Analyse-Dienste. Im Fenster sind zwei Dienste und deren Last abgebildet. In den Lastgraphen ist gut zu erkennen, daß der Mediator zwischen beiden Diensten wechselt und die Last ausgeglichen hält.

Literaturverzeichnis

[1] ANSAware Release 4.1, Manual Set (Februar 1993).

[2] M. Banfield, C. Edwards, N. Charton und D. Hutchinson. Providing Scaleable QoS-based Connectivity Services. In „Proceedings of TINA'99", Hawaii, USA (April 1999).

[3] C. Becker. „Modellierung anwendungsspezifischer Kommunikationsbibliotheken mit Design Patterns". Diplomarbeit Universität Kaiserslautern, Fachbereich Informatik (Februar 1996).

[4] C. Becker. MICO: Stream Service: http://www.mico.org (1998).

[5] C. Becker und D. Bölsche. New Means for ERP Systems by eContracting. In „Proceedings of AMCIS 2000, Mini Track: Advanced IT Applications in ERP Systems", Long Beach, USA (August 2000).

[6] C. Becker und K. Geihs. MAQS - Management for Adaptive QoS-enabled Services. In „Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services", San Francisco, USA (Dezember 1997).

[7] C. Becker und K. Geihs. QoS as a Competitive Advantage for Distributed Object Systems. In „Proceedings of EDOC'98", La Jolla, USA (November 1998).

[8] C. Becker und K. Geihs. Quality of Service - Aspects of Distributed Programs. In „Second Workshop on Aspect-Oriented Programming", Kyoto, Japan (April 1998).

[9] C. Becker und K. Geihs. Generic QoS Specifications for CORBA. In „Proceedings of KIVS'99", Darmstadt, Germany (März 1999).

[10] C. Becker und K. Geihs. Generic QoS-Support for CORBA. In „Proceedings of ISCC'00", Antibes, France (Juli 2000).

[11] C. Becker, K. Geihs und J. Gramberg. Representing Quality of Service Preferences by Hierarchies of Contracts. In „Proceedings of Elektronische Dienstleistungswirtschaft und Financial Engineering", Augsburg, Germany (September 1999).

[12] C. Becker und J. Zinky. Quality of Service in Distributed Object Systems. *ECOOP 2000 Workshop Reader, Eds. Malenfant, Moisan and Moreira, Springer* (2000).

[13] L. M. J. Bergmans und M. Aksit. Aspects and Crosscutting in Layered Middleware Systems. In „Reflective Middleware Workshop (RM 2000)", New York, USA (April 2000).

[14] K. Birman. A Review of Experiences with reliable Multicast. *Software – Practice and Experience* **Vol. 29(9), 741-774** (1999).

[15] K. P. Birman und T. J. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. Computer Systems* **Vol. 5(1), 47-76** (1987).

[16] G. S. Blair, G. Coulson, P. Robin und M. Papathomas. An Architecture for Next Generation Middleware. In „Proceedings of Middleware'98", The Lake District, England (September 1998).

[17] G. Booch. „Object-Oriented Analysis And Design with Applications, 2nd Ed." The Benjamin/Cummings Publishing Company (1994).

[18] A.T. Campbell, G. Coulson und D. Hutchinson. A Quality of Service Architecture. *ACM SIGCOMM Computer Communication Review* **Vol. 24(2), 6-27** (1994).

[19] L. Cardelli. A Semantics of Multiple Inheritance. *Information and Computation* **Vol. 76, 138-164** (1988).

[20] P.E. Chung, Y. Huang, S. Yajnik, D. Liang, J.C. Shih, C.-Y. Wang und Y.-M. Wang. DCOM and CORBA Side by Side, Step by Step, and Layer by Layer. *C++ Report* **vol. 10, no. 1, January 1998, pp. 18-29** (1998).

[21] G. COULOURIS, J. DOLLIMORE UND T. KINDBERG. „Distributed Systems". Addison-Wesley Publishing Company (1994).

[22] K. CZARNECKI. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Dissertation, Fachbereich Informatik, Technische Universität, Ilmenau (1999).

[23] J. DANIEL, B. TRAVERSON UND S. VIGNES. Integration of Quality of Service in Distributed Object Systems. In „Proceedings of DAIS'99", Helsinki, Finnland (Juni 1999).

[24] J. DANIEL, B. TRAVERSON UND S. VIGNES. A generic QoS management framework for distributed environments. In „ECOOP Workshop on Quality of Service in Distributed Object Systems http://www.vsb.cs.uni-frankfurt.de/misc/QoSDOS/", Cannes, France (Juni 2000).

[25] M. DEBUSMANN, R. KROEGER UND CH. WEYER. Towards an automated management of distributed applications. In „Proceedings of DAIS'97", Cottbus, Germany (September 1999).

[26] N. K. DIAKOV, H. J. BATTERAM, H. ZANDBELT UND M. J. VAN SINDEREN. Monitoring of Distributed Component Interactions. In „Proceedings of Workshop on Reflective Middleware", New York, USA (April 1999).

[27] P. DINI UND A. HAFID. Towards Automatic Trading of QoS Parameters in Multimedia Distributed Applications. In „Proceedings of Open Distributed Processing and Distributed Platforms", Toronto, Canada (Mai 1997).

[28] W. EMMERICH. „Engineering Distributed Objects". Wiley (2000).

[29] P. FELBER, R. GUERRAOUI UND A. SCHIPER. The Implementation of a CORBA Object Group Service. *Theory and Practice of Object Systems* **Vol. 4(2), 93-105** (1998).

[30] A. FLADENMULLER, A. SENEVIRATNE UND E. HORLAIT. A Hybrid QoS Management Scheme for Distributed Multimedia Applications.

In „Proceedings of the Second Workshop on PROMS“, Salzburg, Austria (Oktober 1995).

[31] D. FLANAGAN. „Java in a Nutshell, 3rd Ed.“ O'Reilly (1999).

[32] G. FRANKHAUSER, B. STILLER, C. VÖGTLI UND B. PLATTNER. Reservation-based Charging in an Integrated Services Network. In „Proceedings of 4th INFORMS Telecommunications Conference“, Boca Raton, USA (März 1998).

[33] S. FRØLUND UND J. KOISTINEN. Quality of Service Aware Distributed Object Systems. Bericht HPL-98-142, HP-Labs, Palo Alto (1998).

[34] S. FRØLUND UND J. KOISTINEN. Quality of Service Specification in Distributed Object System Design. In „Proceedings of the COOTS 98“, Santa Fee, USA (März 1998).

[35] E. GAMMA, R. HELM, R. JOHNSON UND J. VLISSIDES. „Design Patterns, Elements of Reusable Object-Oriented Software“. Addison-Wesley Publishing Company (1995).

[36] K. GEIHS UND C. GEBAUER. Load Monitor – Ein CORBA-basiertes Werkzeug zur Lastbestimmung in heterogenen verteilten Systemen. In „Proceedings of ITG/GI-Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen“, Freiberg, Sachsen (September 1997).

[37] A. GOLDBERG. „Smalltalk–80: The Language and its Implementation“. Addison–Wesley Publishing Company (1985).

[38] J. GRAMBERG. Dienstgüte-Verhandlung in CORBA. Diplomarbeit, Fachbereich Informatik, Goethe Universität, Frankfurt (Februar 1999).

[39] A. GUPTA, D. O. STAHL UND A. B. WHINSTON. The Economics of Network Management. *Communications of the ACM* **Vol. 42(9), 57-63** (1999).

[40] G. HARTER UND K. GEIHS. An Accounting Service for Heterogeneous Distributed Environments. In „Proceedings of Distributed Computing Systems“, San Jose, USA (Juni 1988).

[41] F. HAUCK, U. BECKER, M. GEIER, E. MEIER, U. RASTOFER UND M. STECKERMEIER. AspectIX An Aspect-Oriented and CORBA-Compliant ORB Architecture. Bericht IMMD IV TR-IV-08-08, Univ. Erlangen Nürnberg (1998).

[42] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ITU-T X.901. ODP Reference Model Part 1. Bericht SC21 N8926rev, ISO/IEC JTC1/SC21/N (1995).

[43] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ODP Trading Function, ITU/ISO Committee Draft Standard ISO/IEC DIS13235 Rec. X.9tr (Mai 1995).

[44] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Quality of Service – Basic Framework. Bericht N9309, ISO (1995).

[45] IONA LTD. ORBIX 2000 White Paper, http://www.iona.com/products/iPortal/orbix2000.wp.pdf (2000).

[46] L. JAKOBSMEIER. Verteilte Multimedia-Systeme mit CORBA. Diplomarbeit, Fachbereich Informatik, Goethe Universität, Frankfurt (Mai 2000).

[47] M. KARSTEN, J. SCHMITT, L. WOLF UND R. STEINMETZ. Provider-Oriented Linear Price Calculation for Integrated Services. In „Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK" (Juni 1999).

[48] S. KÄTKER. Dienstorientierte Fehleranalyse in verteilten Systemen. Dissertation, Fachbereich Informatik, Goethe Universität, Frankfurt (Februar 1999).

[49] E. KELLY, N. MERCOUROFF UND P. GRAUBMANN. TINA–C DPE Architecture and Tools. In „TINA '95 Proceedings", Melbourne, Australia (Februar 1995).

[50] G. KICZALES UND ANDERE. Aspect-Oriented Programming. Bericht SPL97-008P9710042, Xerox Palo Alto Research Center (1997).

[51] H. KNOCHE UND H. DE MEER. Quantitative QoS-Mapping: A Unifying Approach. In „Proceedings of IWQoS'97", New York, USA (Mai 1997).

[52] J. KOISTINEN. Dimensions for Reliability Contracts in Distributed Object Systems. Bericht HPL-97-119, HP-Labs, Palo Alto (1997).

[53] J. KOISTINEN UND A. SEETHARAMAN. Worth-Based Multi-Category Quality-of-Service Negotiation in Distributed Object Infrastructures. In „Proceedings of EDOC'98", La Jolla, USA (November 1998).

[54] F. KUHNS, D. C. SCHMIDT UND D. L. LEVINE. The Performance of a Real-time I/O Subsystem for QoS-enabled ORB Middleware. In „Proceedings of the International Symposium on Distributed Objects and Applications (DOA '99)", Edinburgh, Scotland (September 2000).

[55] G. LI UND D. OTWAY. An Open Architecture for Real-Time Processing. Bericht APM.1270.02, APM Ltd, Cambridge, UK (1994).

[56] C. LINHOFF-POPIEN UND D. THISSEN. Integrating QoS Restrictions into the Process of Service Selection. In „Proceedings of IWQoS'97", New York, USA (Mai 1997).

[57] B. LISKOV UND J. WING. A New Definition of the Subtype Relation. In „Proceedings of ECOOP'93", Kaiserslautern, Germany (1993).

[58] C. VIDEIRA LOPES UND G. KICZALES. D: A Language Framework for Distributed Programming. Bericht SPL97-010, P9710047, Xerox Palo Alto Research Center (1997).

[59] J. P. LOYALL, R. E. SCHANTZ, J. A. ZINKY UND D. E. BAKKEN. Specifying and Measuring Quality of Service in Distributed Object Systems. In „Proceedings of ISORC'98", Kyoto, Japan (April 1998).

[60] J.P. LOYALL, D. E. BAKKEN, R. E. SCHANTZ, J. A. ZINKY, D.A. KARR, R. VANEGAS UND K.R. ANDERSON. QoS Aspect Languages and their Runtime Integration. *Springer LNCS 1511* (1998).

[61] S. MAFFEIS. Adding Group Communication and Fault–Tolerance to CORBA. In „Proceedings of the USENIX Conference on Object–Oriented Technologies", Monterey, USA (Juni 1995).

[62] S. MAFFEIS. The Object Group Design Pattern. In „Proceedings of COOTS'96", Toronto, Canada (Juni 1996).

[63] B. METZLER, R. WITTMANN UND M. ZITTERBART. Towards Scalable Quality-based Heterogeneous Multicast Services. In „Proceedings of KIVS'99", Darmstadt, Germany (März 1999).

[64] B. MEYER. „Object-oriented Software Construction". Prentice-Hall (1988).

[65] L.E. MOSER, P.M. MELLIAR-SMITH UND P. NARASIMHAN. Consistent Object Replication in the Eternal System. *Theory and Practice of Object Systems* **Vol. 4(2), 81-92** (1998).

[66] M. MÜLLER. Entwurf und Implementierung einer Dienstgütespezifikation für CORBA. Diplomarbeit, Fachbereich Informatik, Goethe Universität, Frankfurt (Februar 1999).

[67] J. NEHMER UND P. STURM. „Systemsoftware". dpunkt (1998).

[68] OBJECT MANAGEMENT GROUP. CORBAservices: Common Object Services Specification. Bericht Revised Edition March 1995, Update July 1997, OMG, Framingham, MA (März 1995).

[69] OBJECT MANAGEMENT GROUP. The Common Object Request Broker: Architecure and Specification, Revision 2.0. Bericht, OMG, Framingham, MA (Juli 1995).

[70] OBJECT MANAGEMENT GROUP. A Discussion of the Object Management Architecture. Bericht 97-06-23, OMG, Framingham, MA (Juni 1997).

[71] OBJECT MANAGEMENT GROUP. Quality of Service (QoS) OMG Green Paper. Bericht 97-06-04, OMG, Framingham, MA (Juni 1997).

[72] OBJECT MANAGEMENT GROUP. CORBA Messaging. Bericht orbos/98-05-06, OMG, Framingham, MA (1998).

[73] OBJECT MANAGEMENT GROUP. CORBAtelecoms: Telecommunications Domain Specifications. Bericht formal/98-07-12, OMG, Framingham, MA (1998).

[74] OBJECT MANAGEMENT GROUP. Realtime CORBA, joint submission. Bericht orbos/98-01-08, OMG, Framingham, MA (1998).

[75] OBJECT MANAGEMENT GROUP. The Common Object Request Broker: Architecture and Specification (Revision 2.3). Bericht 98-10-03, OMG, Framingham, MA (Juli 1998).

[76] OBJECT ORIENTED CONCEPTS. Open Communications Interface; in ORBacus 4.0.1 Dokumentation, http://www.ooc.com (2000).

[77] OBJECT ORIENTED CONCEPTS. ORBacus 4.0.1 Dokumentation, http://www.ooc.com (2000).

[78] J. POSTEL. Transmission Control Protocol, RFC 793 (September 1981).

[79] A. PUDER. „Typsysteme für die Dienstvermittlung in offenen verteilten Systemen". Dissertation, Goethe Universität Frankfurt (1997).

[80] A. PUDER, S. MARKWITZ, F. GUDERMANN UND K. GEIHS. AI–based Trading in Open Distributed Environments. In „Proceedings 3rd International IFIP Conference on Open Distributed Processing (ICODP'95)", Brisbane, Australia (Februar 1995).

[81] E. PULVERMÜLLER, H KLAEREN UND A. SPECK. Aspects in Distributed Environments. In „Proceedings of GCSE'99", Erfurt, Germany (September 1999).

[82] D. REED UND R. FAIRBAIRNS. Nemesis Kernel Overview. Bericht http://www.cl.cam.ac.uk/Research/SRG/netos/pegasus /publications/overview/brief-overview.html, University of Cambridge, UK.

[83] K. RÖMER. MICO –MICO is CORBA, Eine erweiterbare CORBA-Implementierung für Forschung und Ausbildung. Diplomarbeit, Fachbereich Informatik, Goethe Universität, Frankfurt (Februar 1998).

[84] D. C. SCHMIDT. ACE: An Object–Oriented Framework for Developing Distributed Applications. In „Proceedings of the USENIX C++– Conference", Cambridge, USA (April 1994).

[85] D. C. SCHMIDT, D. L. LEVINE UND S. MUNGEE. The Design of the TAO Real–Time Object Request Broker. *Computer Communications Journal* **vol. 21(4)** (1998).

[86] D.C. SCHMIDT UND C. CLEELAND. Applying Patterns to Develop Extensible ORB Middleware. *IEEE Communications Special Issue on Patterns* **Vol. 16 (4)** (1999).

[87] F. B. SCHNEIDER. Replication Management using the State-Machine Approach. *Distributed Systems, Ed. S. Mullender* **Addison Wesley, pp. 169-197** (1993).

[88] F. SIQUEIRA UND V. CAHILL. Quartz: Supporting QoS-Constrained Services in Heterogeneous Environments. In „Proceedings of the 7th IEEE Workshop on Future Trends in Distributed Computing Systems", Cape Town, South Africa (Dezember 1999).

[89] B. STILLER. „Quality-of-Service – Dienstgüte in Hochleistungsnetzen". Thompson Publishing (1995).

[90] B. STROUSTRUP. „The C++ Programming Language, 2nd Ed." Addison-Wesley Publishing Company (1993).

[91] A. SZTAJNBERG UND O. LOQUES. Bringing QoS Specifications to the Architectural Level. In „ECOOP Workshop on Quality of Service in Distributed Object Systems http://www.vsb.cs.uni-frankfurt.de/misc/QoSDOS/", Cannes, France (Juni 2000).

[92] A. S. TANENBAUM. „Computer Networks". Prentice/Hall International (1989).

[93] A. S. TANENBAUM. „Modern Operating Systems". Prentice/Hall International (1992).

[94] TINA Consortium. „'94 Report on Fault Tolerance and Resource Configuration Management, TR_MRK.006_1.0_94 " (Juli 1994).

[95] TINA Consortium. „Quality of Service Framework DRAFT, TR_MRK.001_1.0_94" (Juli 1994).

[96] INTERNATIONAL TELECOMMUNICATION UNION. „E.800 Quality of Service and Dependability Vocabulary" (1993).

[97] R. VAN RENESSE, K.P. BIRMAN UND S. MAFFEIS. Horus: A Flexible Group Communication System. *Communications of the ACM* **Vol. 39 (4)** (1996).

[98] R. VANEGAS, J. A. ZINKY, J. P. LOYALL, D. KARR, R. E. SCHANTZ UND D. E. BAKKEN. QuO's Runtime Support for Quality of Service in Distributed Objects. In „Proceedings of Middleware'98", The Lake District, England (September 1999).

[99] A. VAYESBURD UND K. BIRMAN. The Maestro Approach to Building Reliable Interoperable Distributed Applications with Multiple Execution Styles. *Theory and Practice of Object Systems* **Vol. 4(2), 93-105** (1998).

[100] D.G. WADDINGTON, G. COULSON UND D. HUTCHINSON. Specifying QoS for Multimedia Communications within Distributed Programming Environments. In „Proceedings of 3rd COST 237", Barcelona, Spain (November 1996).

[101] J. WALDO, G. WYANT, A. WOLLRATH UND S. KENDALL. A Note on Distributed Computing. Bericht 94-29, Sun Microsystems Laboratories, Inc. (1994).

[102] J. WROCLAWSLI. The use of RSVP with IETF Integrated Services, RFC 2210 (1997).

[103] J. A. ZINKY, D. E. BAKKEN UND R. E. SCHANTZ. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems* **Vol. 3(1), 55-73** (1997).

Lebenslauf

18.7.1968	Geboren in Hanau/Main
1974 - 1978	Philipp-Reis-Schule Gelnhausen
1978 - 1987	Grimmelshausen Gymnasium Gelnhausen
1987-1989	Studium der Mathematik, Universität Frankfurt
1990	Wehrdienst
1991 - 1993	Studium der Informatik, Universität Karlsruhe (TH)
1993 - 1996	Studium der Informatik, Universität Kaiserlautern
seit 1997	Wissenschaftlicher Mitarbeiter am Fachbereich Informatik, Professur für Verteilte Systeme/Betriebssysteme bei Prof. Dr. K. Geihs, Johann Wolfgang Goethe-Universität Frankfurt