INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Nutzung der ODP-Viewpoint Languages für das Management der verteilten Anwendung WWW

Diana Stricker

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dr. Bernhard Neumair
	Alexander Keller
Abgabedatum:	15. Mai 1998

INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Nutzung der ODP-Viewpoint Languages für das Management der verteilten Anwendung WWW

Diana Stricker

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dr. Bernhard Neumair
	Alexander Keller
Abgabedatum:	15. Mai 1998

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Mai 1998

....

(Unterschrift des Kandidaten)

Zusammenfassung

Ein wichtiges Merkmal der heutigen Netzwerke wie dem Internet oder den meisten internen Firmennetzen ist, daß sie bezüglich der Netzkomponenten, der Endsysteme und der darauf laufenden Anwendungen heterogen sind. Der Trend geht immer mehr zu Client/Server-basierten verteilten Anwendungen. Das Ziel dieser Entwicklung besteht darin, daß abhängig von den gestellten Anforderungen die beste Kombination aus Hardware- und Softwarekomponenten dazu genutzt wird, die Aufgaben bestmöglich zu lösen. Der Preis für die gewonnene Flexiblität ist eine gesteigerte Komplexität des Managements. Ein Managementmodell, das unabhängig von der angewendeten Managementarchitektur möglichst viele Arten von verteilten Anwendungen überwachen und steuern kann, ist von größter Bedeutung, um den Ansprüchen heutiger Kommunikationsstrukturen zu genügen.

Im Rahmen dieser Diplomarbeit wird ein Managementmodell für das Management der verteilten Anwendung WWW entwickelt. WWW-Dienste werden von Web-Servern bereitgestellt, von Web-Clients (sogenannten Browsern) genutzt und über Proxies vermittelt. Für das Management von verteilten Anwendungen, die WWW-Dienste realisieren, gibt es noch kaum effektive Lösungen. Verschiedene Möglichkeiten wurden untersucht, wobei diese meistens nur ein Monitoring der Komponenten ermöglichen und kaum Möglichkeiten gegeben sind, steuernd darauf einzuwirken. Außerdem sind sie auf SNMP als Managementprotokoll angewiesen, wodurch eine Unabhängigkeit von der verwendeten Managementarchitektur nicht gegeben ist. Ein eigener Lösungsweg wurde anhand konkreter Managementszenarien entwickelt und durch Beobachtung mehrerer Anwendungen für WWW-Dienste.

Um ein effektives Management von unterschiedlichen verteilten Anwendungen, die WWW-Dienste realisieren, zu gewährleisten, muß das Managementmodell unabhängig von der Managementarchitektur sein und allgemeingültige Managementinformation definieren. Dazu werden Konzepte des *Reference Model of Open Distributed Processing (RM-ODP)* angewandt, das ein Rahmenwerk für die Entwicklung von verteilten Anwendungen bietet. Fünf verschiedene Sichten (*Viewpoints*) auf den Entwicklungsprozeß einer verteilten Anwendung erlauben unter Anwendung des Referenzmodells, daß die Komplexität der Anwendung aufgeteilt und deren Entwicklung und Beschreibung dadurch erleichtert wird. Durch Analyse speziell der Konzepte des *Computational* und *Engineering Viewpoints* unter Managementaspekten werden generische Objektklassen definiert, die allgemein benötigte Managementinformation über verteilte Anwendungen definieren. Eine Bottom-Up-Analyse verschiedener Anwendungen erlaubt die Definition von anwendungsspezifischen Merkmalen. Diese werden in Form von Attributen und Methoden in Klassen zusammengefaßt und anschließend in das generische Modell eingebunden. Das so entstandene Managementmodell umfaßt generische Klassen, die für unterschiedlichste Ressourcen gelten und die in weiteren Klassen verfeinert werden, um speziellere Anwendungen, nämlich die des WWW-Managements, abzudecken.

Das Modell wird mit Hilfe von Methoden der *Object Modeling Technique (OMT)* erstellt, die einen objektorientierten Modellierungsansatz bietet, der unabhängig von der verwendeten Managementarchitektur ist. Das Modell läßt sich dadurch auf Informationsmodelle unterschiedlicher Managementarchitekturen abbilden und wird im Rahmen dieser Diplomarbeit auf Basis der *Common Object Request Broker Architecture (CORBA)* entwickelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Unter den Aspekt des Anwendungsmanagements fallen sowohl steuernde als auch überwachende Tätigkeiten über Softwareanwendungen, um jegliche Probleme während des Betriebs zu vermeiden. Das fängt bei der richtigen Installation der Komponenten einer Anwendung an, betrifft die Konfiguration und führt über das Überwachen des laufenden Betriebs zum aktiven Fehlermanagement bei auftretenden Problemen. Das Gebiet des Anwendungsmanagements hat sich aus der Notwendigkeit heraus entwickelt, daß Softwarepakete über die letzten Jahre kontinuierlich immer komplexer und umfangreicher wurden. Waren es früher noch um die 1000 Lines of Code, die eine Anwendung ausmachten, gibt es heute schon Anwendungen, wie z.B. SAP R/3, die einige Millionen Lines of Code umfassen.

In einer Firma stehen heutzutage Systemmanager vor neuen Aufgaben, für deren Lösungen erst noch Werkzeuge entwickelt werden müssen. Die bisherige Entwicklung der Software verläuft manchmal leider nach dem Prinzip, daß die Probleme, die während des Betriebs aufkommen, erst nachträglich berücksichtigt werden. Das heißt, daß Managementaspekte bei der Entwicklung der Software noch nicht berücksichtigt wurden.

Auch die Tatsache, daß die hohe Anzahl an Hard- und Softwarekomponenten einer Firma heutzutage über größere geographische Entfernungen verteilt ist, macht es für einen Administrator erheblich schwerer, Änderungen oder ein effektives Management (wie z.B. Fehler-, Leistungs- oder Sicherheitsmanagement) mit klassischen Methoden durchzuführen. Es erfordert andere Werkzeuge, um im Vergleich zu Mainframe-Architekturen Client-Server-Umgebungen und verteilte Anwendungen zu überwachen und steuern.

Ein weiteres Problem entsteht durch die Heterogenität der Hard- und Softwarekomponenten in einem System, da diese meistens von unterschiedlichen Herstellern stammen und auch (z.B. bei Rechnerarten: Mainframes vs. PCs) unterschiedliche Ausprägungen besitzen. Die Daten, die über das System zu Managementzwecken ausgetauscht werden, müssen über verschiedene Umgebungen verschickt und in unterschiedliche Formate umgewandelt werden, was die Managementaufgabe zusätzlich komplexer und umfangreicher macht.

Mehrere Gesichtspunkte müssen also im Rahmen des Anwendungsmanagements berücksichtigt werden, um eine zufriedenstellende Funktionalität der Software zu gewährleisten.

1.1 Motivation

Nachdem die weltweite Verbreitung von Information über das Internet in den letzten Jahren in unterschiedlichen Anwenderkreisen sehr stark an Bedeutung und Akzeptanz zugenommen hat, drängen immer mehr Firmen und Privatpersonen mit immer umfangreicheren und aufwendig gestalteten Web-Sites in dieses Medium.

Das World Wide Web (WWW) ist der populärste Informationsdienst, der das Internet als Transportbasis nutzt. Aufgrund der Fähigkeit des HyperText Transport Protocol (HTTP), des Anwendungsprotokolls, über das Web-Clients mit Web-Servern Daten austauschen, können beliebige Dokumente miteinander verknüpft werden. Das können Text-, Bilder- oder auch Audio- und Videodateien sein. Die Anzahl und Vielfalt der über das Web ausgetauschten und abrufbaren Dokumente nimmt kontinuierlich zu. Die Verbreitung von Anwendungen, die für das Bearbeiten und Darstellen der unterschiedlichsten Dateiformate benötigt werden, wächst stetig. So verursacht die Verwaltung aller HTML-Dateien, Skripten und Images, die über das WWW verbreitet werden, einen steigenden Aufwand.

Zusätzlich zu dieser Entwicklung vermehrt sich die Anzahl der Server und Clients, die im Internet Informationen zur Verfügung stellen bzw. auf Informationen zugreifen. Falls eine Firma ihren Mitarbeitern einen Internet-Zugang bereitstellt, müssen die Clients und Zugangsrechner dafür installiert und konfiguriert werden, die Benutzerverwaltung muß eingerichtet werden, und wichtige Sicherheitseinrichtungen sollen das interne Netz vor Eindringlinge aus dem Internet schützen. So erhöht sich innerhalb der Firma der Verwaltungsaufwand für alle Rechner- und Software-Ressourcen.

Außerdem vergrößert sich der Verwaltungsaufwand, falls diese Firma mit einer eigenen Homepage im Internet präsent ist. Dann muß die richtige Funktionsweise des Web-Servers, dessen Performance und ebenso die Aktualität der Daten, die Inhalt der Homepage sind, gesichert sein. Um die Übersicht und Kontrolle darüber zu behalten, benötigt ein Systemverwalter entsprechende Managementwerkzeuge, die ihm diese Arbeit erleichtern oder durch automatisierte Prozesse sogar abnehmen.

Für Internet-Service-Provider, die die Anbindung ihrer Kunden an das Internet und teilweise auch deren Web-Auftritte verwalten, bieten sich neben den oben aufgezählten Szenarien noch eine Menge anderer. Sie müssen z.B. jedem Kunden Zugangsdaten zu dessen Homepage bereistellen, damit dieser die Anzahl der Requests auf seine Seiten sieht und erfährt, woher auf diese Seiten zugegriffen wurde. Weiter stellen sich pro Kunde Fragen nach der Abrechnung der angebotenen Dienste, nach Sicherheitsaspekten oder nach der Auslastung von Verbindungen. Ziel der Provider sollte die Vermeidung von Engpässen und Ausfällen sein, die durch unzureichende Überwachung und Wartung aller Soft- und Hardware-Ressourcen entstehen können.

Diese Entwicklung, teilweise verbunden mit der Unkenntnis mancher Anwender (z.B. über die Funktionsweise von Browsern), ziehen einen erhöhten Bedarf an Managementwerkzeugen nach sich, um auf jedem Anwendungsgebiet im Rahmen des WWW eine gewisse Übersicht und Sicherheit gewährleisten zu können.

1.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll speziell die verteilte Anwendung World Wide Web (WWW) zu Managementzwecken bezüglich ihrer richtigen Funktionsweise analysiert und modelliert werden.

Dazu sollen die Konzepte des Referenzmodells für Open Distributed Processing (RM-ODP) über verteilte Anwendungen berücksichtigt und die objektorientierte Modellierungstechnik Object Modeling Technique (OMT) nach Rumbaugh angewendet werden. Das mit Hilfe dieser Methoden gewonnene Top-Down-Modell der Anwendung WWW soll also für die Belange des Anwendungsmanagements geeignet sein.

Der Fokus dieser Diplomarbeit richtet sich auf das Managementwissen, das über die verteilte Anwendung WWW gesammelt werden kann, und dessen Aufbereitung für integrierte Managementanwendungen.

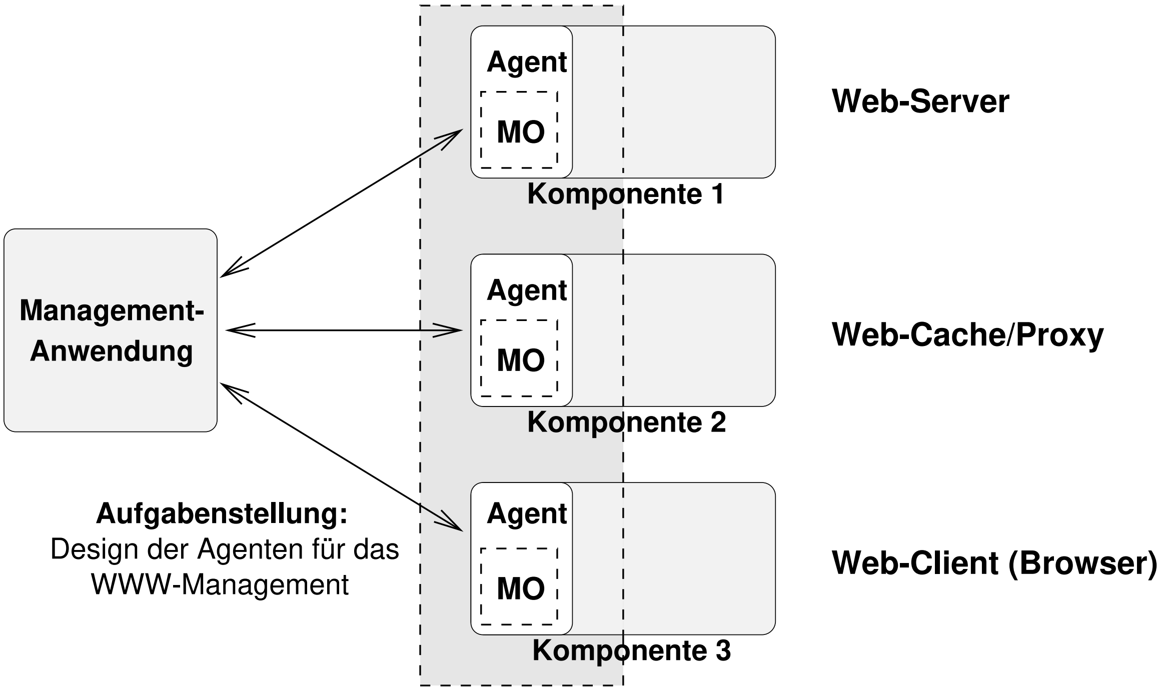

Abbildung 1.1: Abgrenzung der Aufgabenstellung

Die Komponenten einer verteilten Anwendung (siehe rechts in Abb. 1.1) liefern über entsprechende Schnittstellen Managementinformationen, die in einer herstellerunabhängigen Weise zu interpretieren sind. So können sie von beliebigen Managementanwendungen abgefragt und bearbeitet werden.

Zu diesem Zweck werden den Komponenten sogenannte Management-Agenten vorgeschaltet, die die Information in Form von Managementobjekten liefern und damit einer Managementanwendung erlauben, steuernd auf die Ressourcen einzuwirken.

Managementobjekte sind eine Abstraktion der Ressource, die für Managementzwecke benötigt wird. Die Abstraktion, also die Modellierung der Managementinformation, soll im Rahmen dieser Diplomarbeit nach einem objektorientierten Ansatz erfolgen. Dabei werden die Schnittstellen, über die auf die Managementinformation zugegriffen wird, als Attribute, Operationen und Meldungen von Objekten realisiert.

Das gesamte Umfeld des Managements der verteilten Anwendung WWW soll nach einem Top-Down-Vorgensmodell analysiert werden. Das heißt, daß in erster Linie die Anforderungen an verteilte Anwendungen allgemein betrachtet werden, die von einer Managementanwendung zu Managementzwecken gestellt werden. Im nächsten Schritt wird durch eine Bottom-Up-Analyse untersucht, welche Merkmale und Eigenschaften speziell Anwendungen charakterisieren, die WWW-Dienste realisieren. Sämtliche Eigenschaften von allgemeinen und speziell von WWW-Anwendungen werden in geeignete Klassen geordnet und in einem Objektmodell zusammengefaßt.

Die Managementanwendungen sollen über einen CORBA-Object Request Broker mit den CORBA-konformen Agenten Informationen austauschen. Das Design der Agenten für das Web-Management und ihre anschließende prototypische Implementierung ist die Aufgabenstellung dieser Diplomarbeit.

1.3 Aufbau der Diplomarbeit

Nach einer allgemeinen *Einführung* und der *Abgrenzung der Aufgabenstellung* im **ersten Kapitel** wird im **zweiten Kapitel** näher auf die Teilbereiche und *Bestandteile des World Wide Web* und des *Anwendungsmanagements* eingegangen sowie eine genaue Analyse der *Szenarien im Rahmen des Managements des Dienstes WWW* gegeben. Das *Vorgehensmodell* am Anfang des Kapitels gibt einen genaueren Einblick in die gesamten Aufgaben und Zusammenhänge der Diplomarbeit. Dabei werden wichtige Fragen zu den gewählten Lösungswegen gestellt, die im Rahmen der Diplomarbeit beantwortet werden.

Anschließend soll in **Kapitel drei** eine Übersicht über den *State of the Art* bisherige Lösungswege für die Problematik aufzeigen und gleichzeitig erklären, warum der neue Weg im Rahmen dieser Diplomarbeit eingeschlagen wird. Weitere Hilfsmittel zum Lösen der Aufgabenstellung der Diplomarbeit, wie das *Referenzmodell für das Open Distributed Processing*, die *Modellierungstechnik OMT* nach Rumbaugh und Grundeigenschaften der *Common Object Request Broker Architecture (CORBA)*, werden auch in diesem Kapitel kurz erläutert, verbunden mit der Beschreibung der jeweiligen Tools, die angewendet werden.

Die detaillierte Analyse der *Szenarien beim Management des Dienstes WWW* aus dem zweiten Kapitel sollen einen Überblick über Problemstellungen in diesem Bereich geben und werden im **vierten Kapitel** bezüglich der von den Web-Komponenten zur Verfügung gestellten Managementinformation ausführlich untersucht.

Kapitel fünf widmet sich der *Entwicklung eines geeigneten Objektmodells*, das das integrierte Management des Dienstes WWW erlauben soll. Eine abschließende *Zusammenfassung* und der *Ausblick* sind Inhalt von **Kapitel sechs**.

Kapitel 2

Anforderungsanalyse

2.1 Vorgehensmodell

In Anlehnung an die definierte Aufgabenstellung ergeben sich einige wichtige Fragen, die im weiteren Verlauf der Diplomarbeit beantwortet werden. Was sind die Merkmale von verteilten Anwendungen, wenn sie aus der Management-Perspektive betrachtet werden? Warum wird eine objektorientierte Modellierung der Managementressourcen bevorzugt? Warum soll dabei nach einem Top-Down-Verfahren vorgegangen werden? Warum eignet sich CORBA als Architektur für integriertes Management?

Als erstes ist zu beachten, daß die Komplexität heutiger IT-Infrastrukturen ein integriertes Management aller Netzwerkkomponenten, Endsysteme und Anwendungen verlangt. Innerhalb einer heterogenen, verteilten Umgebung soll das Überwachen und Verwalten beliebiger Ressourcen (ob Hardware oder Software) unabhängig von der verwendeten Managementarchitektur auf *einer* Managementplattform möglich sein. Dadurch kann die Vielfalt und Komplexität der zu steuernden Hardware- und Software-Ressourcen und -Strukturen eines Netzwerks vereinfacht und gleichzeitig ein einheitliches Management unterschiedlicher Ressourcen garantiert werden.

Die Anforderungen des integrierten Managements können anhand von konkreten Management-Szenarien sehr gut untersucht werden. Um das herstellerunabhängige Modellieren unterschiedlicher Ressourcen zu gewährleisten, ist ein **Top-Down-Vorgehen** beim Definieren von Managementinformation sehr hilfreich. Ein Informationsmodell soll möglichst viele Ressourcen eines verteilten, heterogenen Systems beschreiben und gleichzeitig die Belange des Managements erfüllen. Das gleiche gilt für Ressourcen bzw. Komponenten von verteilten Anwendungen. Dafür ist es notwendig, daß durch Modellierung geeignete, generische Managementobjektklassen definiert werden, die von den Ressourcen abstrahieren und einer Managementanwendung entsprechende Managementinformation zur Verfügung stellen. Dadurch wird der Schwerpunkt auf die Anforderungen des Managements an das System und die verteilte Anwendung gelegt und nicht auf system-, hersteller- oder anwendungsspezifische Merkmale.

Im Rahmen dieser Diplomarbeit geschieht die Definition von generischen Objektklassen, die ein integriertes Management speziell von verteilten Anwendungen in offenen, heterogenen, verteilten Systemen erlauben, in Anlehnung an das *Referenzmodell für Open Distributed Pro-*

cessing (RM-ODP). Dabei werden mit Hilfe der *ODP Viewpoint Languages* (siehe Abbildung 2.1) die Komponenten einer verteilten Anwendung festgelegt, die für das Management relevant sind (z.B. *cluster*, *capsule*, *channel* oder *node*). Hierbei werden verstärkt die Konzepte der *Computational* und der *Engineering Language* betrachtet, da sie für die Zwecke des integrierten Managements die wichtigsten Merkmale abdecken. Das ODP-Referenzmodell gibt allgemein einen standardisierten Rahmen für die Entwicklung von verteilten Anwendungen vor und beschreibt somit relevante, allgemeingültige Aspekte von verteilten Anwendungen. Konzepte der Computational Language erlauben die funktionale Zerlegung einer verteilten Anwendung in eine Menge von Objekten, die über festgelegte Schnittstellen miteinander kommunizieren und Dienste anbieten bzw. nutzen. Das Management der Anwendung erfolgt mit Hilfe von Methoden, die auf Objekte ausgeführt werden. Im Gegensatz dazu legen Konzepte der Engineering Language die Infrastruktur der Verteilung fest, die für die Realisierung der Anwendung wichtig ist.

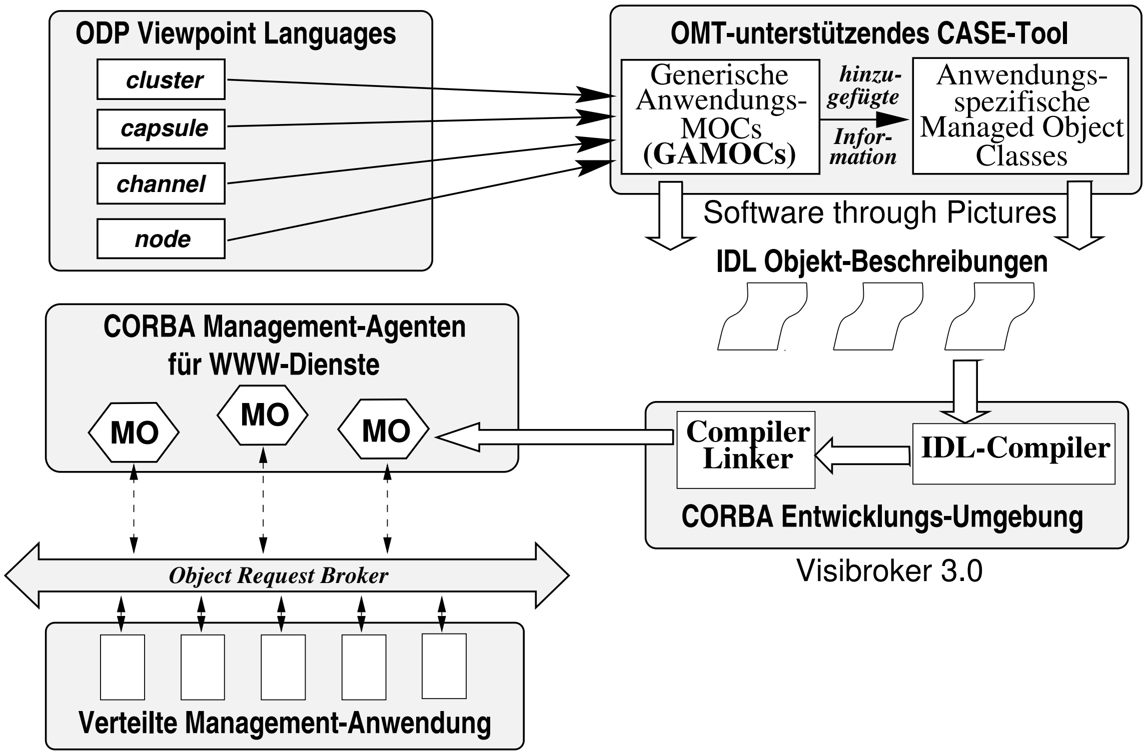

Abbildung 2.1: Vorgehensmodell (nach [AK97])

Nach Analyse der **Anforderungen an das Anwendungsmanagement** und der **Konzepte des RM-ODP** bezüglich verteilter Anwendungen können **generische Management-Objektklassen** (*Generic Application Managed Object Classes - GAMOCs*) definiert werden, die managementrelevant sind. Ihnen werden anschließend durch ein **Bottom-Up-Verfahren** anwendungsspezifische Merkmale hinzugefügt. Dabei wird berücksichtigt, welche Informationen eine Anwendung für Managementzwecke zur Verfügung stellen kann. Die generischen Basisklassen werden verfeinert, wodurch man **anwendungsspezifische Management-Objektklassen** gewinnt. Dabei ist es sehr wichtig, sich zu fragen, welche anwendungsspezifischen Merkmale allgemein für Ressourcen dieser Klasse gelten und welche nicht. Nur allgemeingültige Merkmale werden dann in die Klasse aufgenommen, um eine Abbildung der Klasse auf möglichst viele Ressourcen zu gewährleisten. Wird zuviel Information in einer Klasse zusammengefaßt, so besteht die Gefahr, daß das entworfene Modell nicht mehr generisch genug ist. Andererseits kann die Definition von zuwenig Information in einer Klasse das Modell für die Belange des integrierten Managements als ungeeignet erscheinen lassen.

Die gewonnenen generischen Objektklassen und deren Beziehungen zueinander sollen mit Hilfe der *Object Modeling Technique (OMT)* modelliert werden, da diese schon in zahlreichen Projekten des Software-Engineering erfolgreich angewandt wurde. Das gesamte Management-Objektmodell kann mit dem OMT-unterstützenden CASE-Tool *Software through Pictures (StP)* (siehe Abschnitt 3.3.1) nach OMT-Regeln erstellt werden. Diese Notation ermöglicht eine architekturunabhängige Darstellungsweise von Managementinformation, da das StP-Tool eine sowohl für Internet-Management-Architekturen als auch für OSI- und CORBA-Architekturen verständliche Spezifizierung der Aufrufschnittstellen von Objekten erlaubt. Für CORBA-Umgebungen können damit **IDL-Schnittstellenbeschreibungen** erzeugt werden.

In einer CORBA-Entwicklungsumgebung werden die IDL-Objektbeschreibungen von einem IDL-Compiler übersetzt. Dieser generiert durch standardisierte *language mappings* Code-Gerüste der Managementagenten in einer gewünschten Programmiersprache. Diese Gerüste werden mit Code gefüllt, von einem Compiler übersetzt und gelinkt und sind somit CORBA-konforme Managementagenten.

In den nächsten Kapiteln werden die einzelnen Methoden zum Lösen der Aufgabenstellung näher erläutert.

2.2 Begriffsbildung

Die verteilte Anwendung *World Wide Web* besteht aus **Server**- und **Client**-Komponenten, die mit Hilfe des **HyperText Transfer Protocol (HTTP)** kommunizieren. Server haben die Rolle der Dienstanbieter, während Clients als Dienstnutzer agieren. Dabei kann ein Client ein Dokument auch über **Proxy-Server** von einem Web-Server beantragen, so daß Proxies in diesem Zusammenhang auch als Teil des Webs betrachtet werden.

2.2.1 Web-Server und das HTTP

Im World Wide Web stellen *Web-Server* Information in Form von Dateien zur Verfügung, auf die mit Hilfe von Clients zugegriffen werden kann. Ein Web-Server ist ein Software-Programm, das auf Dateien zugreifen und sie anschließend an Clients verschicken kann. Nach dem Client-Server-Prinzip stellt ein Web-Client eine Anfrage an einen Web-Server, der daraufhin die Anfrage bearbeitet und ein Ergebnis zurückliefert. Web-Server warten auf Anfragen und starten dann erst Prozesse und weitere Programme, um die Anfragen zu bearbeiten. Es können also nur auf Initiative der Clients Daten zwischen Servern und Clients ausgetauscht werden.

Web-Server können auch spezielle Programme ausführen, die es erlauben, den Dokumenten oder dem Web-Server selbst erweiterte Funktionalitäten hinzuzufügen. Sogenannte *Hilfsprogramme*, die das Erweitern der Server-Funktionalität erlauben, stellen sich aus vordefinierten Funktionen zusammen, mit denen die Kernfunktionen des Servers angesprochen und somit erweitert oder verändert werden können. Das Schreiben von Hilfsprogrammen erfordert umfassende Programmierkenntnisse, da ein fehlerhaftes Programm den Server zum Absturz bringen kann.

Mit Hilfe von *CGI-Programmen* können Web-Server Clients dynamisch erzeugte Web-Seiten anbieten. Meistens ist ein CGI-Programm mit einem Formular verknüpft. Nach Eingabe der Daten in das Formular und anschließendem Abschicken des Formulars werden die eingegebenen Daten zusammen mit dem Namen des CGI-Programms, das sie bearbeiten soll, vom Web-Client an den Server geschickt. Der Web-Server ruft dann das CGI-Programm auf und übergibt ihm die Daten.

Ein Server nutzt das Common Gateway Interface, um Daten an das CGI-Programm zu übergeben. Dabei wird die Art der Datenübertragung durch die zwei HTTP-Methoden `GET` oder `POST` bestimmt, mit denen ein CGI-Programm aufgerufen werden kann. Wird ein CGI-Programm mit Hilfe der `GET`-Methode aufgerufen, so übergibt der Client die Daten im URI-Pfad an den Server. Ein **URI** (*Uniform Resource Identifier*) ist eine Zeichenreihe, die eine bestimmte Datei identifiziert. Der Ort, an dem die Datei auf einem Web-Server gespeichert ist, kann entweder *absolut* - also mit Servername, Pfad und Name der Datei - oder *relativ* zu einem Basis-URI - also nur Pfad und Name der Datei - angegeben werden. Dagegen enthält ein **URL** (*Uniform Resource Locator*) im Gegensatz zu einem URI zusätzlich noch das Zugriffsprotokoll, mit dem auf die Datei zugegriffen wird. Ein Beispiel-Aufruf für ein CGI-Programm mittels einem URI wäre:

```
GET /cgi-bin/script?ort=hier&zeit=jetzt HTTP/1.0
```

"`/cgi-bin/script`" stellt dabei das aufgerufene CGI-Programm dar, dem, getrennt durch ein "?", die Daten übergeben werden müssen. In diesem Fall sind es die *Variable*=*Wert*-Paare "ort=hier" und "zeit=jetzt". Am Ende der Anfragezeile wird die HTTP-Version angegeben, die für die Anfrage verwendet wird.

Der Server muß diesem CGI-Programm die Daten bereitstellen. Dazu verwendet er entweder Umgebungsvariablen oder die Standardeingabe. Im Falle eines `GET`-Aufrufs wird die Zeichenkette in der Umgebungsvariable "QUERY_STRING" bereitgestellt. Bei einem `POST`-Aufruf wird in der Variable "CONTENT_LENGTH" die Länge der Zeichenkette in Bytes eingetragen, wobei die eigentliche Zeichenkette dem CGI-Programm über die Standardeingabe (STDIN) zur Verfügung gestellt wird. Die `POST`-Methode eignet sich im Gegensatz zur `GET`-Methode eher für die Übertragung von größeren Datenmengen.

Das CGI-Programm verarbeitet die ihm übergebenen Daten und erzeugt eine Ausgabe, die vom Web-Server an den Client geschickt wird (siehe Abbildung 2.2).

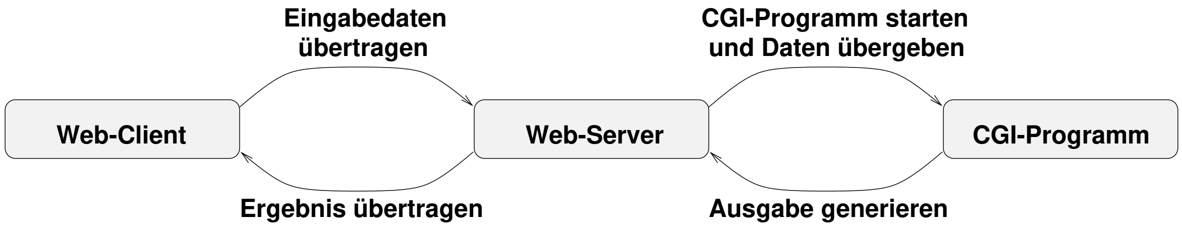

Abbildung 2.2: Ablauf beim Aufruf eines CGI-Programms

Clients und Server benutzen das *HyperText Transfer Protocol (HTTP)* zur Kommunikation. Für jede Anfrage eines Web-Clients an einen Web-Server wird eine Verbindung zwischen den beiden hergestellt, über die Daten ausgetauscht werden und die anschließend wieder beendet wird. Ein Client schickt dabei die Anfrage an den Server, woraufhin dieser die angeforderte Datei zurückschickt. Enthält das zurückgeschickte Dokument auch noch eingebundene Bilder-Dateien oder Dateien anderen Formats, so muß zur Übertragung jeder einzelnen Datei eine eigene Verbindung zwischen dem Server und dem anfragenden Client aufgebaut werden, da je HTTP-Request nur eine Datei übertragen werden kann.

Das HTTP ist ein zustandsloses Protokoll der Anwendungsebene. Das heißt, daß sich Web-Server nach Beendigung einer Verbindung keine Statusinformationen zu der Verbindung merken. Der Server gibt nach dem Absenden der Antwort alle Ressourcen, die zur Bearbeitung der Client-Anfrage nötig waren, wieder frei.

Eine Anfrage wird als *HTTP-Request* bezeichnet und enthält folgende Felder:

```
HTTP-Methode Request-URI HTTP-Version
```

Die *HTTP-Methode* kann `GET`, `HEAD`, `POST`, `PUT`, `DELETE` oder `TRACE` sein und beschreibt die Art, wie eine Datei vom Server an den Client zurückgesandt werden soll. Mit der Methode `GET` wird z.B. eine Datei angefordert, deren Name in der Request-URI angegeben wird, während mit der Methode `HEAD` nur Informationen über die Datei angefordert werden, nicht aber deren Inhalt. Diese Methode wird meistens verwendet, um das Datum oder die Größe der Datei zu erfahren. Die Adresse der Datei kann in dem *Request-URI* auf zwei Arten angegeben werden:

- die absolute Adresse der Datei, die auch den Namen des Servers enthält, auf dem die Datei gespeichert ist
- der relative Pfad der Datei ohne den Namen des Servers

Die erste Art wird angewandt, falls die Anfrage an Proxy-Server gerichtet ist, während die zweite Art von Clients nur an Web-Server gesendet wird. In diesem Fall wird in der Version 1.1 des HTTP zusätzlich noch ein `Host`-Header mit der Anfrage mitgeschickt, in dem der Name des Servers angegeben wird. Das Feld *HTTP-Version* gibt die Version des für die Anfrage verwendeten Protokolls an. Ein HTTP-Request kann also folgendermaßen ausschauen, falls ein Client von dem Server `www.myserver.tld` die Datei `file.html` anfordern will:

```
GET http://www.myserver.tld/file.html HTTP/1.1
```

falls die Anfrage über einen Proxy umgeleitet wird, oder

```
GET file.html HTTP/1.1
Host: www.myserver.tld
```

falls die Anfrage direkt an den Server gesendet wird, auf dem sich die Datei befindet.

Die Antwort auf eine Anfrage wird als *HTTP-Response* bezeichnet und besteht aus einer *Status-Zeile*, einem *Response-Header* und gegebenfalls dem Inhalt der angeforderten Datei, dem *Message-Body*. Die Status-Zeile gibt die verwendete HTTP-Version an, einen Status-Code (siehe dazu Anhang A) und die dazugehörige Beschreibung des Status-Code. In dem Response-Header sendet der Server zusätzliche Informationen über sich selbst an den Client, Informationen über die Art und den Typ der gerade übertragenen Datei oder auch über die Art der Authentifizierung, die notwendig ist, falls man auf eine geschützte Datei zugreift.

Durch das steigende Interesse am World Wide Web hat sich die Struktur der Web-Server, die Informationen zum Abruf anbieten, sehr stark verzweigt und erweitert, so daß eine Vielzahl kommerzieller Dienstleister das Bereitstellen der Server übernommen hat. Solche Dienstleister heißen Internet Service Provider und bieten ihren Kunden nicht mehr nur die Anbindung ihrer Unternehmen an das Internet, sondern unter anderem auch den Betrieb deren Web-Server oder auch nur das Bereitstellen der Informationsseiten im Web.

Wenn jemand im World Wide Web Informationen zur Verfügung stellen möchte, so müssen die gesamten Dokumente auf einem Web-Server gespeichert werden. Ob eine Firma ihren eigenen Web-Server betreibt oder sich aus Gründen der Effizienz oder der Bequemlichkeit mit mehreren anderen einen teilt, muß in Erwägung gezogen werden. Für jede Entscheidung gibt es mehrere Gründe und mehrere Lösungen. Die bekanntesten sind folgende:

Eigener Server im Haus: In diesem Fall betreibt eine Firma ihren eigenen Server, der in ihren Räumen auf dem Firmengelände plaziert ist und über den sie Informationen im Internet zur Verfügung stellt. Lediglich für die Anbindung an das Internet wird ein Provider ausgesucht.
Hier ist es sehr wichtig, daß innerhalb der Firma ausreichend qualifizierte Mitarbeiter zum Betreiben des Servers vorhanden sind oder für diesen Aufgabenbereich von außen hinzugenommen werden. Außerdem sollte dafür gesorgt sein, daß innerhalb der Firma genügend Hardware- und Netzwerkressourcen (wie z.B. PCs und Netz-Leitungen, die über genügend Bandbreite verfügen) vorhanden sind, um einen qualitativen und schnellen Informationsdienst realisieren zu können. Falls eine dieser beiden Bedingungen nicht erfüllt werden kann, sollte eine der nächsten Möglichkeiten in Betracht gezogen werden.

Eigener Server bei einem Provider: Falls eine Firma einen aufwendigen und vielbesuchten Web-Auftritt unterhalten möchte und deswegen einen eigenen Server-Rechner benötigt, für dessen Betrieb und Wartung aber keine eigenen Ressourcen wie Hard- und Software und Personal zur Verfügung stellen möchte, dann ist diese Variante eine sinnvolle Lösung.
Hier ist für den Fall, daß nur Informationen im Internet veröffentlicht und weitere Internetdienste nicht benötigt werden, keine eigene Internet-Anbindung der Firma notwendig, was aber in dieser Kombination eher nicht vorkommt. Für die Aktualisierung der Dateien auf dem Server wird ein physikalischer Zugriff der Firma auf den Server benötigt. Dieser Zugriff kann über eine Wähl- oder Standleitung zum Internet Provider hergestellt werden.
Bei einer Wählleitung wird eine Verbindung zum Internet nur dann aufgebaut, wenn Daten zwischen Internet und lokalem Rechner ausgetauscht werden. Im Gegensatz dazu bedeutet eine Standleitung eine permanente Verbindung zwischen Kunde und Provider. Diese lohnt sich natürlich nur dann, wenn höhere Datenvolumina ausgetauscht werden oder die Nutzungszeit der Leitung sehr hoch ausfällt.

Mieten von Speicherplatz auf dem Server eines Providers: Falls eine Firma nicht die Mittel, Kenntnisse oder das Personal dafür hat, einen eigenen Server zu betreiben, kann sie diese Aufgabe vollständig an einen Internet Service Provider vergeben. Die Firma muß sich dann weder um die richtige Installation und Konfiguration des Servers kümmern, noch muß sie ihn warten oder überwachen.

2.2.2 Web-Client

In der Client-Server-Umgebung des World Wide Web steuern die *Web-Clients* die Abläufe. Sie sind die Dienstnutzer, die Anfragen an Web-Server senden und anschließend eine Antwort empfangen. Web-Clients werden auch Browser genannt, wobei zwischen zeilenorientierten und graphischen Browsern unterschieden wird. Die zeilenorientierten Browser dürften mittlerweile fast gar nicht mehr angewendet werden, da graphische Browser eine sehr benutzerfreundliche und intuitive Bedienung und Nutzung der WWW-Dienste ermöglichen. Erst durch graphische Browser kam es zu dem großen Boom des WWW, da immer mehr Menschen leichter Zugang dazu fanden.

Ein Web-Client sendet also mit Hilfe des HTTP eine Anfrage an einen Web-Server, woraufhin dieser ihm ein *HTML-Dokument* zurückschickt. HTML ist die Seitenbeschreibungssprache im WWW, die die Realisierung der Hyperlinks ermöglicht. Dadurch ist die einfache Navigation zwischen Servern auf der ganzen Welt erst möglich. Der Web-Client interpretiert die HTML-Anweisungen aus dem erhaltenen Dokument und präsentiert dem Benutzer die Web-Seite.

Um dynamische Web-Seiten auf dem Client zur Verfügung zu stellen, muß nicht unbedingt erst ein Programm auf dem Server aufgerufen werden, wie das bei CGI-Programmen der Fall ist. Der Server kann z.B. ein in einem HTML-Dokument referenziertes Java-Applet als Antwort auf eine Anfrage zurücksenden. Ein Java-Applet ist ein in Java kodiertes Programm, das auf dem Client ausgeführt wird. Dieser muß allerdings über einen Java-Interpreter verfügen, was mittlerweile in jedem neueren Browser der Fall ist. Eine weitere Möglichkeit wurde von Netscape durch die Entwicklung der Programmiersprache JavaScript bereitgestellt. JavaScript-Anweisungen werden in den HTML-Code eines Dokuments eingebettet und somit zusammen mit dem Dokument an den Client übertragen. Der Netscape-Browser interpretiert daraufhin die JavaScript-Anweisungen und stellt den erzeugten Inhalt dar. Beide Mechanismen stellen komplexe Funktionalitäten zur Verfügung, um HTML-Seiten dynamisch zu gestalten. Weitere Mechanismen und neue Entwicklungen diesbezüglich (wie z.B. VRML oder Cascading Style Sheets) sollen hier nicht weiter erläutert werden.

2.2.3 Web-Proxy

Proxy-Server sind spezielle HTTP-Server, die in einem Netz gegenüber einem Web-Client die Rolle des Web-Servers übernehmen können, da sie die Anfragen entgegennehmen. Gegenüber Web-Servern nehmen sie die Rolle von Web-Clients ein, da sie die von Clients entgegengenommenen Anfragen an Server weiterrreichen. Die daraufhin ankommenden Antworten verschicken sie weiter an die Clients.

Durch Proxy-Server lassen sich die Zugriffe auf andere Server kontrollieren und einschränken. Zugriffsrechte können basierend auf dem Zugriffsprotokoll, der IP-Adresse oder dem Domänen-Namen des zugreifenden Benutzers oder der Ressource, auf die zugegriffen wird, vergeben werden.

Zusätzlich können Proxies als *Caching-Proxies* erweitert werden, so daß sie die Dateien, die sie als Ergebnis einer Anfrage an Clients weiterleiten, in einem Cache zwischenspeichern. Stellt ein Client eine Anfrage für ein Dokument, auf das bereits ein anderer Client zugegriffen hat, so übergibt ihm der Proxy die Kopie aus dem Cache. So muß keine neue Verbindung zu dem

Server aufgebaut werden, auf dem das Dokument ursprünglich gespeichert ist. Dadurch kann die Antwortzeit für den Client verkürzt und sogleich der Internet-Verkehr außerhalb des Netzes hinter dem Proxy reduziert werden. Andererseits ist mit veralteten Daten im Cache zu rechnen; diesem Problem wird teilweise mit Datumsangaben in dem HTTP-Header der Dokumente vorgebeugt. Das Datum gibt an, bis wann diese als nicht veraltet betrachtet werden können. Nicht jeder Server liefert diese Angaben automatisch in einem Header mit, so daß hier von seiten des Proxy-Servers Mechanismen festlegen sollten, wie lang Dateien im Cache noch als aktuell gelten oder nicht.

Proxy-Server können auch mehrfach hintereinandergeschaltet werden, so daß man von einer Verkettung oder "Proxy-Chaining" sprechen kann. So kann beispielsweise ein größeres Unternehmen jeder Abteilung einen eigenen Proxy-Server zur Verfügung stellen, um den Mitarbeitern so einen schnelleren Zugriff auf Dokumente zu ermöglichen. Die Abteilungs-Proxies werden von einem zentralen Proxy bedient, der den Zugang zu allen Ressourcen außerhalb des Unternehmensnetzes regelt und kontrolliert.

2.3 Funktionen des Anwendungsmanagements

Das Gebiet des Anwendungsmanagements befaßt sich mit der Verwaltung und Überwachung von Software-Anwendungen innerhalb eines Systems. Durch eine starke Verbreitung der *verteilten* Anwendungen, die immer umfangreicher und verzweigter gestaltet werden, nimmt das Anwendungsmanagement eine größere und auch gleichzeitig komplexere Rolle ein. Die Funktionalität einer verteilten Anwedung ist in eine Menge von kooperierenden Teilkomponenten unterteilt, die auf einem oder mehreren Rechnern realisiert sind. Diese Verteilung ist allerdings bezüglich der Funktionalität der gesamten Anwendung für den Benutzer transparent.

Obwohl das Anwendungsmanagement klar definierte Aufgabengebiete umfaßt, kann bei verteilten Anwendungen nicht mehr genau zwischen Aufgaben des Anwendungs-, Netz- oder Systemsmanagements unterschieden werden. Tritt ein Fehler während des Betriebs der Anwendung auf, so ist auf den ersten Blick schwer zu unterscheiden, ob der Fehler im Rahmen der Anwendung zu finden ist, ob es im Netz, über das die Komponenten der Anwendung miteinander kommunizieren, Probleme gibt, oder ob die Endsysteme, auf denen die Teilkomponenten der verteilten Anwendung ausgeführt werden, fehlerhaft funktionieren. So ist das Zusammenfassen aller Aufgaben im Rahmen des Anwendungs-, Netz- und Systemmanagements zum Ziel des integrierten Managements von großer Bedeutung.

Die Installation und Konfiguration einer Software-Anwendung in einem offenen, heterogenen und verteilten System gehört zu den Aufgaben des Systemmanagements. Die Verteilung und Installation der Software-Komponenten auf mehrere Rechner in einer verteilten Umgebung berücksichtigt die Verteilungs-Struktur der Endsysteme. Durch Konfiguration der Software-Komponenten werden diese an die Endsysteme, auf denen sie installiert und ausgeführt werden, angepaßt.

Die Kontrolle und Steuerung der verteilten Anwendungskomponenten, deren Verhalten während des laufenden Betriebs und die Überwachung der von ihnen erbrachten Dienste wird durch Aufgaben des Anwendungsmanagements übernommen.

Obwohl das Anwendungsmanagement im Rahmen eines integrierten Managements ein noch relativ junges Forschungsgebiet ist, haben die Desktop Management Task Force (DMTF) und die Firma Tivoli einige Definitionen diesbezüglich veröffentlicht.

Die DMTF befaßt sich in der *Software Standard Groups Definition* mit dem Inventory, also dem Sammeln von Softwaredaten zu Managementzwecken, während Tivoli in der *Applications Management Specification (AMS)* den Schwerpunkt auf das Monitoring oder Überwachen von verteilten Anwendungen legt.

2.3.1 DMTF Software Standard Groups Definition

Die DMTF hat sich auf die Standardisierung von Managementinformationen konzentriert, die sich aus den Anforderungen der Software-Verteilung, -Installation und -Konfiguration herausbilden. Durch Standardisierung dieser Informationen kann die Verteilung, Installation und Konfiguration von Software-Kopmonenten hersteller- und hardwareunabhängig dargestellt werden. Somit können die damit verbundenen Aufgabengebiete auf möglichst viele verteilte Anwendungen übertragen und vereinheitlicht und gleichzeitig die damit zusammenhängenden Aufgaben des Managements erleichtert werden.

Generell unterscheidet die DMTF bei der austauschbaren Managementinformation zwischen statischen und dynamischen Daten einer Software. Die *statischen* Daten sind Attribute, deren Wert der Softwarehersteller bei der Entwicklung der Anwendung initialisiert. Sie dienen der Identifikation der Software, z.B. durch Informationen über Name, Hersteller, Version usw. Weitere statische Informationen betreffen z.B. Abhängigkeiten zwischen den Komponenten, die bei der Installation festgelegt sein müssen, Unterstützung von seiten des Herstellers im Falle von Problemen (wie z.B. Telefonnummer oder Art der Unterstützung) oder auch Tips zur Wartung und Pflege der Software für Anwender oder Administratoren und für Managementanwendungen.

Dynamische Attribute verändern ihre Werte während oder nach der Installation. Dies sind z.B. der Ort oder Pfad, wo die Software installiert ist, oder auch der Status der Installation. Diese Informationen sind für die Überwachung der Verteilung und Installation der Anwendungs-Komponenten sehr wichtig.

2.3.2 Tivoli Applications Management Specification

In der AMS wird der gesamte Lebenszyklus einer Anwendung unter Managementgesichtspunkten berücksichtigt. Das ergibt sich aus den Problemstellungen des Managements von verteilten Anwendungen.

Aufgrund der Komplexität und der Heterogenität von verteilten Anwendungen, von denen einige Komponenten auf möglicherweise unterschiedlichen Systemen ausgeführt werden, ergeben sich Problemstellungen bei der Installation des Softwarepakets. Ein Administrator sollte Kenntnisse über die Architektur der Anwendung haben, über die Zielverzeichnisse, in denen die Dateien installiert sind, oder über die Möglichkeiten, die eine Anwendung bietet, um Informationen über den Status oder Ausführungszustand der Anwendung zu sammeln.

Mit steigender Anzahl der zu verwaltenden Anwendungen wird die Menge des Managementwissens immer größer, so daß auch der Verwaltungsaufwand zunimmt.

Ein weiteres Problem ergibt sich daraus, daß verteilte Anwendungen meistens nicht mit dem Ziel entwickelt wurden, leicht verwaltbar zu sein. Es stehen gemeinhin nicht genügend Schnittstellen zur Verfügung, um auf Daten zugreifen zu können, die für das Management der Anwendungen relevant sind. Das kann zu Schwierigkeiten bei der Deinstallation oder der Wartung und Pflege der Software führen.

Die häufige Aktualisierung von Software, die durch immer bessere Entwicklungsmöglichkeiten gegeben wird, verlangt nach entsprechenden Verwaltungswerkzeugen, die die Verteilung der Softwarekomponenten überprüfen und somit kennen, um zu wissen, welche Dateien gelöscht und welche aktualisiert werden müssen.

Nach Berücksichtigung der oben aufgeführten Gründe ist es notwendig, den *gesamten Lebenszyklus einer verteilten Anwendung* zu beobachten, um sie angemessen überwachen und steuern zu können. Dieser Zyklus umfaßt folgende Phasen:

- Struktur und Topologie der Anwendung,
- Installation und Verteilung der Anwendung,
- Abhängigkeiten innerhalb der Anwendung,
- Monitoring und Ereignismeldungen sowie
- operationelle Kontroll-Tasks.

Schon bei der Entwicklung der Software muß der Aspekt des Managements berücksichtigt werden. So sollte eine klare *Strukturierung und Topologie* der Softwarekomponenten einer verteilten Anwendung realisiert und definiert werden, um die Abhängigkeiten zwischen den Komponenten festzuhalten.

Diese Informationen sind für die *Installation und Verteilung* der Software wichtig, die zentral ausgeführt werden, was bei verteilten Anwendungen unerläßlich ist, nachdem es, wie schon erwähnt, zu solchen Fällen kommen kann, daß die Software auf Rechnern läuft, die über größere geographische Entfernungen verteilt ist. Die Zerlegung der Anwendung in Komponenten und die Festlegung der Zielverzeichnisse, in denen die Dateien installiert werden sollen, ist auch bei der Deinstallation der Anwendung von großer Wichtigkeit. Diese Gesichtspunkte dürfen nicht außer acht gelassen werden.

Zusätzlich ist nicht nur die Festlegung aller Komponenten wichtig, die von anderen abhängig sind, sondern auch die jeweilge *Art der Abhängigkeit*, also die Bedingungen, die für jede Komponente erfüllt sein müssen, um letztendlich die korrekte Funktion der verteilten Anwendung zu gewährleisten.

Während der aktiven Ausführung der Anwendung kann man mit Hilfe des *Monitoring und der Events* den aktuellen Zustand der Software überprüfen. Dadurch kann die verteilte Anwendung aktiv gesteuert, können Fehler während des Betriebs analysiert und behoben oder können Tests durchgeführt werden, um die Performance der Anwendung zu überprüfen.

Zum aktiven Steuern einer Anwendung gehören auch die *operationellen Kontroll-Tasks*, mit denen man z.B. Prozesse der Anwendung explizit starten oder stoppen oder auch Nachrichten an bestimmte Benutzer senden kann. Diese Kontroll-Tasks sollen so konfiguriert sein, daß sie die Aufgaben eines Administrators vereinfachen und möglichst automatisch ausführen.

Zusammenfassend läßt sich sagen, daß die Managementaufgaben, die den gesamten Lebenszyklus der Anwendung berücksichtigen, die Wartung und Pflege von verteilter Software erleichtern und effektiv gestalten sollen.

2.4 Beispiel-Szenarien beim Web-Management

Das World Wide Web ist ein weit verzweigtes Netz von Informationen. Diese Informationen stehen in Form von Texten, Bildern oder Multimediadaten (wie z.B. Ton und Video) auf Web-Servern in Dateien gespeichert, zur Verfügung.

Für den Zugriff auf diese Daten bzw. Dateien benötigt man einen Web-Client, auch Browser genannt. Clients stellen Anfragen an Web-Server, sogenannte Requests. Ein Server bearbeitet die ankommenden Requests und schickt die Antwort als Response an die entsprechenden Clients zurück.

Die Verfügbarkeit des Dienstes WWW hängt stark von der Performance der Web-Server ab. Das Interesse an der Leistung von Web-Servern kann dabei aus unterschiedlichen Gesichtspunkten betrachtet werden (siehe dazu Abb. 2.3).

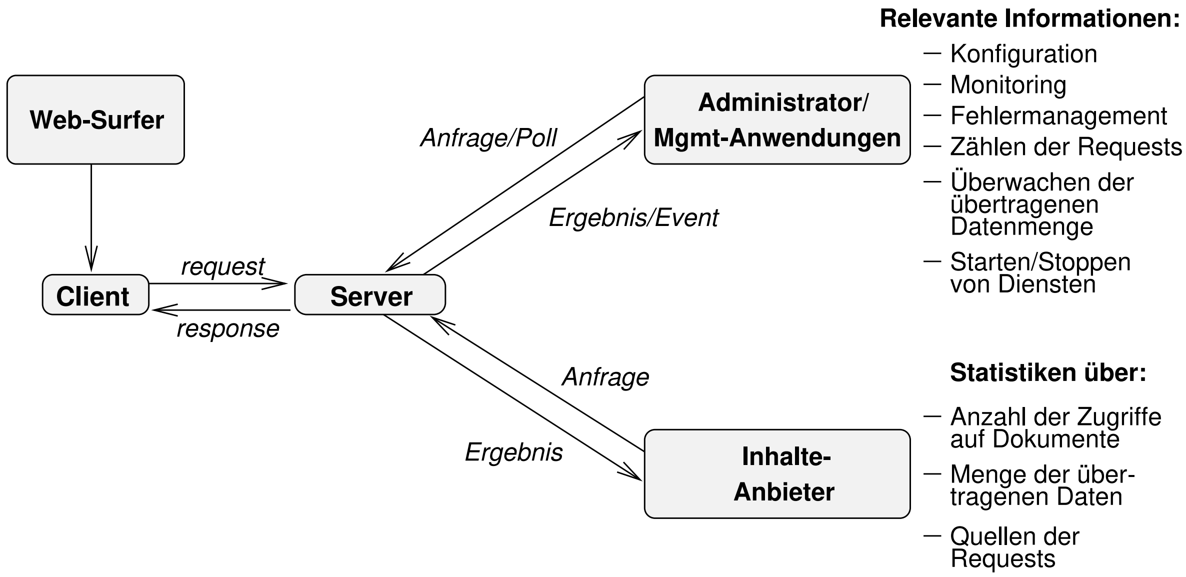

Abbildung 2.3: Beispiel-Szenarien beim Web-Management

Der **Server-Betreiber**, der die Systeme, Server-Software und Web-Dokumente betreut, benötigt Managementinformationen über die Ressourcen. So kann er die korrekte Funktion des Servers überwachen, dessen Performance, wie viele Anfragen er in welcher Zeitspanne bearbeitet, ob er sie richtig bearbeitet usw.

Ein **Inhalte-Anbieter** dagegen interessiert sich hauptsächlich für Statistiken über die Anzahl der Zugriffe auf seine Daten, über die Menge der dabei übertragenen Daten, über die Quellen der Anfragen usw.

Ein **Web-Surfer** legt dagegen vermehrt Wert auf ein interessantes und in sich schlüssiges Angebot an Daten, in dem keine veralteten Links vorkommen, die auf nicht mehr vorhandene Dateien verweisen; vor allem legt er Wert auf eine hohe Verfügbarkeit der Daten und eine schnelle Bearbeitung seiner Anfragen.

Dabei kann anhand folgender Skizze (Abbildung 2.4) anschaulich dargestellt werden, daß normalerweise nur wenige Server-Betreiber oder Server-Betreuer (Administratoren) Zugriff auf einen Server haben, dagegen mehrere Inhalte-Anbieter Informationen über diesen Server im World Wide Web zur Verfügung stellen, und eine sehr große Anzahl an Web-Surfern auf diesen Server zugreifen, um die Information zu lesen.

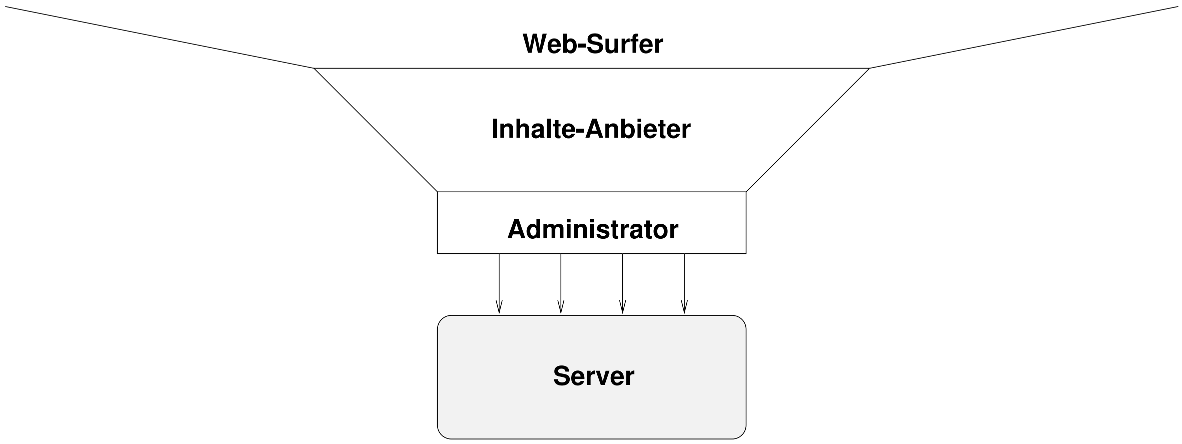

Abbildung 2.4: Anteil der Nutzer-Gruppen an den Zugriffen auf einen Web-Server

2.4.1 Anliegen eines Server-Betreibers

Für den Server-Betreiber ist die Anzahl der *richtig* bearbeiteten Zugriffe auf die auf dem Server befindlichen Dokumente von Bedeutung, um seinen Kunden eine hohe Performance und fehlerfreies Funktionieren der Server bieten zu können. Dafür muß ihm das Managementwissen und die Managementanwendung neben Monitoring der Server auch ein aktives Eingreifen und das Konfigurieren der Server ermöglichen, um Leistungs- und Fehlermanagement betreiben zu können.

Installation und Konfiguration des Web-Servers

Um spätere Fehlfunktionen während des Betriebs des Web-Servers möglichst ausschließen zu können, ist von Anfang an auf eine richtige Installation und Konfiguration zu achten. Ebenso besteht für einen Server-Administrator im Laufe des Server-Betriebs die Notwendigkeit, die Konfiguration des Servers zu ändern und den Server somit an neue Anforderungen anzupassen.

Falls ein Betreiber für mehrere Kunden mehrere Server-Maschinen betreut oder vielleicht virtuell auf einer Maschine realisiert, stellen sich ihm unterschiedliche Management-Situationen, die er bewältigen muß. Die Frage, ob ein virtueller Server pro Kunde eingerichtet wird, muß schon bei der Konfiguration bedacht werden. Hier muß eine richtige Einteilung und Verteilung der vorhandenen Ressourcen vorgenommen werden, d.h. wieviel Speicherplatz für jeden Kunden zur Verfügung gestellt wird, wo die Dateien jedes Kunden abgelegt werden usw. Falls mit der Zeit dann weitere Kunden hinzukommen, werden neue Ressourcen installiert und konfiguriert, die Konfiguration des Servers wird geändert, und neue Dateien werden in dem Document-Store des Servers gespeichert.

Monitoring und Überwachung der Performance des Web-Servers

Während des Betriebs eines Servers besteht die Notwendigkeit, daß darauf ablaufende Prozesse überwacht und die Auslastung des Servers und dessen Ressourcen überprüft werden, falls es zu Engpässen kommt. Ebenso sind Angaben über den über den Server abgewickelten Verkehr notwendig, um dessen Auslastung zu untersuchen und vielleicht bei Bedarf mehr Ressourcen, wie z.B. Speicherplatz oder Bandbreite, zur Verfügung zu stellen.

Analysen über die Anzahl der bearbeiteten Requests können auch dazu genutzt werden, um die Performance des Servers zu untersuchen. Es kann die Anzahl der *richtig* oder *falsch* bearbeiteten Requests mitverfolgt werden, die Zeiten, in denen der Server sehr stark oder sehr schwach ausgelastet ist, um vielleicht die Nutzung der Ressourcen einzuteilen und zu koordinieren, oder eben auch der Verkehrsfluß an Bytes, der über den Server ausgetauscht wird. Die Belastung des Servers ist ein wichtiger Aspekt für den Betreiber, da er im Notfall neue Hardware zur Verfügung stellen muß, um eine höhere Performance seines Servers zu gewährleisten.

Sicherheitseinrichtungen

Ein Server-Betreiber muß den Server und die angebotenen Dokumente gegen unbefugte Zugriffe schützen. Dabei möchte er festlegen können, welche Benutzer von welchen Domänen auf angebotene Dateien zugreifen können oder nicht. Er möchte die Zugriffe nur für Benutzer erlauben, die sich mit einem bestimmten Namen und dem dazugehörigen Paßwort, oder abhängig ihres Hosts oder der Domäne, von der aus sie auf die angebotenen Dateien zugreifen, identifizieren können. Außerdem ist es vielleicht von Vorteil, falls der Betreiber festlegen kann, ob sich der Zugriff von bestimmten Benutzern auf einzelne Dokumente oder auf Verzeichnisse (und somit auf die gesamt darin enthaltenen Dateien) beschränken soll, oder ob z.B. nur der Zugriff auf bestimmte Dateitypen erlaubt oder verwehrt wird, wie z.B. CGI-Programme.

Ein weiterer wichtiger Aspekt der Sicherheitseinrichtungen gegenüber den Benutzern des Servers besteht darin, die sichere Übertragung der Daten und deren Verschlüsselung gegenüber Dritten zu gewährleisten. So ist es sehr wichtig für manche Anbieter, wie z.B. Direktbanken, die ihre Dienste auch über das Internet anbieten, daß die privaten Daten ihrer Kunden verschlüsselt übertragen werden, da es sich um Kreditkartennummern, Vermögensangaben oder Adressen der Kunden handeln kann. Somit kann die Sicherheit der Kunden gegenüber kriminellen Angriffen von Dritten besser geschützt und mehr Kunden dazu animiert werden, die Dienste des Anbieters zu nutzen.

Falls ein Server-Betreiber seinen Kunden, die auf dem Server Dokumente für Benutzer öffentlich anbieten, erlaubt, diese selbst auf den Server zu spielen, sie bei Bedarf zu löschen oder sie in andere Verzeichnisse zu verschieben, so ist er darauf angewiesen, Bestimmungen festzulegen und Sicherheitseinrichtungen zu aktivieren, die ihn bestimmen lassen, welche Benutzer sich auf dem System einloggen dürfen, in welchen Verzeichnissen diese Dateien ändern oder löschen können und um welche Art des Zugangs es sich überhaupt handelt (ftp, telnet usw.). Diese Funktionalität ist sehr wichtig für Internet Service Provider, die Web-Server für ihre Kunden verwalten oder, falls es sich um Privatkunden handelt, diesen erlauben, in Benutzerverzeichnissen Dateien abzulegen und somit eine eigene kleine Homepage auf dem Server des Providers zu realisieren.

Abrechnung der angebotenen Dienste

Ebenso ist es für Internet Service Provider sehr wichtig, daß sie die angebotenen und geleisteten Dienste gegenüber ihren Kunden abrechnen können. Für die Abrechnung der zur Verfügung gestellten Dienste braucht ein Betreiber die Übersicht über die übertragene Datenmenge jedes einzelnen seiner Kunden, über die Menge der auf dem Server gelagerten Daten und über die Dienstgüte, mit der die eingehenden Anfragen bearbeitet werden.

Eine Klassifizierung der abzurechnenden Dienste nach der Art der übertragenen Dokumente oder der Anzahl der übertragenen Bytes deckt die Bedürfnisse detaillierter ab. Außerdem ist ein Provider in manchen Fällen daran interessiert, dem Inhalte-Anbieter Angaben über die Benutzer zur Verfügung zu stellen, die auf die Dokumente zugegriffen haben, verbunden mit Angaben darüber, welche Dokumente und wie viele Bytes von diesem abgerufen wurden. So kann der Inhalte-Anbieter die ihm durch die Übertragung von Dokumenten aus seinem Angebot entstehenden Kosten an deren Verursacher, in diesem Fall den Benutzer der Daten, übertragen.

Fehlermanagement

Sehr wichtig im Betrieb eines Web-Servers sind Aufgaben aus dem Gebiet des Fehlermanagements. Fehler müssen nicht nur lokalisiert, sondern auch analysiert und anschließend behoben werden. Fehler können z.B. beim Übertragen von Dateien entstehen, beim Ausführen von CGI-Programmen oder beim Bearbeiten von Anfragen. Ein fehlerhaftes Programm, das ein Server zum Bearbeiten einer Anfrage benutzt, kann den Betrieb des Servers unterbrechen oder stören. Auch eine fehlerhafte Konfiguration kann das richtige Bearbeiten von Anfragen beeinflussen. Deswegen sind Einrichtungen zur konstanten oder periodischen Überwachung der Server-Funktionen sehr wichtig, um den Fehlerursachen auf den Grund zu kommen und um Fehler leichter lokalisieren zu können.

2.4.2 Anliegen eines Inhalte-Anbieters

Das World Wide Web gewinnt bei kommerziellen Anbietern immer mehr an Bedeutung. Das Web bietet ihnen ein Umfeld, in dem sie weltweit Produkte vorstellen und anbieten können. Ganze Kataloge mit Photos und einer Beschreibung der Produkte, verbunden mit Online-Bestellformularen, können in dieser Umgebung zu einer weltweiten Verbreitung der Ware genutzt werden.

Somit kann ein großes Publikum angesprochen werden, was allerdings eine gezielte Aufteilung der Kunden nach verschiedenen Ländern und Sprachen bedeuten kann. Je nachdem, ob ein Anbieter unterschiedliche Kundenkreise anspricht, möchte er auch wissen, welche Kunden sein Web-Angebot häufiger nutzen, aus welchen Ländern die Anfragen vermehrt erfolgen usw. Für den Inhalte-Anbieter sind Informationen über die Menge der übertragenen Daten oder die Quellen der Requests deswegen von Bedeutung, da er dadurch neue Werbekunden anlocken oder mit speziellen Werbebotschaften in anderen Sprachen die Zielgruppe ansprechen kann, die aus fremden Ländern auf das Angebot zugreift.

Es gibt viele Möglichkeiten, exakte **Kundenprofile** nach eigenen Wünschen zu erstellen und diese Daten für Marketingzwecke einzusetzen. So ist es auch möglich zu untersuchen, zu welchen Uhrzeiten die Kunden vermehrt auf das Angebot zugreifen, oder auch speziell zu welchen Tageszeiten aus welchen Ländern am häufigsten welche Seiten aufgerufen werden. Diese Angaben können dazu genutzt werden, in dieser Zeitspanne auf den bestimmten Seiten vermehrt Werbung zu schalten oder Sonderangebote anzupreisen.

Weiter gibt es für einen kommerziellen Anbieter die Möglichkeit zu untersuchen, von welchen Seiten im Internet denjenigen Links gefolgt wird, die auf seinen Web-Auftritt verweisen. So kann er auch diese Angaben zu Marketingzwecken nutzen und auf diesen Seiten eine bessere Präsenz zeigen, um mehr Kunden zu locken.

Diese Art von Untersuchungen finden ihre Anwendung aber nicht nur in der kommerziellen Nutzung des Internets. Universitäten oder öffentliche Einrichtungen möchten auch detaillierte Profile über ihre Kunden und Besucher aus dem Netz haben, um sich ihrem Kundenkreis vorteilhaft zu präsentieren und somit vermehrt Interesse an ihren Aktivitäten und Angeboten zu wecken. Sie interessiert auch, zu welchen Zeiten auf welche Angebote am häufigsten zugegriffen wird, welcher Typ von angebotenen Dateien (z.B. Text-, Bilder- oder Tondateien) am meisten Interesse wecken oder wann die meisten Fehler beim Bearbeiten der Anfragen auftreten, um Engpässe diesbezüglich zu vermeiden.

Aus Sicht eines Inhalte-Anbieters sind also **Langzeitstatistiken** über die Vorlieben der Benutzer und deren Verhalten beim Zugriff auf die von ihm angebotenen Dateien interessant. Für manche Anbieter sind auch Statistiken über einzelne Dokumente von Bedeutung, wie oft diese abgerufen werden, welche Benutzer oder Organisationen diese am meisten abrufen, wie viele Anfragen in bestimmten Zeitintervallen dafür ankommen und wie viele Bytes dabei oder insgesamt übertragen werden. Ebenso sind **Kundenanalysen** z.B. bezüglich des Ursprungslandes, von wo die Anfragen kommen, oder der Tageszeit, zu der die meisten Anfragen gestellt werden, interessant.

2.4.3 Wünsche eines Web-Surfers

Das Hauptanliegen eines Web-Surfers ist ein schneller Zugriff auf angebotene Dokumente und die Aktualität der Seiten. Er möchte nicht von veralteten Links behindert werden, von Seiten, auf denen ihm mitgeteilt wird, daß angeforderte Dokumente nicht gefunden werden können, oder von Meldungen, daß der angewählte Server gerade nicht erreichbar ist. Eine hohe Verfügbarkeit des Servers und der darauf befindlichen Dokumente ist von jedem Server-Betreuer unbedingt zu verfolgen. Ebenso ist darauf zu achten, daß Web-Surfer das Bedürfnis haben, daß ihre Daten in gewissen Fällen besonders vor dem Zugriff Dritter geschützt und verschlüsselt werden. Sowohl Server-Betreuer als auch Inhalte-Anbieter müssen die Wünsche und Bedürfnisse ihrer Kunden ernst nehmen und sie weitestgehend erfüllen, falls sie ihren gewonnenen Kundenkreis erhalten und auch erweitern wollen.

2.4.4 Überwachung und Steuerung weiterer Web-Komponenten

Nicht nur Server, sondern auch **Web-Clients** müssen im Web verwaltet werden. So ist es wichtig, daß bei der Installation der Clients oder bei auftretenden Problemen der Administrator die Konfiguration auch entfernt vornehmen kann. Im Falle von Fehlfunktionen soll auch das Starten und Stoppen von Prozessen (wie z.B. bei Java-Applets) möglich sein.

Außer Web-Servern und -Clients gibt es im Web noch **Proxy-Server**. Sie sind eine spezielle Art von HTTP-Servern, die hinter einer Firewall plazierten Clients den Zugriff auf das Internet ermöglichen. Sie nehmen Requests von den Clients an und übergeben sie weiter an Server außerhalb der Firewall. Die ankommenden Responses von außerhalb der Firewall werden überprüft und an die entsprechenden Clients innerhalb der Firewall weitergeleitet. Um das eigene Netz hinter der Firewall zu schützen, sind spezielle Sicherheitsbestimmungen im Rahmen des Proxy-Managements zu fällen und durchzusetzen. Außerdem ist auf eine richtige Konfiguration des Proxy-Servers zu achten, da hiermit das richtige Funktionieren während des Betriebs mit einer hohen Wahrscheinlichkeit garantiert werden kann.

2.5 Zusammenfassung

Um die gängigen Szenarien beim Web-Management geeignet zu bearbeiten, ist ein breites Managementwissen notwendig und ebenso qualitative Managementanwendungen, die dieses Wissen abfragen, es verwerten und anwenden können, um den Zugriff auf das Web so hoch wie möglich zu halten. Ein aktives Steuern der einzelnen Anwendungs-Komponenten muß mit Hilfe der Managementanwendung zum Zweck eines effizienten Managements unbedingt möglich sein.

Als Ergänzung zu dem beispielhaften Einblick in einige Managementszenarien beim Web-Management werden in Kapitel 4 konkrete Managementanforderungen anhand typischer Szenarien ausgearbeitet.

Kapitel 3

State of the Art

3.1 Ansätze der IETF für das Management des Dienstes WWW

Die *Internet Engineering Task Force (IETF)* hat ein Informationsmodell definiert, wonach das Managementwissen über eine Ressource in einer Managementinformationsbasis (Management Information Base - MIB) gesammelt wird. Die Information wird in Form von Managementobjekten gespeichert, wobei dies nicht im objektorientierten Sinn gemeint ist: die Managementobjekte hier sind nur einfache Variablen oder Tabellen, die in der Form eines Baumes strukturiert sind.

Durch das herstellerunabhängige Speichern der Managementinformation über Clients und Server im Web in dafür definierte MIBs können Managementanwendungen über das weit verbreitete *Simple Network Management Protocol (SNMP)* diese mit gewohnten Methoden überwachen und steuern.

Erste MIBs für die Überwachung von Web-Servern befinden sich in der Entwicklung. In dem Internet-Draft *Definitions of Managed Objects for WWW Services* wird eine MIB definiert, deren Ziel es ist, Managementobjekte festzulegen, die in einer heterogenen Umgebung Leistungs- und Fehlermanagement von WWW-Diensten erlauben.

Zusätzlich befaßt sich eine Arbeitsgruppe der IETF mit der Anwendung der Standards zum Überwachen und Steuern von Web-Servern und veröffentlicht die Ergebnisse in dem Request for Comments Nr. 2039 *Applicability of Standards Track MIBs to Management of World Wide Web Servers*.

Dabei unterscheiden sie zwei Sichtweisen eines Servers. Einerseits ist ein Server ein Computer, der über Speicher verfügt, über ein Betriebssystem und die Server-Software. Somit kann er aus der Sicht der Ressourcenbenutzung administriert werden. Variablen dafür werden in der *Host Resources MIB* definiert. Andererseits kann er als eine Black Box gesehen werden, die Requests der Clients bearbeitet und sie als Responses zurückschickt. Für diesen Fall werden dienstspezifische Managementinformationen benötigt. Diese Art von Managementwissen ist in der *Network Services Monitoring MIB (NSM-MIB)* definiert.

Ist dagegen für einen Administrator von Bedeutung, welche und wie viele Prozesse der Server-Software laufen, ihre Konfiguration oder der aktuelle Status des Servers, dann findet er in der

System Application MIB (sysApplMIB) Informationen darüber.

Als Ergänzung sind weitere spezielle, anwendungsspezifische Attribute zu der Konfiguration und Arbeit mit Server-Software in der *Application Management MIB (applmib)* definiert.

3.1.1 Host Resources MIB

Die Host Resources MIB definiert Objekte, die für das Management von Endsystemen geeignet sind. Die Objekte sind in sechs Gruppen eingeteilt. Die ersten drei Gruppen enthalten Attribute zu Betriebssystemressourcen (*hrSystem*), Speicherressourcen (*hrStorage*) und zu den logischen Geräten eines Endsystems (*hrDevice*). Die anderen drei Gruppen enthalten Attribute zu der auf dem Endsystem lokal installierten Software (*hrSWInstalled*), zu den sich gerade in Ausführung befindenden Software-Komponenten (*hrSWRun*) und zu den von den Software-Komponenten benutzten Speicher- und Prozessor-Ressourcen (*hrSWRunPerf*). Dabei erlaubt diese MIB nur das Monitoring der Endsysteme und betrachtet verteilte Systeme und verteilte Anwendungen gar nicht. Ein aktives Steuern der Ressourcen ist damit nicht möglich.

3.1.2 Network Services Monitoring MIB (NSM)

Diese MIB definiert Objekte, die ein Monitoring von Anwendungen erlaubt, die Netzdienste oder verteilte Systemdienste zur Verfügung stellen. Die festgelegten Attribute sind in zwei Tabellen gespeichert. Die erste Tabelle (*applEntry*) enthält Informationen über jeden der betrachteten Dienste, über die aktuellen und die gesamten Abhängigkeiten der Komponenten, während die zweite Tabelle (*assocEntry*) detailliertere Einträge zu allen offenen Kommunikationsverbindungen der Anwendungen enthält.

3.1.3 Application MIB

System Application MIB (sysApplMIB)

Die System Application MIB definiert Objekte für das Konfigurations-, Fehler- und Leistungsmanagement von Anwendungen. Verteilte Anwendungen werden dabei nicht betrachtet, sondern nur Anwendungen, die auf einem Rechner ausgeführt werden und eine Ansammlung von ausführbaren und anderen Arten von Dateien sind. Die Attribute der MIB sind in zwei Gruppen eingeteilt und sind eine Erweiterung der Attribute aus der Host Resources MIB. In der ersten Gruppe (*System Application Installed*) sind Objekte zu der installierten Software definiert, während in der zweiten Gruppe (*System Application Run*) Einträge zu jeder während der Laufzeit ausgeführten Anwendung zu finden sind. Sämtliche Einträge in der MIB erlauben nur ein Monitoring der Anwendungen, also kein aktives Steuern oder Eingreifen in den Ablauf der Anwendung. Die Werte der MIB-Einträge werden unter anderem nur mit den Möglichkeiten des Betriebssystems gewonnen, nicht aus speziellen Eigenschaften und Methoden der Anwendungen. Dafür wurden anwendungsspezifische MIBs entwickelt, die eine Instrumentierung der Anwendung voraussetzen, um managementrelevante Informationen speziell über die Anwendung zu gewinnen.

Application Management MIB (applmib)

Im Rahmen er Application Management MIB sollen die generischen, in den zuvor besprochenen MIBs definierten Objekte spezifiziert und für bestimmte Arten von Anwendungen verfeinert werden. Hierfür ist aber eine Instrumentierung der jeweiligen Anwendungen notwendig, so daß über Management-Schnittstellen notwendige Informationen abgefragt werden können. Die MIB definiert Attribute für das Anwendungsmanagement und teilt sie in Tabellen ein. Mit Hilfe der definierten Objekte ist eine Sicht auf die Dienste der Anwendung und die Beziehungen der Anwendungskomponenten zu diesen Diensten möglich. Außerdem wird für jede offene Datei ein Eintrag in eine weitere Tabelle ergänzt, um so einen Überblick über alle I/O-Aktivitäten auf Dateien der installierten Anwendung zu erhalten. Ebenso werden Attribute definiert, die es erlauben, jeden Prozeß zu erfassen, der eine Datei aufhält, oder Informationen über jede offene Verbindung und die darüber stattfindenden Transaktionen der Anwendungskomponenten zu sammeln. Auch die aktuelle Ressourcenbenutzung durch die Anwendungskomponenten kann mit Hilfe der definierten Attribute der MIB beobachtet werden. Attribute einer Tabelle sind speziell dazu definiert, um dem Administrator die Möglichkeit zu geben, aktuell ausgeführte Anwendungskomponenten zu kontrollieren oder anzuhalten, Prozesse zu stoppen und Komponenten zu rekonfigurieren.

WWW-MIB (applmib-wwwmib)

Um WWW-spezifische Attribute und Eigenschaften zu berücksichtigen, die auf keinen Fall in die Anwendungs-MIBs aufgenommen werden können, wurde die WWW-MIB entwickelt. Diese liegt momentan nur als Internet-Draft vor und ist noch nicht vollständig standardisiert. Sie verfolgt das Ziel, WWW-spezifische Objekte aufzunehmen, die für möglichst viele Anwendungen gelten, die WWW-Dienste jeder Art realisieren. Als WWW-Dienste werden Aktionen angesehen, die auf Dokumente ausgeführt werden können. Clients können Dokumente mit Hilfe von Requests anfordern, Server stellen Dokumente bereit und senden sie als Response nach einem Request an Clients zurück. Proxy-Server können sowohl die Rolle von Clients als auch die der Server annehmen und Dokumente entsprechend bearbeiten. In der MIB wird zwischen eingehenden und ausgehenden Requests unterschieden, um so Statistiken über Clients, Server und Proxies mit nur einem Set von Attributen erstellen zu können. Die Attribute der WWW-MIB sind in drei Gruppen eingeteilt. In der ersten Gruppe (*Service Information*) werden Informationen über sämtliche WWW-Dienste definiert, die auf einem Host laufen. In der zweiten Gruppe (*Protocol Statistics*) werden Informationen über den Umfang der erhaltenen oder gesendeten Daten eines Dienstes definiert. Die dritte Gruppe der Attribute (*Document Statistics*) enthält Informationen über Dokumente, auf die schon zugegriffen wurde. Darunter fallen Statistiken über die Anzahl der letzten Versuche, auf alle oder auch nur auf bestimmte Dokumente zuzugreifen, oder der Datenmenge, die mit den Dokumenten übertragen wurde. Auch diese MIB erlaubt nur das Monitoring bestimmter Informationen über einzelne Aufträge, ein aktives Steuern und Eingreifen in die Bearbeitung der einzelnen Aufträge ist nicht möglich.

3.1.4 Wertung dieses Modells

Dadurch, daß das Informationsmodell des Internet-Managements bewußt einfach gehalten wurde, konnte es schnell angewendet werden, und es kam somit zu einer großen Verbreitung auf dem Gebiet des Netz- und System-Managements. Damit wurde eine unkomplizierte Definition von Managementinformation verfolgt, verbunden mit schnell verfügbaren lauffähigen Implementierungen des Modells. Diese Einfachheit wird häufig als Vorteil angesehen, wobei sie jedoch einige gravierende Nachteile aufwirft:

- Durch das Anwenden des Informations-Modells zur Strukturierung der Managementinformation ist man von vornherein auf SNMP als Managementprotokoll festgelegt.
- Die Strukturierung der Managementinformation in einer MIB ist für komplexe Managementaufgaben ungeeignet. Der fehlende objektorientierte Ansatz läßt umfangreichere Operationen auf den Managementobjekten nur eingeschränkt zu. Hier wird eine Aktion durch Zuweisung eines Wertes an eine Variable ausgelöst, nicht durch Aufrufen von Operationen auf Objektinstanzen.
- Der Schwerpunkt wird bei SNMP-MIBs häufig auf das Monitoring gelegt, nicht auf aktives Management. Auch hier würde ein objektorientierter Ansatz Operationen auf Managementobjekte erlauben und die Managementaufgaben erleichtern.
- Außerdem kommt es zu einer Überlappung der Managementinformation zwischen MIBs und deren Erweiterungen, den thematisch nachfolgenden MIBs, so daß die Zusammenhänge nicht immer klar werden. Konzepte der Objektorientierung könnten solche Probleme verhindern. Das Prinzip der Vererbung würde die Wiederverwendung von Code ermöglichen und damit die mehrfache Nutzung der Operationen und Attribute von Managementobjekten.

3.2 Das Referenzmodell für Open Distributed Processing (RM-ODP)

Das ODP-Referenzmodell der ISO bietet einen standardisierten Rahmen für die Entwicklung und Beschreibung von verteilten Anwendungen, die in einer offenen, heterogenen und verteilten Systemumgebung interagieren können sollen. Das heißt, daß sämtliche Protokolle und Programmierschnittstellen standardisiert und offengelegt sind, daß eine Vielfalt der Systemkomponenten von unterschiedlichen Herstellern gegeben und akzeptiert wird und daß die verteilten Anwendungen in unterschiedlichen Programmiersprachen kodiert sein können.

Die Anforderungen von verteilten Anwendungen, wie z.B. die Transparenz der Verteilung, die Flexibilität der Anwendungskomponenten oder die Portabilität auf andere Plattformen, sollen ebenso berücksichtigt und im Referenzmodell eingebunden werden. Das Referenzmodell stellt einheitliche Konzepte und Regeln zur Verfügung, um eine Architektur für verteilte Anwendungen zu beschreiben und betrachtet diese als bestehend aus Objekten, die an festgelegten Schnittstellen miteinander Informationen austauschen und somit Dienste anbieten bzw. nutzen.

Um die Komplexität einer verteilten Anwendung faßbarer und verständlicher zu machen, wird die Sichtweise des RM-ODP auf verteilte Anwendungen in unterschiedliche Sichten, sogenannte *Viewpoints*, unterteilt. Das Referenzmodell bietet fünf Viewpoints an, die unterschiedliche Aspekte und Phasen in der Entstehung und dem Betrieb einer verteilten Anwendung abdecken, ohne sich merklich zu überschneiden. In jedem Viewpoint werden nur bestimmte Aspekte zu einem Hauptthema berücksichtigt, wobei unwichtige oder nicht dazugehörende Informationen weggelassen werden.

Der **Enterprise Viewpoint** beschreibt die Anforderungen eines Unternehmens an die verteilte Anwendung, deren Ziele und Vorstellungen, sowie den Verwendungszweck der Anwendung innerhalb eines Systems. Im **Information Viewpoint** werden die Struktur der Information und der Informationsfluß innerhalb der verteilten Anwendung festgelegt. Der **Computational Viewpoint** definiert die Objekte und deren Dienste, die sie anderen Objekten zur Verfügung stellen, ohne die Verteilung der gesamten Anwendung zu berücksichtigen. Diese Verteilung oder Infrastruktur der Anwendung wird im Rahmen des **Engineering Viewpoint** definiert, indem generische Infrastrukturobjekte eingeführt werden, die die Verteilung realisieren und somit eine Kommunikation der Objekte der verteilten Anwendung ermöglichen. Die technische Realisierung und Implementierung der verteilten Anwendung sowie die Spezifizierung der Hard- und Software, die angewendet werden soll, werden im **Technology Viewpoint** festgelegt.

Jeder Viewpoint wird durch eine *Viewpoint Language* definiert, die das Vokabular und die grammatikalischen Regeln zur Beschreibung der Konzepte des Viewpoints festlegt. Im Rahmen dieser Diplomarbeit sind die zwei Viewpoints **Computational Viewpoint** und **Engineering Viewpoint** von großer Bedeutung, da sie die Objekte beschreiben, die für die Realisierung von Managementagenten benötigt werden, die innerhalb einer integrierten, verteilten Managementanwendung agieren.

3.2.1 Computational Viewpoint

Der Computational Viewpoint zerlegt eine verteilte Anwendung in Objekte, die an festgelegten Schnittstellen miteinander agieren, ohne die Infrastruktur der Verteilung zu berücksichtigen, die ja vom Engineering Viewpoint abgedeckt wird. Dabei werden nur die Schnittstellen definiert, ohne Rücksicht auf die Implementierung zu nehmen. Mit Hilfe der an den Schnittstellen verfügbaren Operationen bieten die Objekte anderen Objekte ihre Dienste an. Im Computational Viewpoint werden die Objekte als **Computational Objects** und die Schnittstellen als **Computational Interfaces** bezeichnet. Bezüglich einer Schnittstelle kann ein Objekt verschiedene Rollen annehmen, wie z.B. Client|Server oder Producer|Consumer. Durch **Templates** werden die Eigenschaften der Computational Objects und der Computational Interfaces spezifiziert. Ein **Computational Object Template** legt alle Informationen fest, die zur Instantiierung des Objekts zur Laufzeit erforderlich sind. Darunter fallen die Anforderung des Objekts an die Umgebung, dessen Verhalten während der Laufzeit und sämtliche Schnittstellen, die es erzeugt. Das **Computational Interface Template** umfaßt die Signatur und das Verhalten einer Schnittstelle, die diese bei der Instantiierung zur Laufzeit annimmt. Die Signatur beschreibt die an der Schnittstelle möglichen Interaktionen und welche Dienste genutzt oder angeboten werden können. Weiter sind auch noch die Anforderungen an die Umgebung in dem Template festgelegt.

Das Referenzmodell legt drei Arten von **Schnittstellen** fest:

- **operation**: Hierbei handelt es sich um Prozedur-ähnliche Interaktionen, die nach dem Client-Server Prinzip funktionieren. Ein Client ruft eine Funktion des Servers auf, der daraufhin eine Antwort zurücksendet. Operationen können synchron oder asynchron sowie bestätigt oder unbestätigt sein. Die Antwort einer Operation wird als *termination* bezeichnet, während bestätigte Operationen als *interrogations* und unbestätigte Operationen als *announcements* bezeichnet werden. Tritt ein Objekt an der Schnittstelle als Dienstnutzer auf, so nimmt es die Rolle eines *Client* an, während es als Diensteanbieter die Rolle eines *Servers* annimmt.

- **flow**: Ein *flow* ist ein Datenstrom zwischen einem Erzeuger (*Producer*) und einem Verbraucher (*Consumer*). Datenströme kommen z.B. bei Video- oder Audioübertragungen innerhalb von Multimedia-Anwendungen vor.

- **signal**: Ein *signal* ist ein Ereignis, das von einem Objekt (*Initiator*) zu einem bestimmten Zeitpunkt generiert und an ein anderes Objekt (*Responder*) geschickt wird. Hierbei handelt es sich um eine atomare, also ununterbrechbare Kommunikation. Beispielsweise kann das eine Unterbrechungsaufforderung (*Interrupt*) sein, die von einem sie initiierendes Objekt an ein antwortendes Objekt geschickt wird.

Weiter definiert das Referenzmodell **Bindungen** (*bindings*), die als eine Art Kommunikationsverbindung zwischen den Schnittstellen zweier Computational Objects hergestellt werden. Es ist wichtig, daß die beiden an eine Bindung angeschlossenen Schnittstellen vom gleichen Typ sind. Bindungen zwischen genau zwei Schnittstellen werden als *primitive* Bindung bezeichnet und können *implizit* oder *explizit* sein. Implizite Bindungen werden nur zwischen **Operation Interfaces** aufgebaut und entsprechen einem entfernten Methodenaufruf. Bei expliziten Bindungen muß die Verbindung zwischen den kommunizierenden Objekten explizit aufgebaut werden. Wird eine Bindung zwischen mehreren Schnittstellen aufgebaut, so handelt es sich um eine *komplexe* Bindung (*compound binding*), die über ein **Binding Object** realisiert wird. Das Binding Object koppelt die Schnittstellen mehrerer Objekte durch primitive Bindungen aneinander, die über eine Kontrollschnittstelle hinzugefügt, überwacht oder gelöscht werden können (siehe Abbildung 3.1).

3.2.2 Engineering Viewpoint

Der Engineering Viewpoint stellt Objekte zur Verfügung, die die Realisierung der Infrastruktur einer verteilten Anwendung erlauben und somit die Kommunikation zwischen den Objekten der verteilten Anwendung ermöglichen. Ein Rechner wird im Rahmen dieses Viewpoints als **Node** bezeichnet, dessen Speicher-, Rechen- und Kommunikationsressourcen von einem **Nucleus** verwaltet werden. Somit kann ein Nucleus dem Betriebssystem gleichgesetzt werden. Weiter werden **Capsules** definiert, die Prozesse darstellen sollen und über einen eigenen Adressraum verfügen.

Die Computational Objects werden zur Laufzeit auf **Basic Engineering Objects** *(BEOs)* abgebildet und stellen Module von ausführbaren Programmen dar. Mehrere solche Module oder Basic Engineering Objects werden zu einem **Cluster** zusammengefaßt und bilden somit ein ausführbares Programm. Ein Cluster stellt den virtuellen Speicher dar, der Objekte der

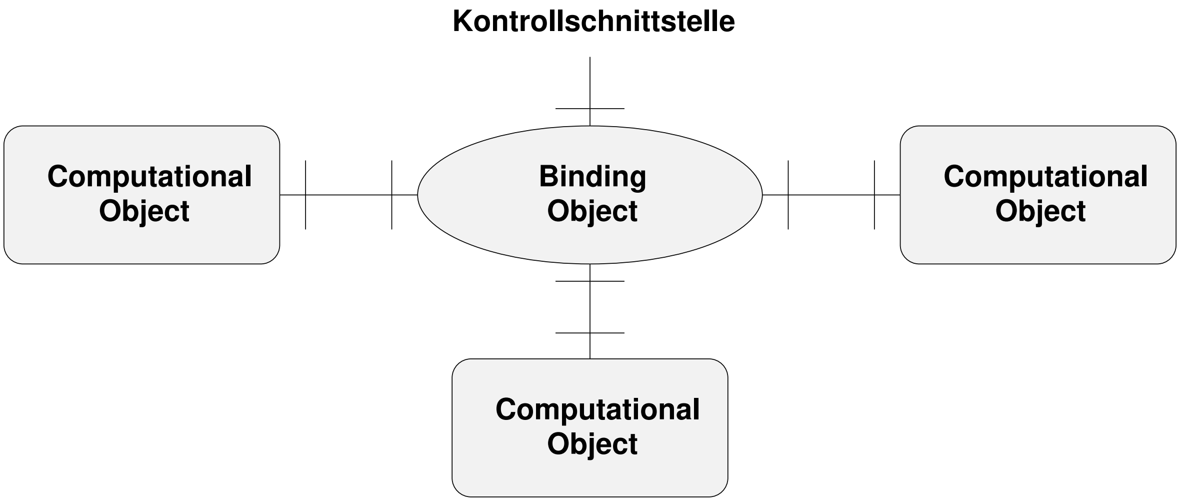

Abbildung 3.1: Binding Object

verteilten Anwendung enthält.

Die Bindungen aus dem Computational Viewpoint werden im Engineering Viewpoint auf **Channels** abgebildet und jedes Computational Interface auf ein **Engineering Interface**. Channels werden nur zwischen Objekten aus Capsules von unterschiedlichen Nodes aufgebaut. Die Hierarchie der festgelegten Objekte des Engineering Viewpoints soll in folgender Abbildung 3.2 verdeutlicht werden.

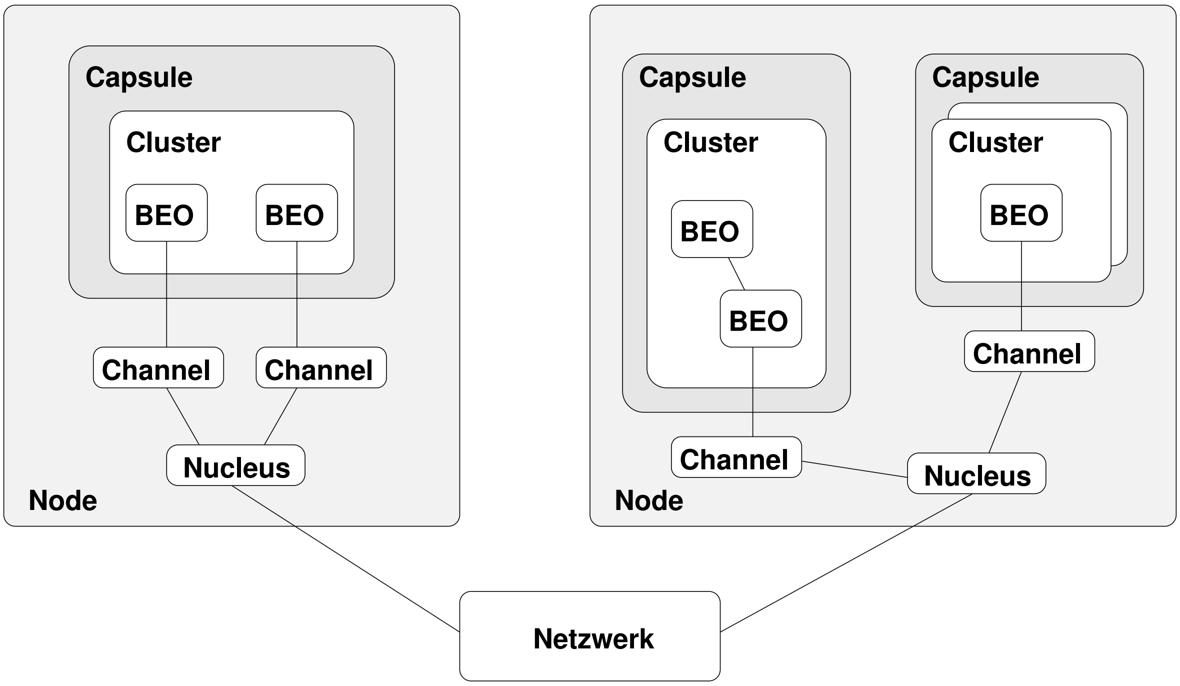

Abbildung 3.2: Objekte des Engineering Viewpoints

Um verschiedene Transparenzen in verteilten Umgebungen berücksichtigen zu können, werden die Channels im Engineering Viewpoint genauer spezifiziert und in weitere Objekte unterteilt, je nachdem, was sie für Aufgaben haben. Ein Channel besteht aus **Stubs**, **Binders**, **Protocol Objects** und einem **Interceptor**. **Stubs** interagieren direkt mit den Basic Engineering Objects und übernehmen das Umwandeln von Datenströmen in Bytefolgen, um sie auf Kommunikationswegen übertragen zu können. Sie nehmen also, im Gegensatz zu Binders und Protocol Objects, Rücksicht auf die interne Struktur der übertragenen Daten. **Binders** sichern die End-to-End-Verbindung des Kanals zwischen zwei oder mehreren Objekten. **Protocol Objects** sichern das Transportsystem des Kanals und implementieren das benötigte Protokoll. Sie können jederzeit ersetzt werden, abhängig vom verwendeten Protokoll. Befinden sich die Basic Engineering Objects in unterschiedlichen Domänen, so wird ein **Interceptor** benötigt, der Format- und Protokollkonversionen oder Sicherheits- und Zugangskontrollen der zwischen den Protokollobjekten übertragenen Daten durchführt. Jede Komponente eines Channels kann durch Kontrollschnittstellen gesteuert werden.

Durch Unterteilen eines Channels in verschiedene Komponenten können unterschiedliche Transparenzen, wie z.B. Orts-, Migrations-, Replikations- oder Zugangstransparenz, realisiert werden. Anwendungsentwickler können die Transparenz-Ebene auswählen, die sie benötigen und somit Verteilungsaspekte der Anwendungskomponenten nach Bedarf ein- und ausblenden. Der Aufbau eines Channels ist in Abbildung 3.3 dargestellt.

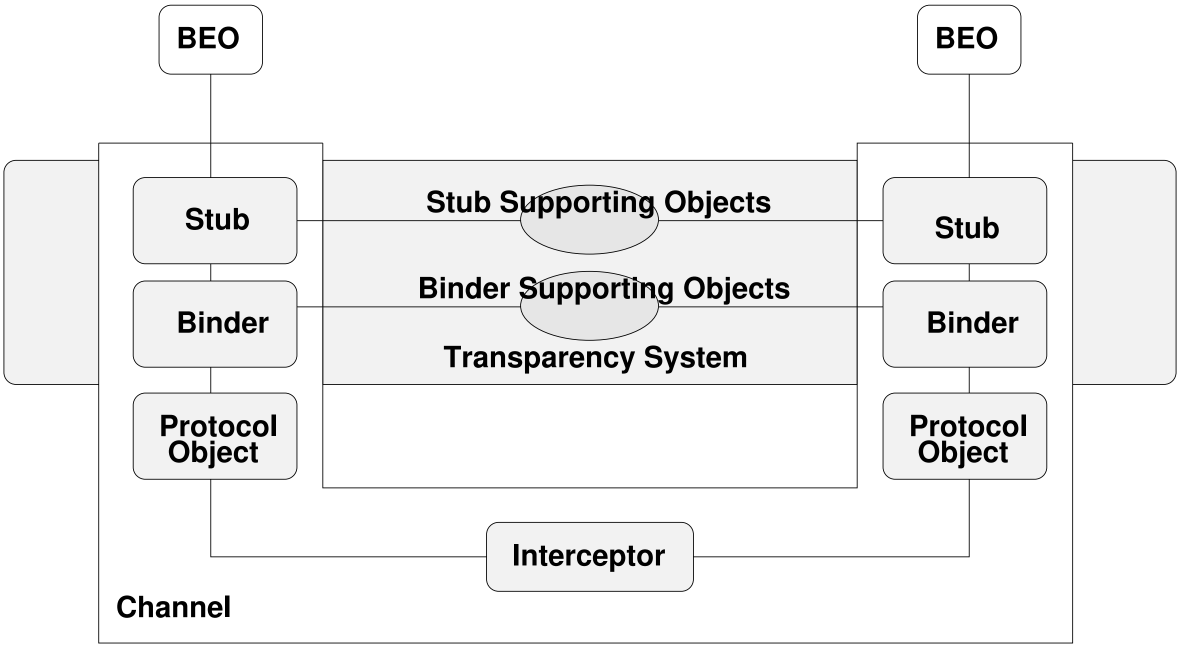

Abbildung 3.3: Spezifizierung eines Channels

3.2.3 Wertung des Modells

Das ODP-Referenzmodell erlaubt dadurch, daß es verschiedene Sichtweisen auf eine verteilte Anwendung anbietet, die Komplexität der Anwendung für Entwickler und Anwender zu vereinfachen. Im Hinblick auf ein integriertes Management ist es durch den objektorientierten Ansatz, mit dem eine verteilte Anwendung betrachtet und realisiert wird, von Anfang an und auch jederzeit später möglich, der Anwendung weitere Komponenten hinzuzufügen und entsprechende Schnittstellen einzubauen, über die auch ein effektives Management der Anwendung möglich ist. Ein weiterer Vorteil des Referenzmodells ist die Tatsache, daß mit Hilfe von Standards das Ziel verfolgt wird, daß verteilte Anwendungen in einer offenen, heterogenen und verteilten Systemumgebung interagieren können. Die Konzepte des Referenzmodells sind unabhängig von spezifischen Merkmalen der verwendeten Managementarchitektur und sind für eine größere Menge von Anwendungen gültig.

3.3 Object Modeling Technique (OMT)

Die *Object Modeling Technique (OMT)* ist eine Software-Engineering-Methodologie, die unter der Leitung von James Rumbaugh entwickelt wurde. Sie unterstützt und berücksichtigt den gesamten Entwicklungsprozeß einer Anwendung. Die einzelnen Schritte werden in fünf Phasen unterteilt:

- **Analyse**: Während dieser Phase werden die Anforderungen an die zu erstellende Anwendung spezifiziert und festgehalten. Hier werden auch die Objekte der Anwendung und deren Relationen im *Objektmodell* definiert; das *dynamische Modell* modelliert den dynamischen Kontrollfluß innerhalb der Anwendung, und das *funktionale Modell* definiert die funktionale Transformation der Daten innerhalb der Anwendung.

- **Systementwurf**: Hier werden die Gesamtarchitektur der Anwendung sowie Grundsatzentscheidungen zur Realisierung und Implementierung festgelegt.

- **Objektentwurf**: Während dieser Phase werden die Modelle, die bei der Analyse definiert wurden, konkretisiert. Sie werden also erweitert und verfeinert, konkrete Objektklassen werden eingefügt und durch Attribute und Funktionen charakterisiert.

- **Implementierung**: Bei der Implementierung werden die Objektklassen der Anwendung in einer Programmiersprache realisiert.

- **Test**: Der anschließende Test aller erzeugten Komponenten soll sicherstellen, daß die Anwendung alle Ziele der Analyse erfüllt und erfolgreich abgeschlossen werden kann.

OMT wird durch zahlreiche CASE-Tools unterstützt, wodurch sie eine weite Verbreitung erlangt hat. Das im Rahmen dieser Diplomarbeit angewendete CASE-Tool heißt *Software through Pictures (StP)* und wird im Abschnitt 3.3.1 näher erläutert.

Nun soll das Objektmodell von OMT näher betrachtet werden, da es die wesentliche Funktionalität bereitstellt, die für die Lösung der Aufgabe in dieser Diplomarbeit benötigt wird. Das Objektmodell stellt in einer Graphik die statische Struktur der Objekte dar, aus denen eine Anwendung besteht. Daraus sind auch die Beziehungen und Abhängigkeiten der Objekte zueinander abzulesen. Objekte sind Instanzen von Klassen, so daß im Diagramm des Objektmodells die Klassen und deren Attribute und Methoden dargestellt werden. Attribute spezifizieren die Eigenschaften eines Objekts, während Methoden Operationen auf die Datenstruktur des Objekts erlauben und somit das Verhalten des Objekts spezifizieren.

Im Objektdiagramm, das als Graph dargestellt wird, werden die Klassen als Knoten und die Relationen zwischen den Klassen als Kanten dargestellt. OMT kennt drei Arten von Relationen: *Assoziation*, *Generalisierung* und *Aggregation*. *Assoziationen* beschreiben Relationen zwischen Objektinstanzen, wobei im Objektdiagramm auch dargestellt werden kann, ob eine Objektinstanz mit einer oder mehreren Instanzen eines anderen Objekts in Relation treten kann. Durch *Generalisierung* wird die Vererbung einer Unterklasse von einer Oberklasse dargestellt. Die *Aggregation* ist eine Spezialform der Assoziation und beschreibt eine "Enthaltensein"-Assoziation. Hier soll aber nicht tiefer auf Funktionen und Merkmale von OMT eingegangen werden. Dafür wird auf die Sekundärliteratur [RBP+91] verwiesen.

3.3.1 Software through Pictures (StP)

Das CASE-Tool *Software through Pictures* (Version 2.4.2) der Firma *Aonix* vereint unter einer Arbeitsoberfläche eine Palette von Werkzeugen, die den gesamten Entwurfsprozeß bei der Entwicklung von objektorientierten Software-Anwendungen unterstützen. Diese Multi-User-Werkzeuge sind in einer Client-Server-Architektur (*StP Core*) eingebunden und arbeiten alle auf einer gemeinsamen Datenbasis (*Repository*). Um OMT-Objektmodelle zu erstellen, muß der entsprechende Diagrammeditor (*OMT Object Model Editor - Version 3.4.2*) aufgerufen werden (siehe Abbildung 3.4).

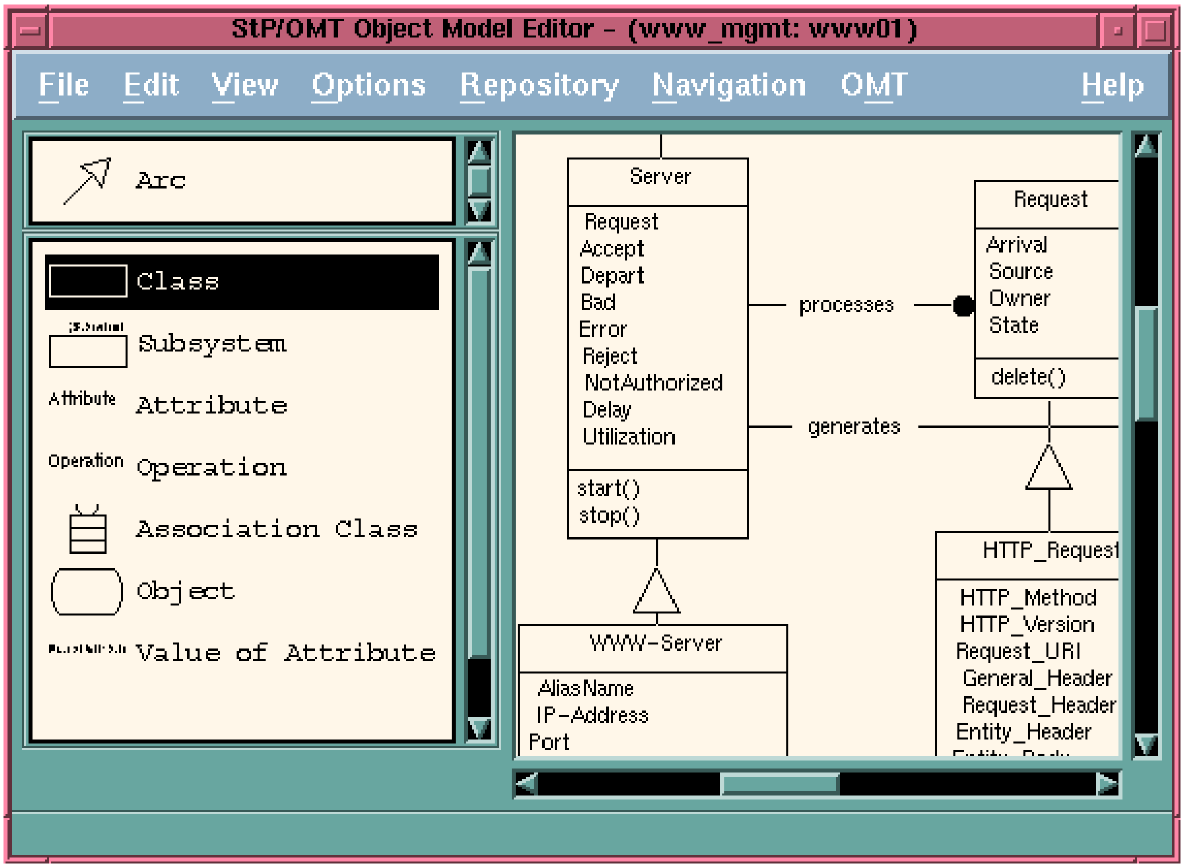

Abbildung 3.4: StP OMT Object Model Editor

Darin können die Symbole für Klassen eingefügt, deren Namen, Attribute und Funktionen eingetragen und anschließend die Beziehungen zwischen den Klassen gezeichnet und benannt werden. Um später jedoch Code für die Klassen erzeugen zu können, sollten allerdings zu jeder Klasse deren Attribute und Funktionen in einer Klassentabelle eingetragen werden, die mit Hilfe eines Klassentabelleneditors erstellt werden kann. Darin können die Typen, Parameter und Rückgabewerte der Attribute und Methoden eingetragen werden. Die Attribute und Funktionen der Klasse werden anschließend in dem Objektmodell aus dem Klassenmodell übernommen. Nach Erstellung des Objektmodells können mittels des CASE-Tools für einzelne oder für alle Klassen eines Diagramms IDL-Objektbeschreibungen erzeugt werden. Für Details über die Benutzung des StP-Tools sei auf die Handbücher [Int96b, Int96c, Int96d, Int96a] verwiesen.

3.4 Common Object Request Broker Architecture (CORBA)

Die *Common Object Request Broker Architecture* ist ein offener Standard für verteilte Objekte und wurde von der *Object Management Group (OMG)* entwickelt. Die OMG ist ein hersteller-unabhängiges Software-Konsortium, das Standards für die Entwicklung und Verteilung von Anwendungen in verteilten heterogenen Umgebungen entwickelt und verbreitet. CORBA beschreibt eigentlich nur die Schnittstellen und Eigenschaften des *Object Request Broker (ORB)*, einer der Schlüsselkomponenten der *Object Management Architecture (OMA)*. Die OMA setzt sich aus einem Objekt- und einem Referenzmodell zusammen. Das Objektmodell definiert, wie verteilte Objekte in einer heterogenen Umgebung beschrieben werden können, während das Referenzmodell die Interaktionen zwischen den Objekten charakterisiert. Die Komponenten des Referenzmodells sind in Abbildung 3.5 dargestellt.

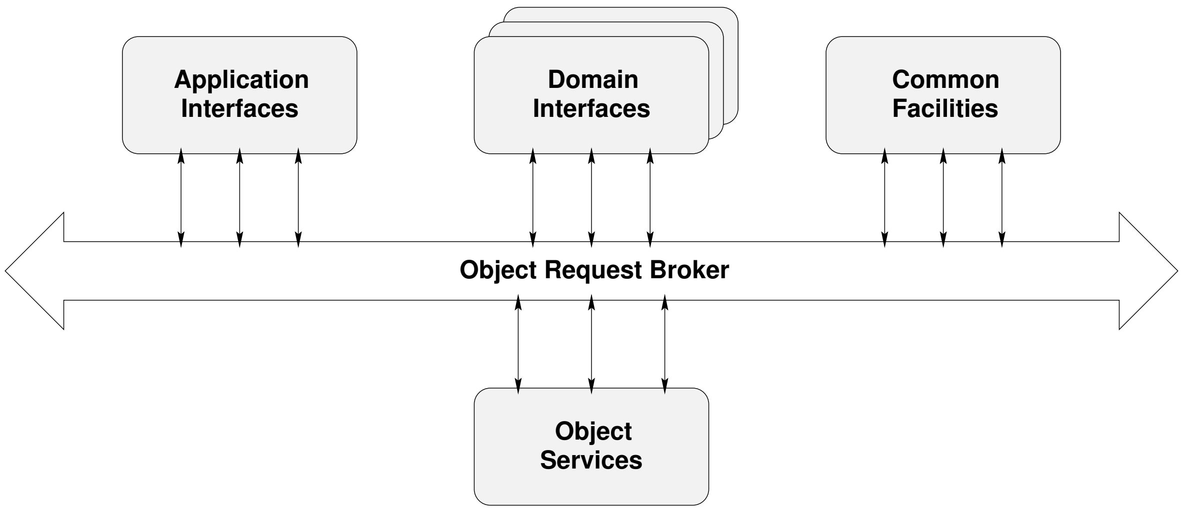

Abbildung 3.5: Object Management Architecture (nach [Vin97])

Der ORB ist die wichtigste Komponente der Architektur und ist hauptsächlich für die Kommunikation der Clients mit den entfernten Objekten verantwortlich. Die Terme “Server” und “Client” beschreiben in CORBA die Rolle der Objekte, die Dienste anderer Objekte nutzen bzw. selbst anderen Objekten zur Verfügung stellen. So kann ein Objekt die Rolle des Servers bezüglich einiger Dienste annehmen, während er die Rolle des Clients für andere Dienste übernimmt. Jeder Methoden-Aufruf von Client-Objekten auf entfernte Server-Objekte wird von dem ORB einschließlich der Parameter an die entsprechenden Server-Objekte weitergeleitet. Die Antwort wird dann umgekehrt vom Server-Objekt an den Client zurückgeschickt. Die Haupteigenschaft des ORB ist es, den Ort des Objekts vor dem Client zu verbergen, dessen Implementierung und Ausführungszustand sowie den Kommunikationsmechanismus, den der ORB zum Versenden der Anfrage und Zurückliefern der Ergebnisse verwendet. Dadurch ist es möglich, daß verteilte Objekte mittels CORBA unabhängig von der zugrundeliegenden Netztechnologie, dem Betriebssystem und der Programmiersprache, in der sie realisiert sind, miteinander agieren können. Für eine nähere und ausführlichere Beschreibung und Erklärung von CORBA und den weiteren OMA-Komponenten sei auf die Sekundärliteratur [Sie96, Vin97, Vog97, Obj95] verwiesen.

3.4.1 Wertung der CORBA-Architektur

Im Rahmen dieser Diplomarbeit wird die CORBA-Architektur zur Realisierung von verteilten Managementanwendungen deswegen ausgewählt, da sie speziell in heterogenen, verteilten Umgebungen einige Vorteile gegenüber anderen Managementarchitekturen bietet:

- CORBA wurde speziell für die Kommunikation und Kooperation von Objekten in offener, heterogener Umgebung konzipiert. Sie beschreibt eine standardisierte Architektur zur Bereitstellung der Infrastruktur für verteilte, objektorientierte Anwendungen. Dadurch können Vorteile der Objektorientierung für verteilte Anwendungen genutzt werden.

- Die Architektur wurde prinzipiell für alle Arten von verteilten Anwendungen definiert. Sie kann also sowohl für verteilte *Management*anwendungen als auch für *allgemeine* verteilte Anwendungen genutzt werden. Es ist somit kein zusätzliches Managementprotokoll (wie z.B. SNMP) notwendig, um die Kommunikation zwischen Managementanwendungen zu realisieren.

- Durch die Beschreibung der Schnittstellen in IDL ermöglicht CORBA eine Unabhängigkeit von bestimmten Programmiersprachen. Objekte, die in unterschiedlichen Programmiersprachen kodiert sind, können unabhängig davon über den ORB miteinander kommunizieren.

- Die Bedeutung von CORBA für das integrierte Management steigt kontinuierlich. Dank der Standardisierung von CORBA setzen mehr und mehr Produkte im Bereich des Managements auf CORBA als Managementarchitektur.

- Zusätzlich unterstützen Tools die Definition der Aufrufschnittstellen von Objekten in der CORBA-Notation *Interface Definition Language (IDL)* und deren Bindung an bestimmte Implementierungssprachen (z.B. C++, Java).

Kapitel 4

Managementinformation bezüglich des Dienstes WWW

Das Internet ist ein Informations-Netzwerk. Die Information wird von Servern bereitgestellt und von Clients abgerufen. In manchen Fällen wird aus Sicherheits- oder Performancegründen der direkte Weg zwischen Servern und Clients von Proxy-Servern unterbrochen. Da Proxies Server sind, die mit spezifischen Funktionalitäten ausgestattet sind, kann man sagen, daß die verteilte Anwendung WWW in die Komponenten Server und Clients unterteilt werden kann.

Das Management der verteilten Anwendung WWW betrifft also das Steuern und Überwachen der Server- und Client-Komponenten während des Betriebs. Dazu zählen unter anderem auch deren Installation und Konfiguration. Da die im Internet ausgetauschte Information sehr vielfältiger Art ist (zu sehen an den unterschiedlichen Formaten der übertragenen Dateien), muß im Rahmen des Anwendungsmanagements zusätzlich zu den Komponenten Server und Client auch die Verwaltung und Organisation der im Web abrufbaren und angewendeten Dokumente berücksichtigt werden.

Zusammenfassend lassen sich die Tätigkeiten im Rahmen des Managements des Dienstes WWW in folgende Bereiche einordnen: **Installation**, **Konfiguration**, **Leistungsmanagement**, **Sicherheitsmanagement**, **Abrechnungsmanagement**, **Fehlermanagement** und **Link-Verwaltung**.

Für Server und Proxies ist jeder dieser Bereiche von großer Bedeutung, da die Verfügbarkeit des WWW-Dienstes hauptsächlich von deren Funktionsfähigkeit und Performance abhängt. Darum ist neben einer stetigen Wartung eine richtige Installation und Konfiguration dieser Komponenten von Anfang an sehr wichtig. Bei Clients hingegen liegen die Schwerpunkte besonders in den Bereichen der Installation, Konfiguration sowie im Fehlermanagement, gegebenenfalls auch im Bereich des Sicherheitsmanagements, falls Browser hier gewisse Einstellungen zur Absicherung gegen Nutzung Dritter vorsehen.

Für jeden Bereich können pro Komponente spezifische Szenarien und Tätigkeitsabläufe ausgearbeitet werden. Dies ist sehr wichtig für die Ziele des integrierten Managements, da möglichst viele Komponenten mit nur einer Managementanwendung überwacht und gesteuert werden sollten. Die für Managementanwendungen notwendige Information über Ressourcen kristallisiert sich durch die Untersuchung der Anforderungen heraus, die an die Anwendung gestellt werden. Das heißt, daß durch eine Top-Down-Analyse der Anforderungen, die zu Managementzwecken an eine Anwendung gestellt werden, sowohl die Informationen über die Komponenten als auch die Funktionen zum Steuern und Überwachen dieser Komponenten hervorgehoben werden.

In den nächsten Unterkapiteln sind die Szenarien und die daraus zu gewinnende Managementinformation anhand einer Bottom-up-Analyse der Komponenten Server, Proxies und Clients nacheinander aufgezählt. Für jede Komponente sind die Anforderungen des Anwendungsmangements nach den Hauptkriterien der Installation und Konfiguration, des Leistungs-, Sicherheits-, Abrechnungs- und Fehlermanagements sowie der Link-Verwaltung aufgeteilt. Für Clients werden nur die Installation und Konfiguration verstärkt betrachtet und damit zusammenhängende Fälle des Fehlermanagements.

4.1 Server

Wie bei jeder Software-Komponente muß bei Servern von Anfang an auf eine korrekte Installation und Konfiguration der Software geachtet werden, um im Web Informationen zur Verfügung stellen zu können. Dabei läßt sich die Konfiguration eines Web-Servers in zwei Hauptbereiche unterteilen: die eigentliche Server-Konfiguration und die Ressourcen-Konfiguration. Die eigentliche Server-Konfiguration umfaßt Parameter wie z.B. die Server- und Document Root, Einrichtung von virtuellen Servern oder Angaben zum Betrieb des Servers. Die Ressourcen-Konfiguration betrifft dagegen Angaben zu den einzelnen Dateitypen, die auf dem Server lagern, das Aussehen von automatisch erzeugten Verzeichnisindizes oder die Einrichtung von Benutzerverzeichnissen. Als Ressourcen werden dabei die gesamten Dateien und Verzeichnisse betrachtet, die innerhalb der Server- und der Document Root zu finden sind, wie z.B. die öffentlich zugänglichen Dokumente oder CGI-Programme, auf die jeder Benutzer zugreifen kann.

Weiter ist auf eine ansprechende Performance des Servers zu achten, da hierdurch längere Wartezeiten für Clients beim Zugriff auf Informationen vermieden werden können. In diesem Bereichs fallen einerseits die Analysen der Log-Dateien von Servern und andererseits die Überprüfung und Steuerung der Ressourcen-Auslastung. Dabei können Langzeitstatistiken über die Auslastung des Servers und das Abrufen von Dokumenten erstellt werden, um zu überprüfen, ob der Server entsprechend ausgelastet oder mit dem Bearbeiten der Anfragen überfordert ist. Hier setzt das Steuern und Überwachen der Prozesse und Ressourcen an.

Sehr wichtig sind auch angemessene Sicherheitseinrichtungen, um den Server gegen Angriffe aus dem Internet zu schützen. Hierfür bieten Web-Server unterschiedliche Mechanismen an. Es ist wichtig, sowohl die physikalischen Ressourcen (darunter fallen das System, die Dateien und die Verzeichnisse) als auch die Dokumente des Servers zu schützen, auf die mit Hilfe von Browsern zugegriffen werden kann.

Bei Fehlern während des Betriebs sind entsprechende Maßnahmen zu unternehmen, um diese zu lokalisieren und zu beheben. Dafür sind im Rahmen des Fehlermanagements neben überwachenden auch steuernde Tätigkeiten notwendig.

Eine weitere wichtige Aufgabe im Rahmen des Managements von Web-Servern betrifft die Abrechnung der an die Nutzer des Servers geleisteten Dienste. Damit können unter anderem der Austausch von Informationen oder das Datenvolumen gemeint sein, das über den Server abgewickelt wird. Und schließlich ist die Organisation und Verwaltung der Daten, die auf dem Server gespeichert sind und im Web zur Verfügung gestellt werden, nicht zu vergessen.

Um eine Gesamtübersicht über die Aufgaben im Rahmen des Managements eines Web-Servers zu erhalten, soll folgende Abbildung alle Tätigkeiten entsprechend zusammenfassen:

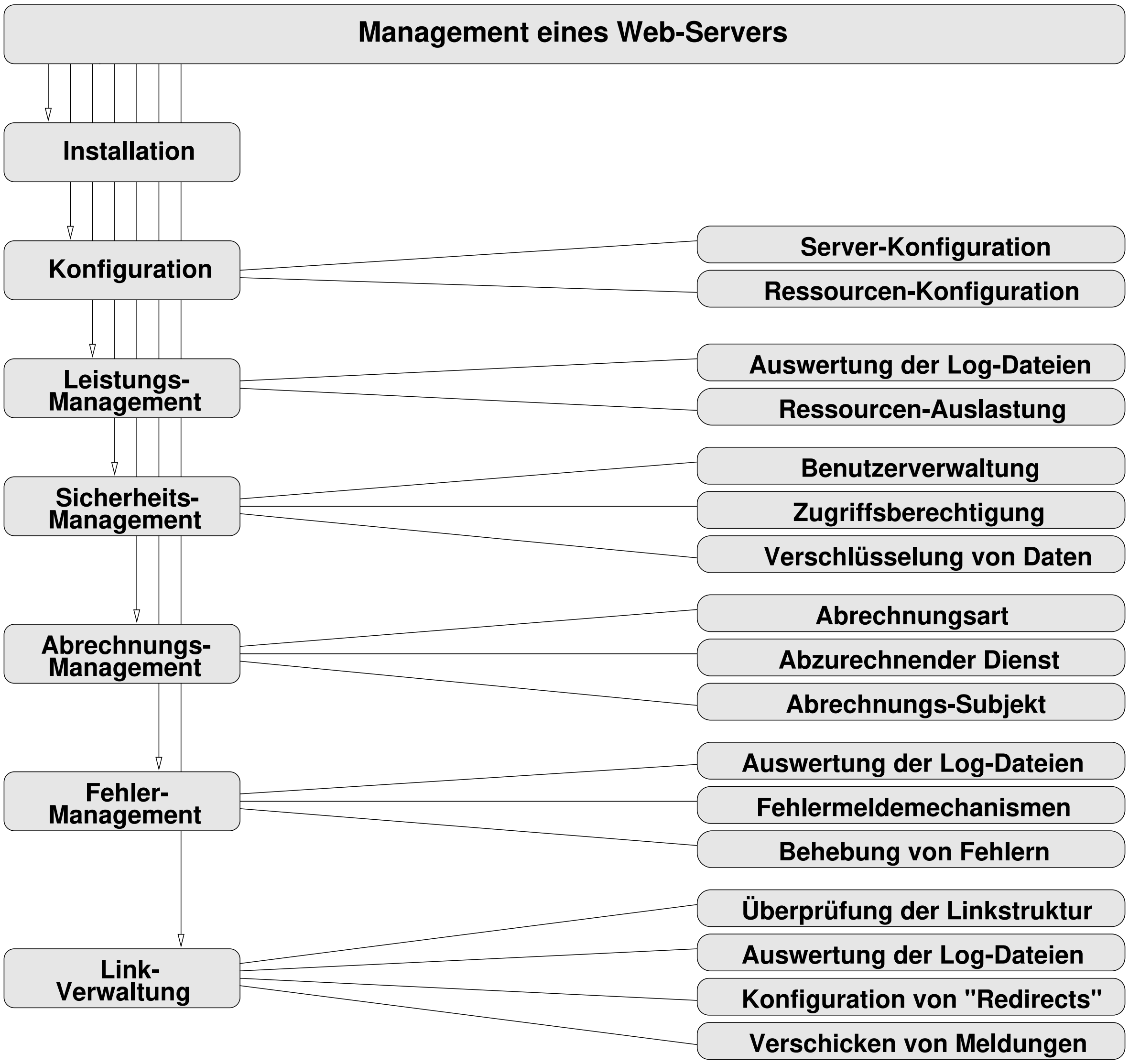

Abbildung 4.1: Gliederung der Management-Bereiche

Im Rahmen dieser Diplomarbeit werden die Merkmale und Eigenschaften eines Web-Servers beispielhaft anhand des Apache Web-Servers dargestellt (siehe dazu [Eil97]) und gegebenenfalls mit anderen kommerziellen Web-Servern verglichen.

4.1.1 Installation

Die richtige Installation eines Web-Servers ist eine wichtige Voraussetzung, um während des späteren Betriebs keine Probleme zu bekommen. Dabei sind einige Punkte zu beachten. Bei der Installation der Server-Software wird eine Anzahl an Unterverzeichnissen angelegt, die alle zum Betrieb notwendigen Dateien des Servers enthalten. Die Anzahl der Unterverzeichnisse kann je nach Software-Hersteller variieren. Im Prinzip gibt es eine Haupt-Binärdatei, den eigentlichen Server (wird auch Server-Dämon genannt) und weitere Dateien, die je nach Bedeutung in den entsprechenden Verzeichnissen geordnet sind. Darunter fallen die Konfigurationsdateien, die den Betrieb des Servers steuern und meistens in einem eigenen Verzeichnis gespeichert sind, um die Übersichtlichkeit zu wahren. Weiter werden Hilfsprogramme mitgeliefert, die ebenfalls in einem eigenen Verzeichnis abgelegt sind. Dazu zählen auch die sogenannten CGI-Skripten, die einige Grundfunktionen eines Web-Servers erfüllen sollen und in einem `cgi-bin`-Verzeichnis gespeichert werden. Weiter gibt es meistens auch ein `log`-Verzeichnis, in dem die Log-Dateien gespeichert werden. Diese Dateien werden nach dem Start des Servers generiert. In ihnen werden Zugriffe auf den Server und Fehlermeldungen mitprotokolliert.

Vor der Installation der Server-Software auf dem System müssen einige Grundvoraussetzungen geschaffen werden. Es sollte sichergestellt sein, daß im Domain Name Service (DNS) ein **Aliasname für den Server** eingetragen ist, der auf das System verweist. Im DNS sind die Paare festgelegt, die einer IP-Adresse einen eindeutigen Hostnamen zuweisen. So können später der Name und die IP-Adresse des Systems geändert werden, ohne alle URLs ändern zu müssen, die auf den Server verweisen. Die einzige Änderung muß im DNS erfolgen.

Folgende Informationen über das System, die für die Installation und Konfiguration der Server-Software notwendig sind, sollten abfragbar sein:

- **Hostname** (z.B. `myserver.mydomain.top-level-domain`)
- **Aliasname** des Systems (z.B. `www.mydomain.top-level-domain`)
- **IP Adresse** des Systems
- **OS**: Betriebssystem
- **Hardware**: Definition der Hardware des Rechners
- **RAM**: verfügbarer interner Speicher
- **Disk**: verfügbarer Plattenspeicher

Diese Informationen müssen dazu benutzt werden, um zu überprüfen, ob die **Anforderungen**, die bezüglich der Installation der Server-Software **an die Server-Hardware** gestellt werden, erfüllt sind. Dabei sind folgende Informationen besonders wichtig:

- **Betriebssystem** (um die entsprechende Version der Server-Software auszuwählen)
- **freier Diskspace** (der für die Installation der Server-Software und später während der Laufzeit als Platz für die log-Files und Dokumente benötigt wird; diese Angabe läßt sich systemspezifisch für die entsprechenden Ressourcen bestimmen)

Um dem Server-Dämon den Zugriff auf Rechner und Netzressourcen zu ermöglichen, ist das **Anlegen eines User-Accounts** (Name, Gruppe und Paßwort) für den Server notwendig.

Damit der Server Requests empfangen kann, benötigt er einen bestimmten Port des Rechners, über den er Anforderungen entgegennehmen kann. Falls der Standardport für HTTP (Port 80) belegt ist, muß ein anderer dafür definiert werden. Deswegen ist ein systemspezifisches **Abfragen über die Belegung der Ports** des Systems noch vor der Installation der Server-Software sehr wichtig.

Anschließend kann der Source-Code der Server-Software **ins richtige Verzeichnis (Server Root) kopiert** werden. In das Server Root-Verzeichnis werden später die gesamten Binär-Dateien der Server-Software installiert. Das Document Root-Verzeichnis enthält die Dokumente, die von Clients im Internet von dem Server abgerufen werden können. Noch vor dem Start sollte eine Grundkonfigurierung des Servers erfolgen (siehe dazu Abschnitt 4.1.2). Danach kann der **Server gestartet** werden. Will man die Grundkonfiguration des Servers allerdings erst nach dem Start vornehmen, so muß das Starten des Servers unter Angabe von unterschiedlichen Eingabeparametern möglich sein. So muß man die Default-Einstellungen, die vom Hersteller für den ersten Start des Web-Servers eingerichtet werden, nicht übernehmen.

Folgende Eingabeparameter sind dabei notwendig:

- der **Port**, über den Anforderungen angenommen werden sollen
- die **Konfigurationsdatei**, die für den ersten Start gelesen werden soll
- die **AccessLog-Datei**, in der die Zugriffsrechte auf die Dokumente des Servers definiert sind

Bei Problemen oder auch nur, wenn man den Server anders konfigurieren möchte, kann man ihn **stoppen**. Dafür gibt es zwei Möglichkeiten. Der Server kann komplett angehalten werden, so daß er keine Dienste mehr anbieten kann. Danach ist ein erneutes Starten des Servers nötig. Er kann aber auch während des Laufs "unterbrochen" werden, wobei der Server dabei nur seine Konfigurationsdateien erneut lädt und neue Log-Dateien öffnet, aber weiterhin Requests bearbeitet. Dieser Vorgang heißt "**Restarten**" des Web-Servers und wird in dieser Diplomarbeit auch so übernommen.

Um den Server zu stoppen, muß der Server-Prozeß angehalten werden. Dafür muß seine Prozeß-ID bekannt sein. Diese anwendungsspezifische Information wird beim Apache-Server in einer bestimmten Datei gespeichert, die unter `logs/httpd.pid` als **PidFile** bekannt ist (siehe dazu Abschnitt 4.1.2). Darin wird die ID des Vater-Prozesses gespeichert, der während des Betriebs weitere Server-Prozesse aufruft. Dieser Prozeß kann mit drei unterschiedlichen Optionen gestoppt werden: `TERM, HUP` und `USR1`, wobei das Senden eines `TERM`-Signals den Server stoppt, während die beiden anderen Signale nur einen Restart des Servers zur Folge haben. Jede Option ruft ein unterschiedliches Verhalten des Server-Vater-Prozesses auf:

TERM: Dieses Signal hat ein sofortiges Stoppen des Vater-Prozesses zur Folge, der aber davor erst seine gesamten Söhne anhält. In dieser Zeit und danach werden keine Requests mehr angenommen oder bearbeitet.

HUP: Nach Erhalt dieses Signals hält der Server-Vater-Prozeß sämtliche Sohn-Prozesse sofort an, auch wenn sie gerade einen Request bearbeiten. Danach liest er seine Konfigurationsdateien neu, öffnet neue Log-Dateien und erzeugt neue Söhne, die dann wieder Requests annehmen und bearbeiten.

USR1: Dieses Signal bewirkt, daß der Vater-Prozeß seine Söhne erst dann anhält, wenn sie den gerade in Bearbeitung befindlichen Request zu Ende führen. Falls sie gerade keinen Request bearbeiten, dann werden sie sofort angehalten. Anschließend liest der Vater-Prozeß seine Konfigurationsdateien erneut und erzeugt neue Log-Dateien und neue Söhne, die sofort neue Requests annehmen und bearbeiten.

4.1.2 Konfiguration

Nach der Installation der Server-Software muß der Server an die eigene Umgebung im System angepaßt werden. Dafür werden die Konfigurationsdateien des Servers bearbeitet. Eine andere Möglichkeit, die einige Server für die Konfiguration zur Verfügung stellen, ist die Konfiguration des Servers über eine Admin-Oberfläche, wie das z.B. beim Netscape Enterprise Server der Fall ist.

Vor dem ersten Start der Server-Software muß eine Grundkonfiguration eingestellt werden. Eine weitere Möglichkeit besteht darin, die Default-Einstellungen aus den Konfigurationsdateien, die mit der Server-Software mitgeliefert werden, zu übernehmen und weitere Einstellungen erst später vorzunehmen.

Im Rahmen der Konfiguration werden zwei Arten von Einstellungen unterschieden:

1. **Server-Konfiguration**: globale Einstellungen, die den Server allgemein betreffen
2. **Ressourcen-Konfiguration**: Konfigurationseinstellungen, die den Inhalt und die Ressourcen des Web-Servers betreffen

Folgende Abbildung soll die Aufteilung der Bereiche innerhalb der Konfiguration eines Web-Servers verdeutlichen:

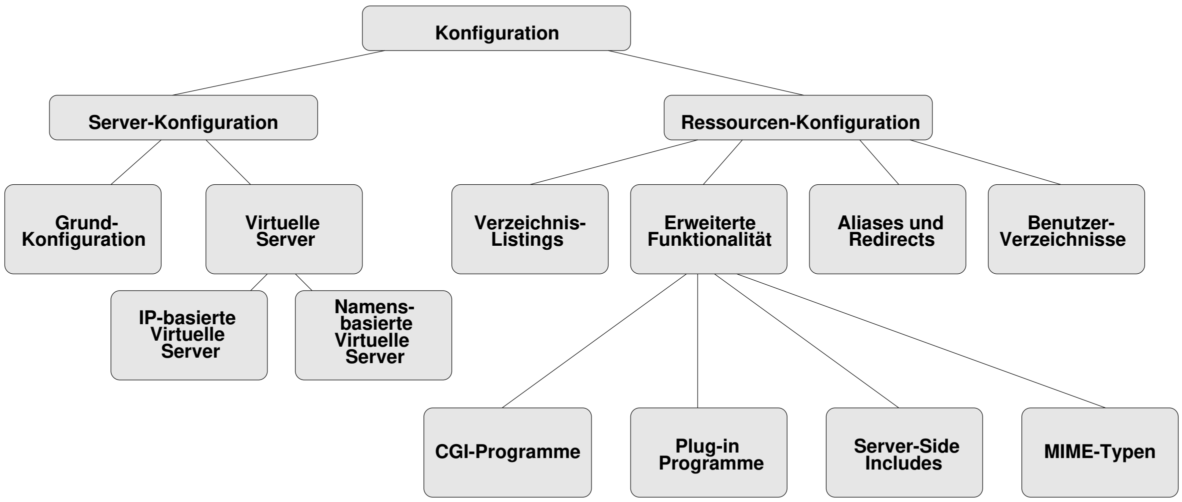

Abbildung 4.2: Gliederung der Konfiguration eines Web-Servers

Die entsprechenden Attribute werden normalerweise in zwei unterschiedlichen Konfigurationsdateien gespeichert, um die Übersichtlichkeit zu bewahren. Beim Apache-Server handelt es sich im ersten Fall um die `httpd.conf`-Datei, während Einstellungen über den Inhalt des Servers in einer `srm.conf`-Datei zu finden sind. Beim Netscape Enterprise Server heißen die Dateien entsprechend `magnus.conf` bzw. `obj.conf`. Während frühere Versionen von Web-Servern noch strikt verlangten, daß die entsprechenden Anweisungen fein säuberlich auf diese zwei Dateien verteilt werden, verfolgen die neuen Web-Server das Ziel, jede Art von Anweisung in beiden Konfigurationsdateien verwenden zu können.

Server-Konfiguration

Die globalen, statischen Einstellungen bestimmen die allgemeine Server-Konfiguration und können in zwei Hauptgruppen unterteilt werden. Die **Grundkonfiguration** eines Servers umfaßt Anweisungen, die unbedingt vorgenommen werden sollten, um den Server an das eigene System anzupassen. Weiter können diese Anweisungen für die Einrichtung von **virtuellen Servern** benutzt werden. Die dazu gehörenden Konfigurationsanweisungen werden in einer **Haupt-Konfigurationsdatei** des Servers angegeben.

Grundkonfiguration

Im Rahmen der Grundkonfiguration werden Anweisungen benutzt, die den Betrieb des Servers regeln sollen. Allgemein sind das Einstellungen der folgenden Parameter:

- **Server Root**: Hier wird der Verzeichnis-Pfad angegeben, in dem die gesamten Server-Binaries gespeichert sind (z.B. `usr/local/etc/httpd`). Jede Angabe eines relativen Dateipfades in weiteren Konfigurationsanweisungen bezieht sich auf dieses Haupt-Verzeichnis.
- **Document Root**: Hier ist der Verzeichnis-Pfad gemeint, in dem HTML-Dokumente und weitere Nutzdateien, auf die der Server zugreifen kann, gespeichert sind. In URLs, mit denen man auf den Dokumenten dieses Servers zugreifen kann, werden alle Unterverzeichnisse und Dateien relativ zu dieser Document Root angegeben.
- **Server Directory**: Verzeichnis, in dem der Server-Dämon gespeichert ist.
- **Servername** (z.B. `www.mydomain.top-level-domain`): Das ist der DNS-Eintrag, der auf das Server-System verweist. So können später der Systemname und die IP-Adresse des Systems geändert werden, ohne alle URLs ändern zu müssen, die auf den Server verweisen. Jeder Server hat einen einzigen eindeutigen Servernamen.
- **Servertyp**: Der Server-Dämon kann unter UNIX als *standalone* oder unter *inetd* laufen. Als *standalone* wartet der Server-Dämon nach dem Start auf Requests und erzeugt eine Kopie von sich selbst für die Beantwortung von einzelnen Requests. Unter *inetd* (dem Internet Super-Server für UNIX) wird jedesmal eine Kopie des Server-Dämons bei jedem Request neu gestartet, wobei hier aber der *inetd*-Server nach Requests horcht und anschließend die Arbeit an den Server-Prozeß abgibt. In einer inetd-Konfigurationsdatei werden alle Netzwerkdienste eingetragen, die unter *inetd* gestartet werden sollen. In diesem Fall muß der WWW-Dienst hinzugefügt werden. Diese Arbeitsweise eignet sich jedoch nur für kleinere Web-Server, die nicht oft Requests entgegennehmen müssen. Bei einem *httpd*-Dienst, wo jede Sekunde bis zu mehrere Dutzend Anfragen ankommen,

würde es bei jeder Anfrage an den Web-Server zu großen Performance-Einbußen kommen, da *httpd* beim Start alle Konfigurationsdateien neu liest. Deswegen sollte hier die Standardeinstellung *standalone* übernommen werden.

- **Server Port**: Diese Angabe spezifiziert den TCP-Port, auf dem der Server nach HTTP-Requests horcht. Der Standardwert für HTTP ist der Port 80. Falls man hier Werte unter 1024 angibt, muß der Server als `root` gestartet werden. Nachdem sich der Server beim Start an diesen Port bindet, wechselt er auf den User (siehe **Server UID**), unter dem er laufen soll. Für jeden anderen Wert außer 80 muß in dem URL, der auf den Server zugreift, der Port spezifiziert werden (z.B. `http://www.company.com:8000/`). Ein Server kann nur eine gültige Eintragung für diesen Wert haben. Außerdem können nicht mehrere Server über einen gleichen Port Requests entgegennehmen, außer es handelt sich um virtuelle Server, die auf unterschiedliche IP-Adressen antworten.

- **Server UID**: Userkennung, die der Server während des Betriebs benutzt

- **Server GID**: Gruppenkennung, die der Server während des Betriebs benutzt

- **PidFile**: Pfadangabe einer Datei, in der die Prozeß-ID des Haupt-Server-Prozesses gespeichert ist. Diese Angabe wird für das Stoppen und das Restarten des Servers benötigt.

- **Server_Admin**: Email-Adresse der Kontaktperson, die man bei Problemen im Zusammenhang mit dem Web-Server benachrichtigen kann

Virtuelle Server

Wenn ein einziger Server Requests unter mehr als einem Servernamen bearbeiten kann, dann spricht man davon, daß er virtuelle Server unterstüzt. Dabei unterscheidet man zwischen **IP-basierten virtuellen Servern** und **Namens-basierten virtuellen Servern**. Virtuelle Server sind z.B. für Internet Provider von großer Bedeutung, da sie auf einem System mehrere Server realisieren und somit ihren Kunden deren eigene Domain als Web-Adresse anbieten können, ohne für jeden Kunden eigene Ressourcen zur Verfügung stellen zu müssen.

Eine weniger elegante Lösung wäre die Zuweisung eines Unterverzeichnisses aus der Document Root an jeden Kunden, wobei dann deren Web-Adresse auf den Provider schließen läßt. Wenn ein Provider z.B. über einen Server unter dem URL `http://www.provider.com` verfügt und für seine drei Kunden orgA, orgB und orgC Web-Sites anbieten will, so sind sie in diesem Fall über folgende Adressen zu erreichen:

- `http://www.provider.com/orgA/`
- `http://www.provider.com/orgB/`
- `http://www.provider.com/orgC/`

Um solche Situationen zu vermeiden, wurden die virtuellen Server entwickelt.

IP-basierte Virtuelle Server

Für diese Art von virtuellen Servern ist die **Zuweisung einer IP-Adresse an jeden virtuellen Server**, der auf dem System realisiert wird, vorausgesetzt. Nur so kann der Server anhand dieser IP-Adresse erkennen, an welchen virtuellen Server auf dem System ein ankommender Request gerichtet ist und in welchem Verzeichnis er nach der angeforderten Datei suchen muß. Dabei ist zu beachten, daß für jeden virtuellen Server eine **eigene Document Root** und auch **eigene Log-Dateien** einzurichten sind, um sich die Verwaltungsarbeit zu vereinfachen.

Weiter können die meisten der oben unter der für Server geltenden “Grundkonfiguration” angegebenen Attribute auch in Bezug auf virtuelle Server angewandt werden, falls sie sich von der Konfiguration des Hauptservers unterscheiden. Jeder virtuelle Server erbt die Grundkonfiguration des Hauptservers. Das einzige Attribut, das in diesem Fall hinzukommen muß, ist die `IP-Adresse`, die einem IP-basierten virtuellen Server zugewiesen wird.

Namensbasierte Virtuelle Server

Namensbasierte virtuelle Server haben sich aus einer zusätzlichen Funktionalität der neueren Version 1.1 des HTTP-Protokolls gegenüber der Version 1.0 entwickelt. In der neuen Version schickt ein Client in dem HTTP-Header seiner Anfrage ein Feld mit dem *Host*-Namen mit und macht somit kenntlich, auf welchen Server er zugreifen will.

In der Version 1.0 des HTTP-Protokolls wurde im Header der Anfrage nur der Pfad der angeforderten Datei relativ zu der Document Root des Servers, auf dem sich die Datei befindet, geschickt. Anhand der IP-Adresse, an die der Server als einziger gebunden ist, wird er lokalisiert und nimmt die Anfrage entgegen. Seit der Version 1.1 des HTTP-Protokolls und der Funktionalität des *Host*-Headers ist die Notwendigkeit der Zuweisung einer IP-Adresse an jeden virtuellen Server auf dem System aufgehoben. Der Hauptserver, der an die IP-Adresse gebunden wird, etscheidet anhand des Host-Namen, an welchen virtuellen Server die Anfrage gerichtet ist (siehe Abbildung 4.3).

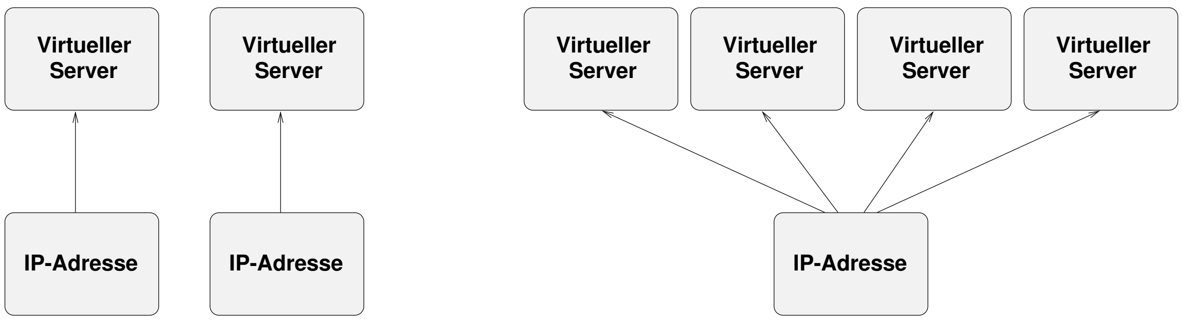

Abbildung 4.3: Unterschied zwischen IP-basierten und namensbasierten virtuellen Servern

Ein Problem bei der Anwendung von namensbasierten virtuellen Servern entsteht allerdings, falls ein Client die neue HTTP-Version nicht unterstützt und somit diesen *Host*-Header nicht mitschickt. Dann ist es für den Hauptserver nicht möglich zu unterscheiden, an welchen virtuellen Server der Request gerichtet ist. Er wird nach einem Dokument in seiner eigenen Document Root suchen und dieses zurückschicken, falls es vorhanden ist.

Auch für diese Art von virtuellen Servern können die meisten der oben unter der "Grundkonfiguration" angegebenen Attribute verwendet werden. Ein weiteres kommt hier hinzu, falls man einem namensbasierten virtuellen Server mehrere Servernamen zuweisen möchte. Dieses Attribut ist **ServerAlias**. Falls eine Anfrage für den Servernamen oder einen der ServerAliases ankommt, so sucht der Hauptserver in der Document Root dieses virtuellen Servers nach der gesuchten Datei und trägt die entsprechenden Log-Daten in die Log-Dateien dieses Servers ein, falls er so konfiguriert ist.

Pro virtuellem Server, ob namensbasiert oder IP-basiert, müssen also mindestens die zwei Attribute **Servername** und **Document Root** in der jeweiligen Konfiguration eingetragen sein. Jedes weitere Attribut ist optional.

Ressourcen-Konfiguration

Die Ressourcen-Konfiguration umfaßt Konfigurationsanweisungen zu den Darstellungsarten, Formaten oder Typen der einzelnen Dateien und Verzeichnisse, die von dem Server für Clients zum Abruf bereit stehen. Weiter können Angaben zu deren Plazierung im Verzeichnisbaum des Servers konfiguriert werden oder zu den entsprechenden Anwendungsprogrammen, die die Darstellung oder Verarbeitung einer Datei übernehmen. Die dazu benötigten Attribute werden in einer **Ressourcen-Konfigurationsdatei** gespeichert. Sie umfassen Angaben zu den folgenden Punkten:

- Erweiterte Funktionalität der HTML-Dokumente
- Verzeichnis-Listings
- Aliases und Redirects
- Benutzerverzeichnisse

Anweisungen, die in der Ressourcen-Konfigurationsdatei eingetragen sind, gelten global für den ganzen Server. Allerdings können auch manche Einstellungen nur lokal für einige Verzeichnisse bestimmt werden. Dafür wird in das gewünschte Verzeichnis ein *Directory Access Control File* plaziert, in dem verzeichnisspezifische Einstellungen festgelegt werden. Dabei ist zu beachten, daß für alle Bestimmungen, die nicht global gelten sollen oder müssen, in der Ressourcen-Konfigurationsdatei einzeln für jede Anweisung eine Erlaubnis zur Überschreibung eingetragen sein muß. In die *Directory Access Control Files* werden auch Zugriffsberechtigungen für das jeweilige Verzeichnis eingetragen, was aber in Kapitel 4.1.4 im Rahmen des Sicherheitsmanagements genauer erläutert ist.

Erweiterte Funktionalität der HTML-Dokumente

Auf den meisten Web-Servern ist es möglich, Programme aufzurufen, um den HTML-Seiten spezifische Funktionalitäten hinzufügen zu können.

Zum Beispiel ist durch das Einbinden von Formularen ein ”Dialog” mit dem Leser einer HTML-Seite möglich. So können von ihm Daten abgefragt werden, statt ihm nur reine Information in Form von Texten und Bildern zu präsentieren. Abhängig von den gelieferten Antwortdaten können mit Hilfe von **CGI-Programmen** dynamische Seiten erstellt und an den Client zurückgeschickt werden.

Diese Skripten können in jeder Programmiersprache geschrieben werden; sie können von ganz einfachen Funktionen der Seitengestaltung mit Hilfe von Templates über Wortsuchfunktionen einer Suchmaschine bis hin zu der Realisierung einer Datenbankanbindung reichen. Allgemein wird zwischen CGI-Programmen und **Hilfsprogrammen** unterschieden, die auf dem Server ausgeführt werden und die Funktionalität des Servers erweitern.

Weiter ist es mit Hilfe des Mechanismus der **Server-Side Includes** möglich, HTML-Seiten dynamisch so zu generieren, daß sie das aktuelle Datum oder andere sich kontinuierlich ändernde Daten enthalten. Zusätzlich kann mit Hilfe von **MIME-Typen** einem Client mitgeteilt werden, welche Art von Datei ihm gerade übertragen wird.

Nachfolgend werden diese vier Mechanismen und die dazugehörigen Konfigurationsanweisungen einzeln erläutert.

CGI-Programme

Das *Common Gateway Interface (CGI)* ist eine Schnittstelle des Servers, über die er Anwendungsprogramme aufrufen und mit ihnen Informationen austauschen kann. Diese Anwendungsprogramme werden meistens CGI-Programme oder -Skripten genannt und können in jeder Programmiersprache implementiert werden. Der Server ruft auf Nachfrage eines Clients ein CGI-Programm auf, übergibt ihm gegebenfalls Daten, falls der Client welche mitgeschickt hat, und setzt die notwendigen Umgebungsvariablen. Ein CGI-Programm erzeugt daraufhin eine Ausgabe, die vom Server an den Client weitergeschickt wird.

Normalerweise können Server so konfiguriert werden, daß entweder ein Verzeichnis als CGI-Verzeichnis gekennzeichnet wird oder einzelne Dateien als CGI-Programme erkannt werden. Beim Apache-Server wird durch Einfügen einer Anweisung **ScriptAlias** in die Ressourcen-Konfigurationsdatei und Angabe eines Verzeichnisnamens ein CGI-Verzeichnis als solches gekennzeichnet. Es können beliebig viele CGI-Verzeichnisse bestimmt werden, wobei dann jede darin enthaltene Datei als solche erkannt und vom Server ausgeführt wird.

Die zweite Möglichkeit besteht darin, einen MIME-Typ oder Handler (serverspezifisch) für CGI-Programme einzuführen, der den Server jede Datei mit der festgelegten Endung als CGI-Programm erkennen und ausführen läßt. Die übliche Datei-Endung, die für CGI-Programme verwendet wird, ist `.cgi`. Der Apache-Server verlangt zusätzlich zu der globalen serverweiten Festlegung des Dateityps `.cgi` als CGI-Programm eine verzeichnisbasierte Erlaubnis für einzelne Verzeichnisse, in denen CGI-Programme ausgeführt werden sollen. Diese Möglichkeit, CGI-Programme anhand von Dateitypen zu definieren, ist für den Fall nützlich, daß man in einem Verzeichnis nicht nur CGI-Programme speichern möchte, sondern auch andere Dateitypen.

Allgemein stellen CGI-Programme hohe Anforderungen an die Sicherheitseinrichtungen eines Servers und können deswegen eine Bedrohung für den Server darstellen. CGI-Programme werden nämlich unter der User- und Group-ID des Servers ausgeführt und haben somit Zugriff auf die gleichen Ressourcen wie der Server. Deswegen ist es wichtig, darauf zu achten, daß ein Server nie unter *root* läuft, um einem CGI-Programm dadurch nicht Zugriff auf das gesamte System zu gewähren. Deswegen wurde in der neuesten Version (Version 1.2) des Apache-Servers eine Funktionalität eingebaut, mit deren Hilfe man den Server so konfigurieren kann, daß er CGI-Programme auch unter der User-ID eines Benutzers ausführen kann, dem das CGI-Programm gehört. Diese Funktionalität (`suEXEC` genannt) wird aber im Rahmen der Diplomarbeit nicht weiter erläutert. Dafür wird auf die Server-Dokumentation und auf [Eil97] verwiesen.

Im Zusammenhang mit CGI-Programmen ist hier noch zu erwähnen, daß es normalerweise sehr schwierig ist, Fehler in den Programmen zu finden. Die Fehlermeldungen eines CGI-Programms werden bei einem Zugriff darauf nirgends ausgegeben, nur die Fehlermeldung des Servers, daß die Ausführung nicht durchgeführt werden konnte. In diesem Fall ist eine Log-Datei für Fehlermeldungen der CGI-Programme von großem Nutzen. Der Apache-Server bietet eine solche an, was in Kapitel 4.1.3 im Rahmen des Leistungsmanagements näher erläutert wird.

Hilfsprogramme

Die meisten Server bieten eine Programmierschnittstelle an, die unter dem Namen *Application Programming Interface (API)* bekannt ist. Darüber können Anwendungsprogramme die Kernfunktionen eines Servers ansprechen oder mit anderen Programmen Informationen austauschen und somit den schon vorhandenen Konfigurationsanweisungen neue Funktionalitäten hinzufügen oder sie überschreiben. Auch neue Konfigurationsanweisungen können erzeugt werden. Jedes Hilfsprogramm muß in den Server eingebunden werden, wofür jeder Server eigene Mechanismen und Konfigurationsanweisungen verwendet. Der Apache-Server unterhält eine Datei, in der sämtliche eingebundenen Programme (in diesem Fall *Modules* genannt) aufgelistet sind. Um diese zu aktivieren, müssen sie vor der Installation des Servers mit dem gesamten Server-Code compiliert werden. Erst dann können sie anhand der **AddModule**-Anweisung miteingebunden und deren Funktionalität genutzt werden.

Der Netscape Enterprise Server wendet dafür eine andere Methode an. Hier werden neue, vorcompilierte Programme mit Hilfe einer **Init**-Anweisung aus der Server-Konfigurationsdatei eingebunden und müssen nicht mehr mit dem gesamten Server-Code compiliert werden. Anschließend können die neuen Funktionalitäten in der Server-Konfigurationsdatei mit den dafür geschaffenen Anweisungen angewendet werden.

Server-Side Includes

Server-Side Includes stellen einen Mechanismus dar, durch den Daten dynamisch in HTML-Dokumente eingefügt werden können. Ein Dokument wird vor der Auslieferung an den Client auf Server-Side Includes hin untersucht und nach deren Ausführung erst an den Client weitergeleitet. Include-Anweisungen werden in Form von Kommentaren in Dokumente eingebunden:

```
<!--#Anweisung Attribut="Wert" -->
```

Folgende Anweisungen können dabei eingebunden werden:

- **config**: Damit kann das Aussehen von Fehlermeldungen, eines Datumformats oder des Formats einer Dateigröße konfiguriert und die dann mit den folgenden Anweisungen eingefügt werden.
- **include**: Diese Anweisung fügt eine Datei, deren Name unter `Attribut=''Wert''` angegeben wird, in das Dokument ein.
- **echo**: Mit dieser Anweisung lassen sich beliebige Umgebungsvariablen des Servers ausgeben.
- **fsize**: Fügt die Größe der mit `Attribut=''Wert''` gemeinten Datei ein.
- **flastmod**: Hier wird das Datum eingefügt, an dem die unter `Attribut=''Wert''` gemeinte Datei zuletzt geändert wurde.
- **exec**: Diese Anweisung führt CGI-Programme oder Shell-Befehle aus. Diese Option kann in der Konfiguration des Servers allerdings ausgeschaltet werden (siehe nächsten Abschnitt).

Server-Side Includes können wie CGI-Programme durch Hinzufügen eines MIME-Typs oder Handlers (abhängig vom jeweiligen Server) konfiguriert werden. Normalerweise wird der Dateiendung `.shtml` das Parsen von Dateien zugeordnet, so daß jede Datei mit dieser Endung nach Server-Side Includes durchsucht und bearbeitet wird. Allerdings können auch Dateien mit der Endung `.html` Includes enthalten, so daß diese Paarung ebenfalls eingestellt werden kann. Das kann aber zu Performance-Einbußen während des Betriebs führen, weswegen dies die unüblichere Methode ist. Beide Einstellungen müssen über **AddType** bzw. **AddHandler** in der **Ressourcen-Konfigurationsdatei** hinzugefügt werden, damit sie global für den gesamten Server gelten. Bei dem Apache-Server muß dann jeweils noch in dem Verzeichnis, in dem Includes in Dateien eingebunden werden sollen, die Option **Includes** im *Directory Access Control File* ausdrücklich eingeschaltet werden.

Andere Server lösen das anders. Will man beim Netscape Enterprise Server Includes zulassen, so gilt diese Einstellung global und muß verzeichnisspezifisch nicht ausdrücklich eingeschaltet werden. Es wird dann serverweit jede `.shtml`-Datei nach Includes durchsucht. Will man aber nur in bestimmten Verzeichnissen diese Einstellung einschalten, so muß die Funktion in den *Directory Access Control Files* in den jeweiligen Verzeichnissen eingestellt und zusätzlich in der **Ressourcen-Konfigurationsdatei** mit **AllowOverride** für Unterverzeichnisse erlaubt werden. Dann würde der Server jede in diesem Verzeichnis befindliche `.shtml`-Datei nach Einfügungen zu Server-Side Includes durchsuchen. Will mann allerdings nur Includes zulassen, die die `exec`-Anweisung nicht ausführen können, so ist das ausdrücklich mit einer Option (beim Apache lautet sie **IncludesNOEXEC**) auszuschalten. Das gilt übrigens für alle Server, wobei die Konfigurationsanweisungen hierfür serverabhängig sind.

MIME-Typen

Mit Hilfe des MIME-Typs einer Datei kann ein Client bestimmen, wie er die erhaltene Datei bearbeiten soll, um sie entsprechend darzustellen. Normale HTML-Dateien haben den MIME-Typ `text/html`. Deswegen werden sie unbehandelt im Browser angezeigt. Audio- oder

Video-Dateien bedürfen allerdings eines zusätzlichen Anwendungsprogramms, das deren Weiterverarbeitung übernehmen kann, um geeignet dargestellt zu werden. Auf dem Client ist jedem MIME-Typ ein Anwendungsprogramm zugeordnet, dem jede Datei dieses Typs zur Weiterverarbeitung übergeben wird. Auch Server nutzen MIME-Typen, um bestimmte Dateitypen einer Bearbeitungsart zuzuordnen.

Auf dem Server werden diese Paare "MIME-Typ" – "Anwendungsprogramm" meistens in einer eigenen **MIME-Typen-Konfigurationsdatei** gespeichert. Ein neuer Typ kann entweder durch Editieren dieser Datei ergänzt oder über die **Ressourcen-Konfigurationsdatei** mit der Anweisung **AddType** hinzugefügt werden, was die sicherere Methode sein dürfte.

Für Dateien, für die der Server keinen MIME-Typ konfiguriert hat, muß er dem Client einen Default-Wert schicken. Dieser kann durch die Anweisung **DefaultType** bestimmt werden.

Falls auf dem Server auch komprimierte Dateien gespeichert sind, die abgerufen werden können, so kann dem Client im Header mit Hilfe eines `ContentEncoding`-Feldes die Komprimierungsmethode mitgeteilt werden. Deswegen sollte mit einer Anweisung **AddEncoding** der jeweiligen Dateiendung die Komprimierungsmethode zugeteilt werden.

Verzeichnis-Listings

Zeigt ein URL, der auf einen Server gerichtet ist, nicht auf eine Datei, sondern auf ein Verzeichnis, so versucht der Server normalerweise zuerst, die Datei *index.html* zu finden. Der Name dieser Datei, nach der der Server suchen soll, muß für jeden Server mit einer **DirectoryIndex**-Anweisung in dessen Konfigurationsdateien eingestellt sein. Findet er eine solche Datei beim Bearbeiten des Request nicht, so kann er HTML-Text erzeugen, der den Inhalt des Verzeichnisses wiedergibt, was allerdings als sicherheitskritisch angesehen werden kann, da es nicht immer wünschenswert ist, daß Benutzer ohne weiteres den Inhalt eines jeden Verzeichnisses kennen sollen.

Für diese Funktion stellt jeder Server normalerweise zwei Einstellungen zur Verfügung: *plain* und *fancy*. *plain* erzeugt eine einfache Liste aller Dateien und Unterverzeichnisse, die Inhalt des in dem URL angegebenen Verzeichnisses sind. Durch Anklicken einer Datei wird diese vom Server geschickt und durch Anklicken eines Verzeichnisses für diesen eine neue Liste erzeugt. Bei der aufwendigeren Indizierung von Verzeichnissen - *fancy* - werden erklärende Elemente in die Liste mit eingebaut. Das heißt, daß entsprechende Icons zeigen, ob es sich um eine Datei oder ein Verzeichnis oder um welchen Typ von Datei es sich handelt (entsprechend ihrer Endung oder der MIME-Typen); das Datum und die Größe der Datei werden angefügt und optional eine Beschreibung des Dateityps.

Folgende mögliche Anweisungen im Zusammenhang mit der Indizierung des Types *fancy* sind zulässig:

- **AddIcon**: Mit dieser Anweisung wird einem Dateityp ein bestimmtes Piktogramm zugewiesen.
- **AddIconByEncoding**: Hier wird einem Dateityp respektive seiner MIME-Kodierung oder Komprimierungsmethode ein Piktogramm zugeordnet.

- **AddIconByType**: Die Zuweisung eines Piktogramms erfolgt hier auf Basis eines MIME-Typs einer Datei.
- **AddDescription**: Mit Hilfe dieser Anweisung kann einem Dateimuster bzw. -typ oder einer bestimmten Datei eine Beschreibung hinzugefügt werden.

Die oben genannten Anweisungen kommen am häufigsten im Rahmen der Konfiguration von Verzeichnisindices vor und sollen einen exemplarischen Auszug aus allen möglichen Anweisungen darstellen. Für weitere Einstellungen sei auf die Dokumentation der einzelnen Server verwiesen.

Aliases und Redirects

Der Mangel an Speicherplatz oder die Verwendung eines schnelleren Rechners kann die Verlagerung von HTML-Dateien oder anderer Web-Dokumente aus der Document Root nach sich ziehen. Die Möglichkeit der Aufspaltung der Document Root ist deswegen eine sehr nützliche Funktion eines Web-Servers. Physikalisch getrennte Verzeichnisse können so in einer virtuellen Verzeichnishierarchie zusammengefaßt werden.

Der Apache-Server stellt die Konfigurationsanweisung **Alias** zur Verfügung, mit deren Hilfe man einem physikalischen Verzeichnis oder einer Datei einen URL zuweisen kann. In der Ressourcen-Konfigurationsdatei können beliebig viele Alias-Anweisungen verwendet werden.

Ebenso kann ein Client beim Zugriff auf einen URL eines Servers automatisch auf einen anderen URL weitergeleitet werden. Das kann mit Hilfe einer **Redirect**-Anweisung erfolgen. Solche Redirects werden bei den neueren Browsern automatisch ohne Interaktion des Benutzers ausgeführt. Anwendung und großen Nutzen finden diese Redirects im Rahmen der Verwaltung von HTML-Dokumenten auf einem Server. Verschobene Dateien ziehen dann nicht mehr das Problem von veralteten Links nach sich, falls andere Dokumente auf sie verweisen. Redirects werden vom Server an den Client im Rahmen des HTTP-Headers geschickt, nachdem ein Server merkt, daß das vom Client angeforderte Dokument nicht mehr im genannten Verzeichnis zu finden ist. Er teilt dem Client dabei mit, wo er das Dokument jetzt finden kann, so daß der Client daraufhin einen erneuten Request an die neue Adresse schickt.

Dabei gibt es die beiden Möglichkeiten, daß eine Datei entweder nur temporär oder doch dauerhaft an einen anderen Ort verschoben wurde. Mit der Anweisung **RedirectTemp** wird eine temporäre Verschiebung konfiguriert, mit **RedirectPermanent** eine dauerhafte. Beide Anweisungen sind Verfeinerungen der **Redirect**-Anweisung.

Benutzerverzeichnisse

Hier ist speziell im Fall eines Internet-Providers die Situation zu erwähnen, daß dieser privaten Personen oder kleineren Firmen die Möglichkeit bietet, im Internet eine eigene Web-Site zu veröffentlichen. Für diese Zielgruppen lohnt sich die Einrichtung eines eigenen Web-Servers oder die Reservierung einer eigenen Domain nicht. Dafür stellt ein Server die Einrichtung von Benutzerverzeichnissen zur Verfügung, deren Web-Adressen meistens so ausschauen: `http://www.provider.top-level-domain/~user/`.

Hier würden Benutzer also auf dem Server des Providers einen gewissen Platz und eine eigene Web-Adresse für den Zugriff darauf zur Verfügung gestellt bekommen. Die Web-Adresse zu einem Benutzerverzeichnis besteht aus vier Teilen:

1. **Zugriffsprotokoll** (im obigen Beispiel `http`)
2. **Servername** des Providers: `www.provider.top-level-domain`
3. **Benutzerkennung** des Benutzers, angeführt von einem *URL prefix* (z.B. ~)
4. **Pfad der gesuchten Datei**

Wenn für einen Server diese Funktionalität eingeschaltet wird (beim Apache erfolgt das durch die Anweisung **UserDir** `enable/disable` in der Ressourcen-Konfigurationsdatei), müssen erst Benutzerverzeichnisse eingerichtet und konfiguriert werden, in denen dann Benutzer ohne Einwirken des Administrators Dateien ablegen können.

Dabei sind folgende Angaben und Funktionen für die Konfiguration der Benutzerverzeichnisse erforderlich:

- Festlegen eines *user URL prefix*, z.B. ~
- Festlegen des Verzeichnisses, nach dem der Server im Home-Verzeichnis der Benutzer sucht, z.B. `public_html`. In diesem Verzeichnis sind die Dateien abgelegt, auf die mit dem URL

  `http://www.provider.top-level-domain/~user/verzeichnis/datei.html`

  zugegriffen werden kann.
- Festlegen einer Datei, in der alle Namen der Benutzer eingetragen sind, die ein Home-Verzeichnis auf dem Server haben (username, groupid, homedir); das kann auch die *password*-Datei des Systems sein. Der Server überprüft anhand dieser Datei, wo sich das Home-Verzeichnis des Benutzers befindet und sucht darin die gewünschte Datei im Unterverzeichnis `public_html`.
- Festlegen eines Verzeichnisses, in dem User-eigene CGI-Skripten gespeichert werden können. Dafür wird ein Aliasname (**ScriptAlias**) festgelegt. Diese Einstellung muß nur für den Fall vorgenommen werden, daß kein MIME-Typ eingestellt ist, so daß automatisch jede Datei mit der Endung `.cgi`, unabhängig von ihrer Plazierung auf dem Server, als CGI-Skript erkannt wird. Ansonsten werden nur die in diesem einen Verzeichnis befindlichen Dateien als CGI-Skripten ausgeführt. In diesem Fall sind CGI-Programme allerdings potentielle Sicherheitslöcher, da nicht garantiert ist, daß jeder Benutzer sichere CGI-Programme schreibt und somit das System gegen mögliche Einbrecher schützt (siehe dazu auch Konfiguration der CGI-Programme).

Falls sich neue Benutzer anmelden, muß für sie jeweils ein Account eingerichtet werden. Dazu gehört das Anlegen einer neuen Benutzerkennung und des dazugehörigen Paßwortes. Außerdem muß das Ändern des Paßwortes aus Sicherheitsgründen für jeden Benutzer unbedingt möglich sein. Weiter müssen die systemspezifischen Zugriffsrechte für jeden neuen Benutzer vergeben und eingerichtet werden.

Manchmal kann es auch notwendig sein, für einen Server den Server-Account zu ändern. Deswegen müssen zusätzlich beim Ändern des Benutzer- und Gruppennamens, unter denen der

Server läuft, die Eintragungen in der Server-Konfigurationsdatei geändert werden. Danach ist ein Restarten des Servers notwendig, um die neuen Konfigurationseinstellungen zu übernehmen.

4.1.3 Leistungsmanagement

Wenn man einen Web-Server unterhält, so ist man auch daran interessiert, gewisse Daten und Informationen über seine Auslastung zu untersuchen und zu beobachten, welche Dokumente am meisten abgerufen werden oder von welchen Domänen oder Benutzern die meisten Anfragen kommen.

Web-Server protokollieren jeden Zugriff auf zur Verfügung gestellte Dokumente in sogenannten Log-Dateien. Durch Analysieren dieser Log-Dateien erhält man Aufschluß über die Auslastung des Servers, über die Domänen oder IP-Adressen der Hosts, die die Dokumente abrufen, über die Dateien des Servers, die abgerufen werden, über den Zeitpunkt, zu dem eine Datei abgerufen wird, und auch über die Größe der Datei, die in Bytes angegeben ist. Gegebenenfalls können auch über die Benutzer, die die Seiten abrufen, Informationen gesammelt werden, falls deren Client diese Angaben bei einer Anfrage mitschickt. Das erfolgt nur in seltenen Fällen, da die meisten Benutzer ihre Clients so konfigurieren, daß sie keine näheren Angaben über den Benutzer preisgeben. Manche Clients übertragen auch den Namen und die Version des Browsers, mit dem auf die Dateien zugegriffen wird. Auch Fehlermeldungen oder allgemein das Ergebnis jeder einzelnen Übertragung werden in Log-Dateien gespeichert. All diese Daten können dazu genutzt werden, um Langzeitstatistiken über den Datenfluß, der über den Server geht, zu erstellen.

Ein weiteres Aufgabenfeld im Rahmen des Performancemanagements eines Web-Servers besteht darin, die Auslastung der Ressourcen zu überwachen und zu steuern, die von dem Server und den damit zusammenhängenden Anwendungsprogrammen genutzt werden. Die nachfolgende Abbildung soll einen Gesamtüberblick über die Aufgaben geben, die im Rahmen des Performancemanagements eines Web-Servers durchgeführt werden.

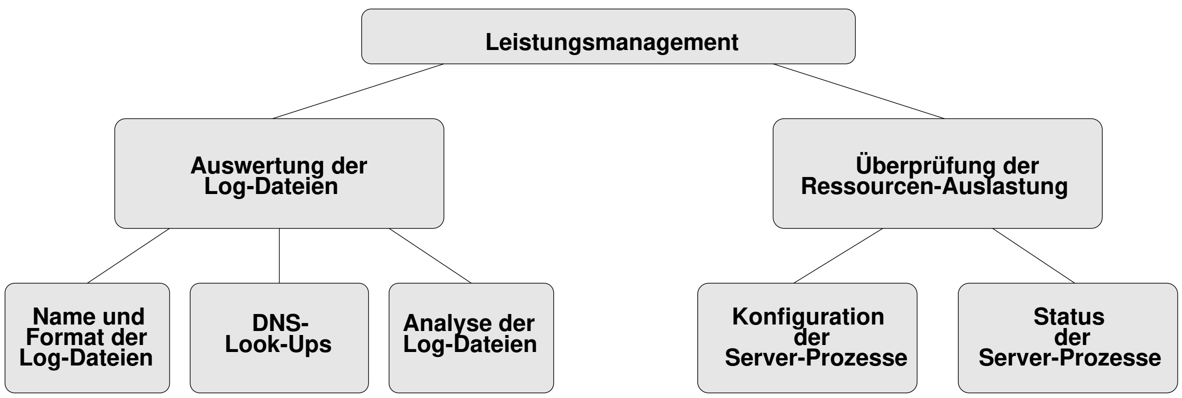

Abbildung 4.4: Leistungsmanagement eines Web-Servers

Auswertung der Log-Dateien

Um Statistiken und Informationen über den Betrieb eines Servers zu erhalten und entsprechend verarbeiten zu können, müssen diese vom Server in einem geeigneten Format zur Verfügung gestellt werden. Nachfolgend werden anhand der von Servern am häufigsten benutzten Log-Dateien die entsprechenden Konfigurations- und Formatanweisungen erläutert.

Name und Format der Log-Dateien

Ein Web-Server protokolliert jeden an ihn gerichteten Request in einer **AccessLog-Datei**. Falls bei der Bearbeitung oder Beantwortung Fehler auftreten, werden diese in einer **ErrorLog-Datei** registriert. Der Apache-Server bietet zusätzlich eine **ScriptLog-Datei** an, in der Fehlermeldungen eingetragen werden, die ein CGI-Programm bei einem fehlerhaften Lauf ausgibt. Falls ein Server auch virtuelle Server unterstüzt, so können für jeden virtuellen Server eigene Access- und ErrorLog-Dateien geführt werden, was den Verwaltungsaufwand erheblich erleichtert.

Die Informationen in der **AccessLog-Datei** werden per Default im *Common Log Format* für Log-Dateien abgespeichert:

```
host rfc931 authuser [date/time] "request" status bytes
```

Die einzelnen Werte bedeuten folgendes:

- `host`: Name oder IP-Adresse des zugreifenden Hosts
- `rfc931`: Login-Name des entfernten Benutzers
- `authuser`: Benutzername, mit dem der Benutzer sich authentifiziert hat
- `[date/time]`: Datum und Uhrzeit, wann die Anfrage ankam
- `request`: Anfragezeile, wie sie vom Client gesendet wurde
- `status`: *HTTP Status Code*, der an den Client nach Bearbeiten der Anfrage zurückgegeben wurde (siehe Anhang A)
- `bytes`: Länge des übertragenen Dokuments in Bytes

Der `request`-Eintrag, der protokolliert wird, kann in weitere Einheiten aufgeteilt und entsprechend untersucht werden. Dies ist z.B. dann notwendig, falls ein Fehler bei der Suche nach einer Datei aufgetreten oder ein unerlaubter Zugriff auf eine Datei erfolgt ist. So kann z.B. erkannt werden, daß bestimmte Dateien immer wieder bestimmte Fehler hervorrufen. Ein `request`-Eintrag kann also in folgende Angaben unterteilt und untersucht werden:

- **HTTP-Methode**, mit der auf die Datei zugegriffen wird
(GET, HEAD, PUT, POST, DELETE)
- **URL** der abgerufenen Datei
- **HTTP-Version**

Manche Server protokollieren auch den Namen und die Version des zugreifenden Browsers, von dem aus der Zugriff erfolgt. Ebenso kann das Format der Eintragungen in den Log-Dateien nach Belieben mit Hilfe von festgelegten **Formatanweisungen** an eigene Bedürfnisse angepaßt werden, die hier aber nicht weiter erläutert werden. Eine Formatanweisung ist aber im Rahmen der Performanceanalyse eines Web-Servers zu erwähnen, und zwar `%T`, die die Zeit angibt, die der Server für die Bearbeitung einer Anfrage gebraucht hat und die aus diesem Grund in die AccessLog-Datei eingefügt werden sollte. Fehlt eine Eintragung, so wird statt dessen ein Bindestrich eingefügt. Meistens tritt das im Falle des Login- und Benutzernamens ein, da diese häufig vom Client nicht mit übertragen werden. Ein Beispieleintrag in der AccessLog-Datei würde folgendermaßen aussehen:

```
myserver.mydomain.tld - -  [08/Oct/1997:10:44:51 +0200] "GET
/proj/Literatur/public-htdocs/MNMPub/index.shtml HTTP/1.0" 200 3094
```

oder

```
195.123.45.6 - -  [08/Oct/1997:10:44:51 +0200] "GET
/proj/Literatur/public-htdocs/MNMPub/index.shtml HTTP/1.0" 200 3094
```

Eintragungen in einer **ErrorLog-Datei** enthalten Informationen über Fehler, die während des Betriebs des Servers beim Bearbeiten oder Beantworten von Anfragen erfolgen, oder allgemeine Informationen über den Betrieb des Servers. Beispiele für ErrorLog-Einträge sind folgende:

```
[Tue Aug  5 11:52:51 1997] access to /users/wiss/hauck/.html-data/hauck.gif
failed for fast.fast.de, reason: File does not exist
```

oder

```
[Tue Aug  5 16:15:05 1997] access to /users/wiss/langer/.html-data
failed for hphalle2h.informatik.tu-muenchen.de, reason: Client denied by
server configuration
```

In der **ScriptLog-Datei** des Apache-Servers wird die Ausgabe von CGI-Programmen gespeichert, die nicht ordnungsgemäß ausgeführt werden konnten. Dabei werden folgende spezifischen Informationen in die Datei eingetragen:

- Dateiname des CGI-Skripts
- HTTP-Status der Server-Antwort
- Inhalt der Client-Anfrage inklusive aller HTTP-Header
- Alle HTTP-Header, die das CGI-Programm ausgegeben hat
- Ausgabe des CGI-Programms auf STDOUT (optional, falls eine Ausgabe erfolgt ist)
- Ausgabe des CGI-Skripts auf STDERR (optional, falls eine Ausgabe erfolgt ist)
- Fehlermeldung (diese Ausgabe erscheint in der ScriptLog-Datei, falls das CGI-Programm erst gar nicht ausgeführt oder gefunden werden konnte)

Zwei weitere Konfigurationsanweisungen stehen im Zusammenhang mit der ScriptLog-Datei zur Verfügung. Mit Hilfe der Anweisung **ScriptLogLength** kann eine maximale Größe angegeben werden, auf die die Größe der Log-Datei beschränkt wird. Die Angabe erfolgt in Bytes. Ebenso kann die Größe des Inhalts der Client-Anfrage beschränkt werden, der in die Log-Datei miteingebunden wird. Die Konfigurationsanweisung dazu lautet **ScriptBufferLength**. Gerade bei POST- und PUT-Anfragen, mit denen CGI-Programme aufgerufen werden, werden große Mengen an Daten übertragen, so daß die zwei zuletzt genannten Anweisungen sicherlich sehr sinnvoll sind.

DNS-Look-Ups

Wenn ein Client auf einen Web-Server zugreift, so kennt dieser erst einmal nur die IP-Adresse des Clients. Der Eintrag in der AccessLog-Datei würde nicht den Hostnamen des Clients anzeigen, sondern nur die IP-Adresse. Um das zu ändern, stellen Web-Server die Funktion zur Verfügung, daß sie bei jedem Zugriff eines Clients einen DNS-Look-Up durchführen; falls der zu der IP-Adresse gehörige Hostname im DNS eingetragen ist, fügt der Server diesen in der AccessLog-Datei ein. Mit Hilfe der zugehörigen Konfigurationsanweisung **HostNameLookUps** kann diese Funktionalität für Web-Server aktiviert werden.

Einen Nachteil hat dies allerdings schon, da es dadurch zu Performance-Einbußen im Betrieb des Servers kommen kann, wenn bei jedem Zugriff ein DNS-Look-Up durchgeführt werden muß. Eine Abhilfe könnte mit Hilfe eines DNS-Cache erreicht werden, so daß einmal abgefragte IP-Adressen gespeichert werden und nicht jedesmal neu abgefragt werden müssen. Diese Funktionalität wird aber normalerweise von Web-Servern noch nicht unterstüzt. Eine weitere Möglichkeit, DNS-Look-Ups während des Betriebs zu umgehen, wäre, eine Auswertung der IP-Adressen erst bei der Analyse von Log-Dateien durchzuführen. Das hätte den Vorteil, daß nur eine einmalige Aktion gestartet wird, in der für jeden IP-Adressen-Eintrag in der Log-Datei ein DNS-Look-Up durchgeführt wird. So würde der Betrieb des Servers nicht eingeschränkt werden. Wie dieses Dilemma gelöst wird, liegt im Ermessen des jeweiligen Systemadministrators.

Analyse der Log-Dateien

Die Analyse der AccessLog-Datei kann nach unterschiedlichen Gesichtspunkten erfolgen. Über jede Art der zur Verfügung stehenden Daten können je nach Bedarf und Anwendungsfeld entsprechende Statistiken geführt werden. Um einen Web-Server kontinuierlich zu überwachen und ein aktuelles Bild über seine Auslastung zu haben, ist die Einrichtung von Zählern ein hilfreiches und notwendiges Werkzeug. Durch ständiges Aktualisieren der Zählerwerte kann eine aktuelle Übersicht darüber erhalten werden, wie stark der Server und dessen Ressourcen ausgelastet sind. Je nach Bedürfnissen und Vorlieben werden von den Systemadministratoren oder den Inhalte-Anbietern bestimmte Zähler bevorzugt. Anschließend sollen einige Beispielszenarien erläutert werden, wobei kein Anspruch auf Vollständigkeit erhoben wird.

Einen Systemadministrator interessieren sicherlich Daten über die Auslastung und Performance des Servers mehr, wohingegen ein Inhalte-Anbieter eher daran interessiert ist, welche

Dokumente am häufigsten abgerufen werden, von welchen Domänen die meisten Anfragen kommen usw.

Daten über die Auslastung des Servers können anhand der folgenden Informationen aus der AccessLog-Datei gewonnen werden:

- **Datum- und Zeitangaben** (wann sind die Anfragen erfolgt?)
- **Zeitangabe** (wie lange hat der Server zur Bearbeitung jeder Anfrage benötigt?)
- **HTTP Status Code** (gibt Aufschluß über den Erfolg oder Mißerfolg bei der Bearbeitung der Anfragen)
- **Bytes-Anzahl** (die der Server an Clients übertragen hat)

Ein Zähler über die Gesamtanzahl der Einträge in der Log-Datei gibt einen groben Überblick über den Arbeitsaufwand und die Auslastung des Servers. Anhand der **Datum- und Zeitangaben** aus der AccessLog-Datei können zeitbezogene Statistiken erstellt werden. Zähler können so konfiguriert werden, daß sie Aufschluß darüber geben, wie viele Anfragen z.B. in einer Stunde ankamen, an einem Tag oder in einem Monat. Angaben über die **Zeit, die der Server zum Bearbeiten einer Anfrage** gebraucht hat, können je nach Bedarf entweder einzeln oder für mehrere Anfragen über bestimmte Zeitintervalle untersucht und analysiert werden. So kann beobachtet werden, wann der Server am meisten ausgelastet ist und zur Bearbeitung der Anfragen mehr Zeit benötigt als in ruhigeren Zeiten. Richtet man Zähler ein, die eine Übersicht über die Häufigkeit der **HTTP Status Codes** geben, so kann man gut erkennen, wie viele Anfragen der Server erfolgreich bearbeitet hat, wie viele fehlgeschlagen sind oder wie viele Anfragen umgeleitet wurden, je nach Wert und Bedeutung des Status Codes. Die Anzahl der **übertragenen Bytes** gibt Aufschluß über die Menge der verschickten Daten und kann dazu verwendet werden, um den Anteil des Web-Servers am Gesamt-Verkehrsaufkommen der Daten im gesamten System zu ermitteln.

Einige nützliche Zähler, die Aufschluß über die Auslastung des Servers geben können, sind folgende:

- **TotalRequests**: Summe aller Einträge in der AccessLog-Datei
- **SuccessfulRequests**: Anzahl der erfolgreich bearbeiteten Anfragen
- **BadRequests**: Anzahl der fehlgeschlagenen Anfragen
- **AcceptedRequests**: Anzahl der angenommenen, aber aufgrund mangelnder Ressourcen nicht bearbeiteten Anfragen
- **MovedTemporarily**: Anzahl der Anfragen auf temporär verschobene Dateien
- **MovedPermanently**: Anzahl der Anfragen auf dauerhaft verschobene Dateien
- **HTTPStatusRequests**: Anzahl eines bestimmten HTTP-Status Codes in der AccessLog-Datei
- **RequestsPerMonth**: Anzahl der Anfragen in einem Monat
- **RequestsPerDay**: Anzahl der Anfragen an einem Tag

- **ServerTimeMonth**: Zeit, die der Server in einem Monat für die Bearbeitung der Anfragen benötigt hat
- **ServerTimeDay**: Zeit, die der Server an einem Tag für die Bearbeitung der Anfragen benötigt hat
- **TotalBytes**: Menge der verschickten Daten, seit dem Zeitpunkt, an dem die Log-Datei geöffnet wurde
- **BytesPerMonth**: Menge der verschickten Daten im Monat (falls der Serveradministrator für jeden Monat eine neue Log-Datei anlegt, so ist dieser Wert gleich dem Wert des Zählers TotalBytes)
- **BytesPerDay**: Menge der verschickten Daten an einem Tag

Informationen darüber, welche Dateien am meisten abgerufen wurden und von welchen Domänen oder Benutzern die meisten Zugriffe erfolgten, geben folgende Daten in der AccessLog-Datei:

- **Hostname** oder **IP-Adresse** des zugreifenden Hosts
- **Login-Name** des zugreifenden Benutzers (falls vom Client geliefert)
- **Benutzername** des zugreifenden Benutzers (falls vom Client geliefert)
- **HTTP-Methode**, mit der auf die Datei zugegriffen wurde
- **HTTP-Version**, die der Client unterstützt
- **URL** der abgerufenen Datei, bestehend aus dem Pfad und dem Namen der Datei
- **Browser**, mit dem auf die Datei zugegriffen wurde

Wird von Inhalte-Anbietern eine Statistik über die **zugreifenden Hosts** verlangt, so können verschiedene Zähler vorteilhaft sein. Es kann ein Zähler eingerichtet werden für jeden Host und dessen Gesamtanzahl an Zugriffen. Andererseits wäre eine Statistik über die Top-Level-Domains, aus denen die Zugriffe erfolgen, von großem Vorteil für Anbieter, die sich dafür interessieren, aus welchen Ländern die meisten Anfragen kommen. Dadurch könnten sie beispielsweise ihre Inhalte an die jeweilige Landessprache anpassen. Allerdings ist zu dieser Art der Zugriffsstatistiken noch anzumerken, daß sie nicht hundertprozentig aussagekräftig sind, da Clients auch auf Umwegen Zugriffe über Proxies starten, die manche oder alle angeforderten Dateien im Cache haben und somit keinen neuen Zugriff an den eigentlichen Server schicken. Diese Anzahl der Zugriffe geht dem eigentlichen Server, der die Dateien anbietet, für seine Statistik verloren. Außerdem werden in den Fällen, in denen ein Proxy auf Dateien zugreift, der Name und die Domäne des Proxy-Servers als zugreifender Host vermerkt, so daß hier auch keine getreuen Angaben über den Ursprung der Anfragen zu erzielen sind.

Spezielle Statistiken über den **Login-** oder **Benutzernamen** der zugreifenden Benutzer sind schwierig zu erstellen, da der Client diese Information nur in seltenen Fällen mitschickt. Eine Möglichkeit, das Nutzerverhalten zu analysieren, ist das Anwenden von **Cookies**. Cookies sind kleine Stückchen Information, die der vom Benutzer verwendete Client im Auftrag des Web-Servers zunächst im Speicher des Client-Rechners festhält und unter Umständen bei Verlassen des Clients in eine Datei schreibt. Der Server kann nicht, wie es teilweise in der Literatur ungenau formuliert wird, selber auf die Platte des Benutzers schreiben. Da dieses Schreiben also vom Web-Server initiiert wird, kann er auch nur schreiben (lassen), was er sowieso schon weiß. Das können z.B. Daten sein, die ein Benutzer beim Ausfüllen eines Formulars in den entsprechenden Feldern einträgt. Falls der Kunde diese Daten für das Ausfüllen eines weiteren Formulars benötigt, so kann eine erneute Angabe dieser Daten durch Verwenden eines Cookie vermieden werden. Bei jedem neuen Verbindungsaufbau mit diesem Web-Server sendet der Benutzer-Client die gespeicherte Information an den Server zurück. Allerdings ist dies auch keine zuverlässige Methode, das Verhalten der Benutzer zu untersuchen, da viele es nicht erlauben, daß der Web-Server solch ein Cookie bei ihnen einsetzt. Die meisten Clients erlauben es, entweder fallweise oder generell, Cookies abzuweisen; ebenso gibt es Clients, die Cookies erst gar nicht unterstützen.

Weitere Statistiken können über die einzelnen **Dateien** erstellt werden, die verschickt wurden. So können Zähler eingerichtet werden, die die Anzahl der übertragenen Dateien eines bestimmten Typs addieren, oder die in einem bestimmten Verzeichnis plaziert sind. Andererseits ist ein Zähler interessant, der angibt, wie viele Zugriffe auf die Index-Seite oder sogenannte Willkommens-Seite eines Anbieters erfolgt sind. Diese Art von Zähler, der die Zugriffe nach speziellen Dateinamen aufschlüsselt, kann für jede andere Datei sinnvoll sein.

Interessant sind auch Gesamtangaben über die Art des Zugriffs auf Dateien. Damit sind die **HTTP-Zugriffsmethoden** GET, PUT usw. gemeint. Ein sehr wichtiger Aspekt wird von Statistiken über die **HTTP-Version** abgedeckt. Wie weiter oben in dem Unterkapitel über virtuelle Server schon erwähnt wurde, unterstüzt die HTTP-Version 1.1 namensbasierte Anfragen an Hosts. Falls ein Anbieter seine Seiten über einen namensbasierten virtuellen Server anbietet, wird er abwägen müssen, ob er die Anfragen, die mit Hilfe des veralteten HTTP-Protokolls gestellt werden, ignorieren kann. Der Anteil der Anfragen mit der neuen Version gegenüber alten Versionen des HTTP-Protokolls kann aus dieser Statistik entnommen werden.

Falls in einem Eintrag protokolliert wird, mit welchem **Browser** auf die Dateien zugegriffen wurde, so könnte in einer Liste aufgezählt werden, wie viele Anfragen mit welchem Browser durchgeführt wurden.

Zusammengefaßt wären folgende Zähler für diese Art von Zugriffsstatistiken interessant:

- **RequestsPerHost**: Anzahl der Anfragen von einem bestimmten Host
- **RequestsPerDomain**: Anzahl der Anfragen von einer bestimmten Domäne
- **FileTypeRequests**: Anzahl der Zugriffe auf Dateien eines bestimmten Typs
- **FileNameRequests**: Anzahl der Zugriffe auf eine bestimmte Datei
- **DirectoryRequests**: Anzahl der Zugriffe auf Dateien, die in einem bestimmten Verzeichnis gespeichert sind
- **HTTPMethodRequests**: Anzahl der Zugriffe, die mit einer bestimmten HTTP-Methode durchgeführt wurden
- **HTTPVersionRequests**: Anzahl der Zugriffe, die mittels einer bestimmten HTTP-Version durchgeführt wurden
- **BrowserRequests**: Anzahl der Anfragen, die über einen bestimmten Browser erfolgten

Die Analyse der ErrorLog- und ScriptLog-Dateien wird im Rahmen des Fehlermanagements in Kapitel 4.1.6 erläutert.

Überprüfung der Ressourcen-Auslastung

Ressourcen, die die Performance eines Servers beeinflussen, sind die gesamten Prozesse, die ankommende Anfragen bearbeiten. Solche Prozesse sind z.B. die Kopien des Server-Dämons, CGI- oder Java-Prozesse. Als Ressourcen können auch die offenen Verbindungen eines Servers angesehen werden, über die er mit den Client-Prozessen Informationen austauscht. Um die Auslastung des Servers zu untersuchen und zu überwachen, ist eine systembezogene Untersuchung notwendig, wie viele Systemressourcen in Anspruch genommen werden.

Dementsprechend sollte der Ressourcenverbrauch des Servers oder auch der virtuellen Server kontrollierbar bzw. begrenzbar sein. Es ist wichtig, daß einzelne Prozesse nicht zuviel CPU-Zeit in Anspruch nehmen oder nicht zuviel Speicherplatz belegen. Ebenso sollten Prozesse nicht unbegrenzt viele Kind-Prozesse aufrufen bzw. starten dürfen, die ihrerseits ebenfalls Ressourcen des Servers und des Systems verbrauchen.

Außerdem sollte ein wichtiger Punkt im Zusammenhang mit den virtuellen Servern beachtet werden. Jeder virtuelle Server, der auf dem System installiert wird, verbraucht zusätzlich die Ressourcen des Systems, die schließlich nicht unbegrenzt zur Verfügung stehen. Ein virtueller Server nimmt Anfragen entgegen, startet Server- und Anwendungsprozesse, um diese zu bearbeiten, und öffnet Verbindungen, um die Anfragen zu empfangen und die Antworten zurückzuschicken. Somit ist es z.B. notwendig, bestimmen zu können, wie viele Prozesse maximal für einen virtuellen Server verwendet werden dürfen.

Konfiguration der Server-Prozesse

Die meisten Server geben dem Administrator entsprechend seines Systems die Möglichkeit, die Anzahl der Prozesse zu bestimmen, die beim Start des Servers von diesem erzeugt werden oder die während des Betriebs laufen dürfen. Der Apache-Server stellt hierfür fünf Konfigurationsanweisungen zur Verfügung, die im Rahmen der Grundkonfiguration innerhalb der **Server-Konfigurationsdatei** eingefügt werden.

Mit Hilfe der Anweisung **StartServers** wird festgelegt, wie viele Kindprozesse der Server beim Start erzeugen soll. Diese Prozesse warten anschließend während des Betriebs auf ankommende Anfragen. Die beiden Anweisungen **MinSpareServers** und **MaxSpareServers** legen die Anzahl der leerlaufenden Prozesse fest, die während des Betriebs je nach Auslastung des Servers dynamisch angepaßt wird. So kann bestimmt werden, wie viele Prozesse immer mindestens oder maximal als Reserve neben den arbeitenden laufen sollen, um schnell aufeinander ankommende Anfragen zügig beantworten zu können. Hier ist besonders auf den im System vorhandenen Speicher zu achten, da eine zu große Anzahl von Prozessen das System überlasten würde. Um Speicherprobleme in Form von Speicherlecks zu umgehen, kann die Lebensdauer eines Kindprozesses begrenzt werden. Mit der Anweisung **MaxRequestsPerChild** kann die maximale Anzahl der Anfragen festgelegt werden, die ein Kindprozeß vor seinem automatischen Beenden bearbeiten soll. Die absolut höchste Anzahl an Kindprozessen, die jemals gleichzeitig auf dem System laufen dürfen, kann mit der Konfigurationsanweisung **MaxClients** festgelegt werden. Ist diese Anzahl erreicht, während weitere Anfragen für den Server ankommen, so werden sie in einer Warteschlange eingereiht und erst beim Freiwerden eines Kindprozesses der Reihe nach bearbeitet.

Ebenso können für die Client-Verbindungen einige Einstellungen festgelegt werden. Mit der Anweisung **KeepAlive** kann die Unterstützung der persistenten Verbindungen ein- und ausgeschaltet werden. Der Vorteil der persistenten Verbindungen liegt darin, daß mehrere Anfragen über eine TCP-Verbindung geschickt werden können und somit nicht für jede Anfrage eine eigene Verbindung aufgebaut werden muß. Wie viele aufeinanderfolgende Anfragen über eine persistente Verbindung erlaubt werden, kann mit der Anweisung **MaxKeepAlive** festgelegt werden. Wie lange auf weitere Anfragen über eine persistente Verbindung gewartet wird, kann mit der Anweisung **KeepAliveTimeout** in Sekunden angegeben werden. Mit der Anweisung **Timeout** kann die Zeitspanne in Sekunden festgelegt werden, in der eine Client-Verbindung folgender Art bearbeitet wird (siehe dazu [Eil97]):

- Die Gesamtzeit zwischen dem Aufbau der Client-Verbindung und der Anfrage darf nicht länger als die Timeout-Zeitspanne sein.
- Die Zeit zwischen dem Erhalt von TCP-Paketen bei POST- oder PUT-Anfragen darf nicht länger als die Timeout-Zeitspanne sein.
- Die Zeit zwischen den Bestätigungen von TCP-Paketen, die als Antwort vom Server geschickt wurden, darf nicht länger als die Timeout-Zeitspanne sein.

Wenn bei einem dieser Fälle die Zeitspanne überschritten wird, so schließt der Server die Verbindung zum Client.

Status der Prozesse

Nachdem die anwendungsspezifischen Einstellungen zu den Prozessen und Verbindungen des Servers festgelegt werden, können systemspezifische Informationen über die verbrauchten Ressourcen des Servers abgefragt werden, da ein Web-Server normalerweise keinerlei Angaben darüber zur Verfügung stellt.

Für jeden einzelnen Prozeß können normalerweise folgende systemspezifischen Werte abgefragt werden:

- **UID**: Name des Users, der den Prozeß gestartet hat
- **PID**: ID des Prozesses
- **PPID**: PID des Prozesses, der diesen Prozeß gestartet hat
- **C**: die Bearbeitungspriorität des Prozesses
- **%CPU**: Anteil verbrauchter CPU-Zeit in Prozent
- **%MEM**: Anteil des verbrauchten RAM-Speicher in Prozent
- **SIZE**: Summe der Programmdaten und der Daten aus dem Stack (in Kilobytes)
- **RSS**: vom Prozeß benötigtes RAM (in Kilobytes)

- **TTY**: Terminal, von dem aus der Prozeß gestartet wurde
- **STAT**: Status des Prozesses
- **START**: Zeit, zu der der Prozeß gestartet wurde (Angabe in Stunden und Minuten)
- **TIME**: bisher verbrauchte CPU-Zeit
- **COMM**: Kommandozeile (mit Argumenten), mit der der Prozeß gestartet wurde

Der Apache-Server stellt ein Status-Modul zur Verfügung, das bei entsprechender Konfiguration eine Status-Seite ausgeben kann. Diese enthält unter anderem Informationen über die Anzahl der belegten und freien Server-Prozesse, über die Anzahl der bearbeiteten Anfragen oder der übertragenen Datenmenge. Die Status-Informationen können auch in einer maschinenlesbaren Form oder in einer Log-Datei ausgegeben werden.

4.1.4 Sicherheitsmanagement

Das Sicherheitsmanagement betrifft mehrere Aspekte des Zugangs und Zugriffs von Benutzern auf einen Web-Server. Zum einen müssen auf **Dateisystemebene** Vorkehrungen getroffen werden, in denen festgelegt wird, welche Benutzer auf welche Server-Ressourcen in welcher Art und Weise zugreifen dürfen. Es sollen nur Benutzer mit gültigen Zugriffsrechten Dateien verändern, hinzufügen oder löschen dürfen, und das auch nur in bestimmten Verzeichnissen. Auch das Ausführen von bestimmten Programmen soll nur in bestimmten Verzeichnissen möglich sein und auch nur für bestimmte Benutzer. So kann das Ausführen von CGI-Programmen ein Risiko für den Server bedeuten, da manche Programme von Eindringlingen dazu genutzt werden können, um auf vertrauliche Informationen des Servers oder des Systems zuzugreifen. Deswegen ist eine Überprüfung jedes einzelnen CGI-Programms, das auf einem Server ausgeführt wird, von großer Bedeutung, was allerdings in den meisten Fällen mit einem sehr großen und vielleicht unvertretbaren Aufwand verbunden sein kann, so daß in solchen Fällen andere Maßnahmen in Erwägung gezogen werden sollten. Ein hohes Sicherheitsverlangen besteht vor allem bei Internet Providern, die ihren Kunden Zutritt zu gewissen Verzeichnissen auf dem Server gewähren und somit vielen potentiellen Angreifern und Gefahrenquellen auf diesem Weg ausgesetzt sind.

Zum anderen betreffen Sicherheitseinrichtungen eines Web-Servers den **öffentlichen Zugriff** auf Dokumente, die der Server im Intra- oder Internet bereitstellt. Manche Anbieter möchten ihr Angebot nur für eine bestimmte Menge von Personen und Gruppen öffnen oder Zugriffe nur von bestimmten Domänen oder Hosts zulassen. Auch der umgekehrte Fall kann auftreten, daß prinzipiell jedem der Zugriff auf die Dokumente des Servers erlaubt und nur bestimmten Personen, Hosts oder Domänen verwehrt wird.

Ein weiterer sehr wichtiger Sicherheitsaspekt betrifft die **Sicherung der über das Internet übertragenen Daten**. Dieser Aspekt ist besonders im kommerziellen Umfeld der Internet-Nutzung von großer Bedeutung, wo persönliche Daten von Benutzern betroffen sind, z.B. deren Kreditkartennummer oder Adresse. Damit Dritte nicht unerlaubten Zugriff auf diese Daten bekommen und sie mißbrauchen, muß im Rahmen der Sicherheitsvorkehrungen eine Verschlüsselung der Daten erfolgen und somit die Einrichtung von Kodiermechanismen.

Folgende drei Hauptaufgaben umfaßt also das Sicherheitsmanagement (siehe Abb. 4.5):

- Benutzerverwaltung
- Zugriffsberechtigung
- Einrichten von Kodiermechanismen

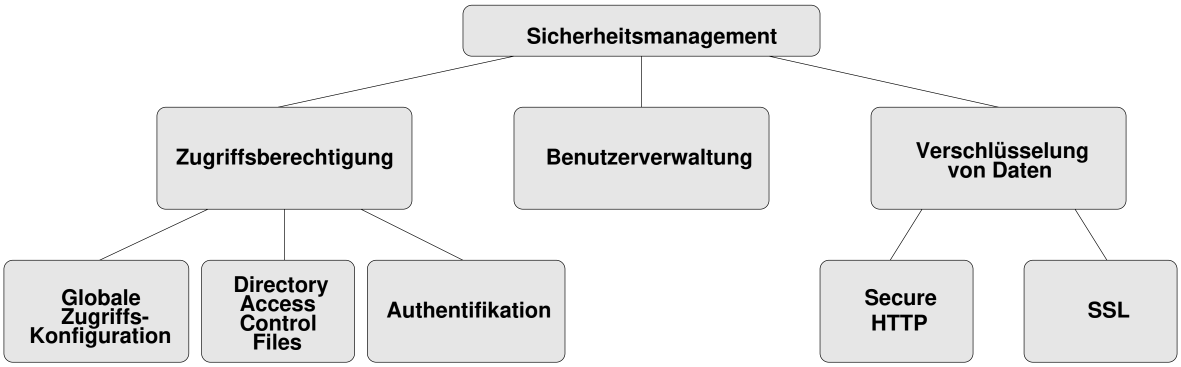

Abbildung 4.5: Sicherheitseinrichtungen eines Web-Servers

Einem Administrator stehen entsprechende Mechanismen zur Verfügung, mit deren Hilfe er eine sehr feine Granularität der Zugriffsrechte von Benutzern auf Dateien und Verzeichnisse errreichen kann. Er kann sowohl auf Betriebssystem-Ebene als auch durch die Rechtevergabe eines Web-Servers sehr genau bestimmen, wer in welcher Art und Weise auf welche Dateien zugreifen darf. Die Verknüpfung beider Mechanismen kann bei richtiger Anwendung eine sehr hohe Sicherheit für die Dateien des Servers zur Folge haben.

Benutzerverwaltung

Für Administratoren und Inhalte-Anbieter, die den Server und desssen Inhalte warten und pflegen, ist es notwendig, daß sie Schreibrechte auf Dateien in bestimmten Verzeichnissen besitzen. Diese werden im Rahmen der **Benutzerverwaltung** systembasiert auf Dateiebene vergeben. Hier muß auch festgelegt werden, welche Benutzer lokal auf dem Server arbeiten können und welche remote einen physikalischen Zugang erhalten, um Dateien ablegen zu können. Grundsätzlich befaßt sich dieser Aufgabenbereich mit Fragen, die den direkten Zugriff von Benutzern auf Dateien und Verzeichnisse des Servers betreffen, die sich unter der Server oder Document Root befinden. Dieser Zugriff erfolgt **nicht** aus dem Inter- oder Intranet über Browser, sondern dient allein der Wartung und Pflege der Dateien, die den Betrieb des Servers regeln, und der Dokumente, die das Web-Angebot des Servers darstellen.

Der allerwichtigste Fall betrifft den Zugang zu den Server-Binaries, den Log- und Konfigurationsdateien, eben den Dateien und Verzeichnissen in der Server Root. Es muß sichergestellt werden, daß Unbefugte keinen Zugriff auf den Server haben. Speziell der Benutzer-Account, unter dem der Server während des Betriebs läuft, bedarf einer näheren Betrachtung. Der User, dem der Server zugeordnet ist, sollte nur auf Dateien Zugriff haben, die für jeden anderen Internet-Benutzer auch lesbar sind. Außerdem darf sich niemand unter diesem User-Login einloggen können, das heißt, daß der Account als gesperrt einzutragen ist. Bei der Vergabe von Schreibrechten auf Konfigurations- und Log-Dateien ist auch erhöhte Vorsicht geboten. Nur Administratoren sollten Zugriff auf diese Dateien haben.

Daten und Verzeichnisse unter der Document Root erfordern die Betrachtung des Sicherheitsaspekts aus einer anderen Perspektive. Hier ist von Interesse, welche Benutzer in welchen Verzeichnissen auf dem Server Dateien ablegen, entfernen oder verändern können.

Für Benutzer, die remote zu einem Unterverzeichnis der Document Root Zugang haben müssen, ist es wichtig, daß sie z.B. per ftp Dateien darin ablegen oder daraus laden können. Hier muß der Administrator des Servers folgende Daten pro Benutzer festlegen:

- Benutzer-Name
- Benutzer-Gruppe
- Paßwort des Benutzers
- Verzeichnis, zu dem er Zugang erhält
- Dateien, für die er besondere Rechte bekommt
- Art des Zugangs (ftp, telnet)
- Art der Berechtigung

  read : Gilt diese Berechtigung für eine Datei oder ein Verzeichnis, so hat der Benutzer nur die Möglichkeit, den Inhalt der Ressource zu lesen, nicht aber sie zu verändern.

  write : Diese Berechtigung ermöglicht einem Benutzer, den Inhalt einer Datei zu verändern oder, falls es auf ein Verzeichnis bezogen ist, alle darin enthaltenen Dateien zu verändern, sofern für diese einzeln keine anderen Rechte bestimmt sind.

  execute : Gilt diese Einstellung für eine Datei, so darf der Benutzer diese ausführen. Bezogen auf ein Verzeichnis bedeutet es, daß der Benutzer überhaupt die Berechtigung hat, in das Verzeichnis zu wechseln.

- wann der Benutzer sich einloggen kann (Tag und Uhrzeit)
- von wo aus er sich einloggen kann (physikalischer Ort)

In den Benutzerverzeichnissen des Servers sollen alle Kunden eines Internet Providers jeweils unter ihrem eigenen Verzeichnispfad (etwa `users/benutzerkennung/public_html/`) Dateien ablegen können. Allerdings stellt die Berechtigung der Kunden zur Ausführung von Programmen in den Benutzerverzeichnissen ein Sicherheitsrisiko für den Server dar. Wie vorher schon erwähnt, können fehlerhafte CGI-Programme in diesem Fall zum Mißbrauch der Rechte benutzt werden. Deswegen sollte das Ausführen von CGI-Programmen nur in bestimmten Verzeichnissen auf dem Server erlaubt sein, zu denen nur vertrauenswürdige Personen Zugang haben. In diesem Fall ist von einer Einrichtung eines MIME-Typs oder Handlers für die Ausführung von CGI-Programmen abzuraten, da dann jede Datei mit der Endung `.cgi`, unabhängig von ihrer Plazierung vom Server, als solche erkannt und ausgeführt wird, außer für den Fall, daß der Server (wie z.B. der Apache oder auch der Netscape Enterprise Server) eine zusätzliche Aktivierung von CGI-Programmen im jeweilgen Verzeichnis, in dem die Programme ausgeführt werden sollen, verlangt. Diese darf in diesem Fall für Benutzerverzeichnisse nicht erteilt werden.

Zugriffsberechtigung

Dieser Bereich des Sicherheitsmanagements legt den Schwerpunkt auf den Zugriff von Benutzern aus dem Internet auf öffentliche Seiten eines Servers. Damit sind hauptsächlich Dateien und Unterverzeichnisse unterhalb der Document Root gemeint. Der Zugriff kann auf Basis von IP-Adressen, von Domain-Namen oder benutzerbasiert erfolgen oder verwehrt werden und teilt sich in folgende Unterbereiche:

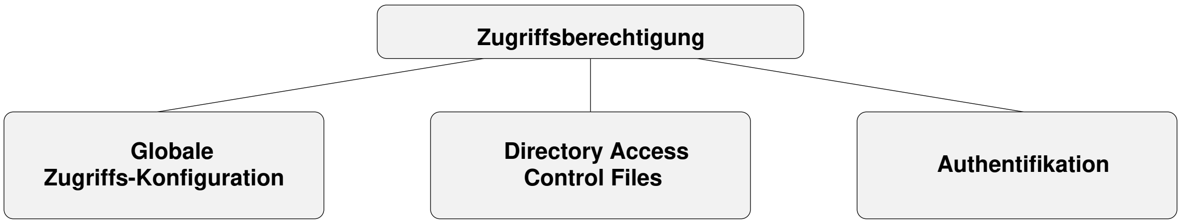

Abbildung 4.6: Zugriffskonfiguration eines Web-Servers

Globale Zugriffskonfiguration

Globale, den ganzen Server betreffende Zugriffsbestimmungen werden in der **Access-Konfigurationsdatei** des Servers festgelegt. Zugriffsberechtigungen werden hier verzeichnisbasiert für einzelne Verzeichnisse vergeben, können aber in den Verzeichnissen selbst durch Plazierung von *Directory Access Control Files* (siehe weiter unten) überschrieben werden. Falls das für bestimmte Verzeichnisse auf keinen Fall erwünscht ist, muß in der Konfigurationsdatei die Anweisung **AllowOverride** des Überschreibens im Zusammenhang mit diesen Verzeichnissen ausgeschaltet werden.

Der Vorteil, der sich daraus ergibt, daß Zugriffseinstellungen global in der **Access-Konfigurationsdatei** des Servers festgelegt werden, besteht darin, daß man auf einen Blick sehen kann, für welche Verzeichnisse welche Zugriffe erlaubt oder verboten werden. So muß nicht in jedem Verzeichnis danach gesucht werden. Für feinere Abstufungen der Zugriffsberechtigung können dann die *Directory Access Control Files* in den jeweiligen Verzeichnissen angewandt werden.

Directory Access Control Files

In jedem Verzeichnis können ein *Directory Access Control File* plaziert und so die Zugriffsrechte auf darin enthaltene Dateien und Unterverzeichnisse bestimmt werden. So kann ein Verzeichnis nur für bestimmte Personen oder Gruppen geöffnet oder geschlossen werden; letzteres geschieht dadurch, daß man es für alle öffnet mit Ausnahme der entsprechenden Personen oder Gruppen. *Directory Access Control Files* müssen dem Server als solche erkennbar gemacht werden. Das geschiet in der **Server-Konfigurationsdatei** mit Hilfe der Anweisung **AccessFileName**. Der dazu eingetragene Name läßt den Server erkennen, daß jede so benannte Datei ein *Directory Access Control File* für das jeweilige Verzeichnis ist, in dem es gespeichert ist, und daß darin Konfigurationsanweisungen über die Zugriffsrechte enthalten sind.

Das Plazieren von *Directory Access Control Files* im jeweiligen Verzeichnis beeinflußt nur den Zugriff auf darin enthaltene Dateien und Unterverzeichnisse. Damit kann für das jeweilige Verzeichnis genau festgelegt werden, welche Benutzer auf welche Bereiche der Document Root zugreifen können. Während des Betriebs ist es in diesem Fall nicht mehr notwendig, den Server jedesmal erneut zu starten, falls die Zugriffskonfiguration für ein bestimmtes Verzeichnis verändert wird. Bei globalen Änderungen, die in der Access-Konfigurationsdatei abgespeichert sind, ist ein Restart des Servers durchzuführen, bei dem die gesamten Konfigurationsdateien erneut gelesen werden, um die neue Konfiguration zu übernehmen.

Eintragungen in den *Directory Access Control Files* und in der *Access-Konfigurationsdatei* bezüglich des Zugriffs auf Dateien in öffentlichen Verzeichnissen des Servers betreffen folgende Daten:

- Name und Pfad des Verzeichnisses, für das Zugriffsberechtigungen eingestellt werden
- eventuell Name und Pfad der Paßwortdatei, die für die Zugriffsbestimmungen verwendet wird
- AllowOverride: kann nur in der *Access-Konfigurationsdatei* angewandt werden und bestimmt, ob für dieses Verzeichnis die lokalen Einstellungen in den *Directory Access Control Files* übernommen werden dürfen oder nicht
- IP-Adresse
- Domain-Name
- Host-Name
- Benutzer-Name
- Benutzer-Gruppe
- Typ der Berechtigung:

  deny : Diese Art der Berechtigung sperrt für jeden Benutzer oder Gruppe, die in der dieser Variablen zugeordneten Liste enthalten sind, den Zugang zu dem Verzeichnis. Falls statt einer Liste dieser Variablen das Wort "all" zugeordnet ist, dann bedeutet das, daß jedem der Zugriff auf dieses Verzeichnis verwehrt wird.

  allow : Benutzer und Gruppen, die in der Liste zu dieser Variablen aufgezählt werden, haben Zugriff auf das Verzeichnis. Auch hier ist die Zuordnung des Wortes "all" an die Variable möglich.

  order : Damit ist die Reihenfolge gemeint, in der die Listen zu den Zugriffsberechtigungen *deny* und *allow* überprüft werden:

  - deny,allow: Der Server untersucht zuerst die *deny*-Liste, dann erst die *allow*-Liste.
  - allow,deny: Der Server untersucht zuerst die *allow*-Liste, dann erst die *deny*-Liste.
  - mutual-failure: Der Server versagt jedem Benutzer und jeder Gruppe, die nicht in dieser Zugriff-Konfigurationsdatei aufgezählt sind, den Zutritt.

require : Diese Variable legt einen Paßwortschutz für das Verzeichnis fest (siehe nächsten Abschnitt über Authentifikation).

Authentifikation

Es reicht oft nicht aus, den Zugriff auf bestimmte Dateien auf Basis des Domainnamen oder der IP-Adresse des Clients zu beschränken. Hier wird eine benutzerdefinierte Zugriffsberechtigung gebraucht, wobei ein Benutzer nur mit einem gültigen Namen und dem dazugehörigen Paßwort Zugriff auf bestimmte Dateien erhält. Dies kann mit Hilfe von Paßwortdateien gelöst werden, die die Paare **Benutzername** und **Paßwort** enthalten. Falls ein Verzeichnis mit einem Paßwortschutz versehen wird, müssen in dem *Directory Access Control File* der Pfad und der Name einer gültigen Paßwortdatei eingetragen sein. So ist es nur für die darin aufgezählten Benutzer möglich, mit dem gültigen Paßwort auf das Verzeichnis zuzugreifen. Durch den Wert der Konfigurationsanweisung **AuthName** wird dem Benutzer mitgeteilt, für welchen Bereich er sich mit seinem Benutzernamen und Paßwort authentifizieren muß. Falls der Anweisung AuthName der Wert "`user/common`" zugewiesen ist, so könnte der Client des Benutzers bei dessen Zugriffsversuch auf das Verzeichnis `user/common` z.B. den Hinweis "*Bitte geben Sie Ihren Namen und Paßwort für den Bereich user/common ein.*" anzeigen.

Mit der Konfigurationsanweisung **AuthType** wird der gewünschte Authentifikationstyp festgelegt, wobei dieser entweder `Basic` oder `Digest` sein kann. Der am meisten angewendete Typ ist der Typ `Basic`, da er von den meisten Clients unterstützt wird. Wird der Typ `Digest` angegeben, so wird im Gegensatz zum Typ `Basic` das Paßwort im Falle einer gewünschten Authentifikation verschlüsselt über die Verbindung geschickt. Allerdings wird dieser Typ erst von der neuen Version 1.1 des HTTP unterstützt. Eine weitere Anweisung **Require** setzt fest, welche Benutzer aus der Paßwortdatei Zugriff auf den Bereich erhalten. Dabei kann **Require** mit drei Varianten eingesetzt werden:

1. **user**: In diesem Fall erhalten nur diejenigen Benutzer Zugang zu dem Bereich, die in der Paßwortdatei eingetragen und zusätzlich hinter dem Begriff **user** aufgezählt sind.

2. **group**: Hier erhält jeder Benutzer Zugang zu dem Bereich, der einer der hinter **group** aufgezählten Gruppen angehört.

3. **valid-user**: In diesem Fall erhält jeder in der Paßwortdatei eingetragene Benutzer Zugang zu dem Bereich.

Verschlüsselung von Daten

Die am weitesten verbreiteten Kodiermechanismen sind die Protokolle *Secure HTTP* und *Secure Socket Layer (SSL)*. Beide Protokolle übertragen alle Daten verschlüsselt, sowohl in Richtung des Servers als auch in Richtung des Clients. Beide Protokolle verwenden ein Schlüsselpaar, bestehend aus einem *public key* und einem *private key*. Mit dem *public key* des Empfängers werden die zu sendenden Daten verschlüsselt. So kann der Empfänger nur mit Hilfe seines *private key* die Daten wieder entschlüsseln. Der einzige Unterschied zwischen beiden Protokollen ist die Tatsache, daß *Secure HTTP* nur für HTTP-Transaktionen verwendet werden kann, wohingegen SSL im Zusammenhang mit jedem anderen Anwendungsprotokoll implementiert werden kann.

Zwischen Server und Clients wird bei Anwendung einer dieser beiden Protokolle ein *public key* des Servers an den Client übertragen. Dieser verwendet den Schlüssel, um seine Daten zu kodieren, und schickt sie so an den Server zurück. Der Server kann als einziger die Daten entschlüsseln, da nur er über den zum *public key* gehörenden *private key* verfügt. So kann eine sichere Transaktion gewährleistet werden. In der umgekehrten Richtung vom Client zum Server wird der gleiche Vorgang durchgeführt.

Die Kombination von HTTP und SSL ist unter der Protokollkennung *https* bekannt. Damit ist bereits aus den URLs ersichtlich, daß ein Server verschlüsselte Daten anbietet. Clients, in denen SSL nicht implementiert ist, lehnen solche URLs sofort als unbekannt ab. Für Server kann diese Funktionalität zu SSL ein- oder ausgeschaltet werden. Falls sie benutzt wird, sind folgende Arbeitsschritte durchzuführen:

- Generieren eines Schlüssel-Paares für den Server (public und private key)
- Angabe der Datei, in der das Schlüssel-Paar gespeichert ist
- Beantragen eines Zertifikats von einer Certification Authority (CA)
- Installieren des Zertifikats der CA
- Einschalten von SSL
- Angabe der Version von SSL, die der Server anwendet

Es gibt in Ergänzung zu diesen beiden Protokollen weitere Anwendungen, die eine Verschlüsselung von Daten bei der Übertragung über das Internet vornehmen. Diese verwenden im allgemeinen aber auch Schlüssel oder ähnliche Methoden und werden deshalb im Rahmen dieser Diplomarbeit nicht weiter untersucht.

4.1.5 Abrechnungsmanagement

Das Abrechnungsmanagement befaßt sich mit der Erfassung und Zuordnung von Kosten, die durch die Inanspruchnahme von Ressourcen entstehen. Dabei ist zwischen der **Abrechnungsart**, dem **abzurechnenden Dienst** und dem **Nutzer** zu unterscheiden, für den die Abrechnung abgewickelt wird (siehe dazu Abbildung 4.7). Ein wichtiger Bereich der Abrechnung im Internet findet zwischen dem Provider und seinen Kunden statt bezüglich der Nutzung des Internets. Außerdem kann eine klassifizierte Abrechnung nach Art der übertragenen Dokumente oder Anzahl der übertragenen Bytes dem Nutzer gegenüber durchgeführt werden, der einzelne Dienste auf Web-Seiten in Anspruch nimmt, wie z.B. das Laden von bestimmten Dateien von einer Seite auf den eigenen Rechner oder das Öffnen von bestimmten Seiten im Browser.

Abrechnungsart

Ein Internet-User nutzt das Internet, um Seiten von einem Server abzufragen, um Mails zu verschicken, um Dateien per ftp zu laden oder um News zu lesen. Durch das Nutzen dieser

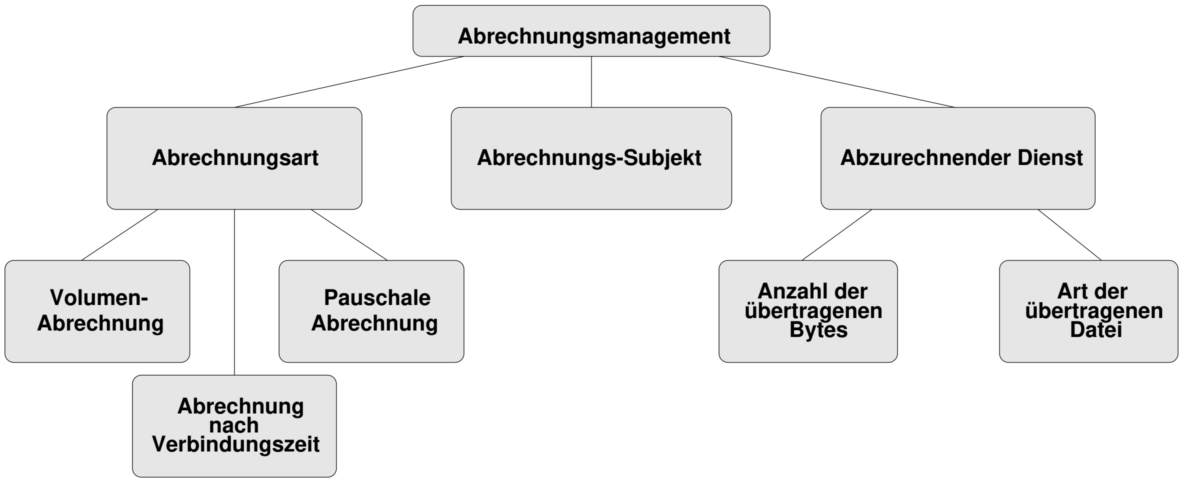

Abbildung 4.7: Abrechnung der Dienste eines Web-Servers

Dienste verursacht er einen gewissen Datenfluß, der normalerweise über das Netz eines Providers geht und den dieser seinen Kunden dann in Rechnung stellt. Weitere Kosten entstehen, wenn der Provider Web-Seiten seiner Kunden bereitstellt. Dabei steigt der Datenfluß bei jedem Zugriff eines Clients auf die Seiten der Kunden. Daneben entstehen natürlich noch Kosten für das Lagern der Dateien auf dem Server des Providers oder für das Aufstellen des Servers in den Räumen des Providers; diese Kosten werden hier nicht berücksichtigt.

Bei der Berechnung der Internet-Nutzung wendet der Provider bestimmte Abrechnungsmuster gegenüber dem Kunden an. Es gibt drei grundsätzliche Abrechnungsarten auf diesem Gebiet:

1. **Volumenorientierte Abrechnung**
   Bei diesem Abrechnungsmodell erfolgt die Berechnung des Datenflusses nach dem übertragenen Datenvolumen. Der Vorteil eines volumenorientierten Tarifs besteht darin, daß nur berechnet wird, was tatsächlich verbraucht wurde. Andererseits können in diesem Fall die Kosten nicht genau kontrolliert werden, da diese erst im nachhinein feststehen. Vor allem bei einem Web-Server ist es vorher schwer abzuschätzen, wie viele Nutzer Daten davon abrufen. Ab einer gewissen Größe des aufkommenden Datenvolumens wäre eine Anbindung des Servers an das Internet über eine Standleitung die beste Lösung. Dafür würde dann die nächste Art der Abrechnung in Frage kommen.

2. **Pauschale Abrechnung**
   Die pauschale Abrechnung findet man hauptsächlich in Zusammenhang mit Standleitungen. Hier werden für einen bestimmten Betrag pro Monat so viele Daten über die Leitung geschickt, wie die Bandbreite es erlaubt. Dabei kann der Provider eine Beschränkung auf ein bestimmtes Kontingent anbieten, um den Datenfluß zu reduzieren oder zu kontrollieren. Meist leidet hier aber die Performance der Verbindung stark darunter.

3. **Abrechnung nach Verbindungszeit**
   Diese Art der Abrechnung ist bei Wahlleitungszugängen üblich, da hier die Kosten über die Dauer der Verbindung kontrolliert werden können.

Diese Art der Kosten werden allerdings systemspezifisch über Router festgestellt. Server geben keinen genauen Aufschluß darüber, wieviel Verkehr über sie abgewickelt wurde. Sie stellen zwar Informationen darüber zur Verfügung, wie viele Bytes verschickt wurden, aber keine über den IP-Verkehr, der beim Verschicken der Datenmengen entstanden ist, der aber im Falle der oben aufgezählten Abrechnungsarten in Rechnung gestellt wird.

Anhand der Log-Dateien kann nur ungefähr der Anteil des Servers am Gesamtverkehr des Systems ermittelt werden. Die schon im Kapitel 4.1.3 im Rahmen des Performancemanagements festgestellten Informationen aus den Log-Dateien wären in diesem Fall sinnvoll:

- wie viele Zugriffe erfolgten auf die Kunden-Domäne?
- über welche Zeitspanne waren sie verteilt?
- wie viele Bytes wurden übertragen?

Dabei kann eine gezielte Abfrage nach einer Zeitspanne von einem Monat gestartet werden, je nachdem, in welchen Zeitabständen die Abrechnung erfolgt. Die Summe der in dieser Zeitspanne übertragenen Bytes gibt Aufschluß über den Bytefluß, der vom Web-Server verursacht wurde.

Abrechnungs-Subjekt

Im Zusammenhang mit der Abrechnung im Internet sind folgende Informationen interessant:

- von welchem Benutzer kamen Anfragen?
- von welchem Host kamen Anfragen?

Dabei kann es aber zu ganz trivialen Problemen kommen, falls man eine benutzerbezogene Art der Abrechnung abwickeln will, da über den Benutzer keine sicheren Angaben gesammelt werden, außer denen, die in der Log-Datei des Servers über ihn protokolliert werden. Hierbei entstehen die meisten Probleme, die gegen diese Art der Abwicklung sprechen.

In AccessLog-Dateien werden Daten über den zugreifenden Host und eventuell über den Benutzer und dessen Authentifikation protokolliert. Da die letzten beiden Angaben nicht mit Sicherheit vom zugreifenden Client mitgeschickt werden, ist es nicht immer möglich, diese zu erhalten. So kann zwar der Host eindeutig identifiziert werden, aber nicht der einzelne User. In einer Firma oder Universität, wo mehrere User am Tag an den Rechnern arbeiten, ist eine solche Identifizierung nicht ohne weiteres möglich. Hier wäre unter Umständen in Zusammenarbeit mit dem dortigen Systemverwalter ein Vergleich der Einlogzeiten der User und der mitprotokollierten Zeiten aus der AccessLog-Datei des eigenen Servers möglich, was aber nicht verallgemeinert werden kann und somit nicht in Frage kommt. Darum müssen andere Lösungen gesucht werden.

Abzurechnender Dienst

Falls der Nutzer eines Angebots im Internet eindeutig festgestellt werden kann, so kann für die angebotenen Dienste eine Abdeckung der Kosten verlangt werden. Dazu ist unbedingt entweder eine Klassifizierung der abzurechnenden Dienste nach der Art der Dokumente oder nach der Menge der übertragenen Bytes erforderlich. So ist eine Zuordnung von festgelegten

Preisen zu den entsprechenden Dokumenten notwendig oder zu der Anzahl der übertragenen Bytes. Bei der Abrechnung nach der Art des Dokuments ist speziell der Typ oder die Endung der Datei wichtig, die übertragen wurde (z.B. gif, Audio- oder Video-Daten).

Folgende Informationen können aus der **AccessLog**-Datei speziell über den abzurechnenden Dienst durch gezielte Abfrage ausgelesen werden:

- eventuell Name des Benutzers, der auf das Angebot zugreift
- wie viele Dateien wurden an den Benutzer übertragen?
- welche Art der Dateien wurden übertragen?
- Anzahl der insgesamt übertragenen Dateien einer bestimmten Art
- wie viele Bytes wurden übertragen?
- Festlegung des Preises pro übertragenem Byte
- Festlegung des Preises pro Art der übertragenen Datei

4.1.6 Fehlermanagement

Im Rahmen des Fehlermanagements ist die Überwachung des Servers und dessen Komponenten während des laufenden Betriebs erforderlich. Falls Fehler auftreten, so müssen sie lokalisiert, deren Ursachen untersucht und Aktionen gestartet werden, um sie zu beheben. Weiter schließt dieses Gebiet das Treffen von Vorsorgemaßnahmen ein, um Fehler zu vermeiden. Genauso wichtig kann das Verschicken von Fehlermeldungen an die zuständigen Personen sein, um im Falle eines Fehlers schnell handeln zu können.

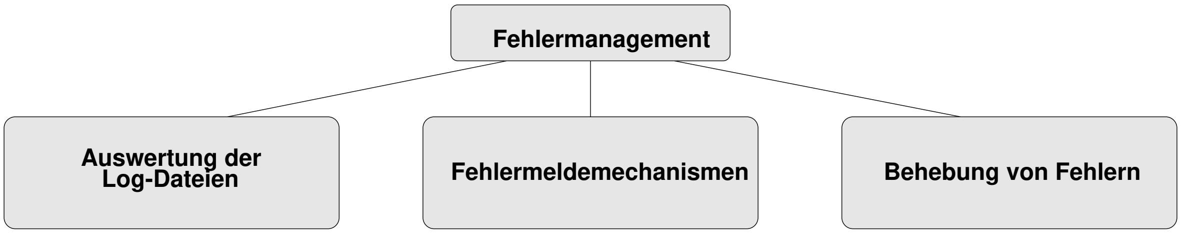

Abbildung 4.8: Fehlermanagement im Rahmen eines Web-Server

Auswertung der Log-Dateien

Jeder Server protokolliert in einer **ErrorLog**-Datei die während des Betriebs auftretenden Fehler. Dabei wird jeder fehlerhaft bearbeitete Request mit dem entsprechenden Datum und der Uhrzeit des Zugriffs aufgezeichnet, die Ressource (Datei oder Verzeichnis), auf die zugegriffen wird, der Name des Host oder die IP-Adresse des zugreifenden Host und der Grund, warum der Request fehlerhaft bearbeitet wurde. Anhand des Fehler-Grundes kann die Art des Fehlers identifiziert und weiter untersucht werden. Interne Fehler, die während des Betriebs des Servers auftreten, können z.B. aufgrund einer falschen Konfiguration oder aufgrund mangelnder Verfügbarkeit von Ressourcen auftreten. Diese erkennt man außerdem auch anhand der mitprotokollierten Werte 50X des HTTP Status Codes in der **AccessLog**-Datei. In ErrorLog-Dateien werden auch Versuche von unerlaubten Zugriffen eingetragen. Diese können im Rahmen des Sicherheitsmanagements untersucht werden. Eine weitere Log-Datei, die speziell der Apache-Server für die Auswertung der Fehler zur Verfügung stellt, ist die **ScriptLog**-Datei, in der die Ausgaben von CGI-Programmen eingetragen werden, falls deren Ausführung fehlerhaft verlaufen ist. Bei dem Netscape Enterprise Server werden diese Einträge mit in die ErrorLog-Datei eingetragen.

Neben den internen Fehlern, die den Betrieb des Servers betreffen, können auch beim Aufruf von Dokumenten Fehler auftreten. Das zeigen die Werte 30X oder 40X des HTTP Status Code in der AccessLog-Datei an. In den folgenden Unterkapiteln werden die häufigsten Werte und ihre Bedeutung im Rahmen des Fehlermanagements in aufsteigender Reihenfolge untersucht, angefangen mit den Werten über 300 (vergleiche dazu Anhang A).

Um die Effizienz und Fehleranfälligkeit eines Servers zu untersuchen, ist eine Statistik über die Anzahl der insgesamt aufgetretenen Fehler und eine Rechnung über die Anzahl der einzeln aufgetretenen Fehler unerlässlich. Dafür muß die AccessLog-Datei nach den jeweiligen Werten des HTTP Status Code gefiltert untersucht und ein Zähler für jeden einzelnen Wert eingerichtet werden, was im Rahmen des Leistungsmanagements in Kapitel 4.1.3 schon erläutert wurde.

Bei Auftreten eines Fehlers ist es notwendig, den anfragenden Client über die Art des Fehlers zu informieren. Dazu können spezielle HTML-Dateien konstruiert werden, die automatisch im Falle des Auftretens eines bestimmten Fehlers an den Client zurückgeschickt werden. Die Konfiguration dieser Option erfordert folgende Informationen:

- Status-Code des Fehlers
- Pfad der Datei, die zurückgeschickt werden soll

Somit können einzelne Anfragen nach den Werten in den Log-Dateien individuelle Informationen und Zugriffsprofile ergeben. Im speziellen Fall des `HTTP Status Code`, der vom Server für jede Anfrage registriert wird, kann die Anzahl der fehlerhaft bearbeiteten Anfragen nach Art des Fehlers untersucht und geordnet werden (siehe dazu Anhang A).

Durch Analysieren der Log-Dateien nach der Häufigkeit jedes Wertes des HTTP Status Code kann die Anzahl der richtig und falsch bearbeiteten Anfragen abhängig von der Art der Quittierung durch den Client festgestellt werden. Diese Analyse kann die Häufigkeit der aufgetretenen Fehler auf einen Blick darstellen und so das Einleiten von notwendigen Gegenmaßnahmen klarmachen. Dafür sind folgende Informationen von Interesse:

- Wert des fehlerhaften Status Code
- Name der Datei, bei deren Übertragung der Fehler aufgetreten ist
- Häufigkeit, mit der der Fehler aufgetreten ist

HTTP Status Code 30X

Der Status Code 30X zeigt an, daß der Client auf Dateien zugreift, die entweder für immer (`301 - Moved Permanently`) oder nur zeitweise (`302 - Moved Temporarily`) von der alten Adresse verlegt wurden, oder daß die gewünschte Datei seit einem gewissen Zeitpunkt nicht mehr verändert wurde (`304 - Not Modified`). Diese bedingte Anfrage muß ein Zeitfeld enthalten, mit dem das Dateidatum der gewünschten Datei verglichen wird. Die Aktionen des Server und des Clients in diesen Fällen verlaufen normalerweise im Hintergrund. Das heißt, daß sie der auf die Datei zugreifende Benutzer nicht zu sehen bekommt. Falls die Datei von dem Ort verlegt wurde, auf den der Client zugreift, schickt ihm der Server eine Antwort mit der aktuellen Adresse der Datei, worauf der Client im Hintergrund eine neue Anfrage startet. Ebenso wird im Falle der bedingten Anfrage (Status Code `304`) nur dann eine Datei zurückgeschickt, falls das vom Client mitgeschickte Datum älter ist als das aktuelle Datum der Datei.

Im Zusammenhang mit diesen Arten von Fehlern sollten Fehlermeldungen an diejenigen Administratoren gehen, die den Bereich der Link-Verwaltung (siehe Abschnitt 4.1.7) betreuen. Sie könnten `html`-Dateien gestalten, die den zugreifenden Benutzer über die Verlegung der Datei informieren und somit die Hintergrundaktivitäten des Servers und des Clients für ihn sichtbar machen.

HTTP Status Code 40X

Die Werte 40X geben im allgemeinen Fehler an, die auf seiten des anfragenden Clients erfolgt sind. Hier handelt es sich um eine falsche Anfrage (`400 - Bad Request`), falls die Adresse vom Benutzer falsch angegeben wurde. Weiter wird dem Client eine Datei nicht übertragen, falls er keine gültige Ausweisung vorzeigen kann und ihm somit der Zugriff auf das Dokument verweigert wird (`401 - Unauthorized`). Falls auf eine Datei absolut kein Zugriff gewährt wird, wird in der AccessLog-Datei der Wert `403` (`Forbidden`) für die Anfrage protokolliert. Außerdem kann eine Anfrage für eine Datei ankommen, die auf dem Server in diesem Verzeichnis nicht gefunden wird. Der dazugehörige Status Code `404` (`Not Found`) zeigt das an. Auch für den Wert `404` sollte eine Fehlermeldung an den Bereich der Link-Verwaltung gehen, um zu überprüfen, ob der Fehler beim Benutzer liegt oder bei dem Anbieter, der die Datei aus Versehen verlegt hat.

HTTP Status Code 50X

Falls der Server aufgrund von internen Fehlern einen Request nicht erfüllen konnte, so wird dazu ein entsprechender Wert über 50X in der AccessLog-Datei eingetragen. Handelt es sich um einen unerwarteten Fehler, der das richtige Ausführen und Beantworten des Requests verhindert hat, so entspricht das dem Status Code `500` (`Internal Error`). Handelt es sich um einen Dienst, den der Server nicht unterstützt, so entspricht das dem Wert `501` (`Not Implemented`). Falls es unbeabsichtigt ist, daß der Server diesen Dienst nicht unterstützt, so muß die Konfiguration des Servers untersucht und gegebenfalls geändert werden. So wird ein Server, der das automatische Indizieren von Verzeichnissen nicht unterstützt, eine Eintragung in die ErrorLog-Datei einfügen, falls ein Benutzer auf ein Verzeichnis zugreift und nicht auf eine darin enthaltene Datei.

Außerdem kann der Fall auftreten, daß ein Anwendungsprogramm, auf das der Server zuzugreifen versucht, um eine Anfrage zu beantworten, nicht in einer bestimmten Zeit ausgeführt werden konnte und somit keine Ergebnisse an den Server zurückkommen (`502 - Bad Gateway`). Der gleiche Fall tritt ein, falls ein CGI-Programm fehlerhaft ausgeführt wurde. Um hier den Fehler genauer untersuchen zu können, ist die Auswertung der ScriptLog-Datei von großer Hilfe. Wenn der Server über ungenügend Ressourcen verfügt und somit die an ihn gestellten Anfragen nicht beantworten kann, wird in der AccessLog-Datei der Wert `503` (`Service Unavailable`) zu dem Request eingetragen. In diesem Fall muß die Benutzung der Ressourcen untersucht werden, falls der Fehler über eine längere Zeit auftritt:

- Anzahl der offenen Verbindungen
- Anzahl der laufenden Prozesse
- Anzahl der freien Prozesse, die auf Anfragen warten
- Anteil der verbrauchten CPU-Zeit

Fehlermeldemechanismen

Um auf der Suche nach Fehlern eine mühsame Bearbeitung der ErrorLog-Dateien eines Servers zu umgehen, kann man sich mit Hilfe von Fehlermeldemechanismen viel Arbeit ersparen. Diese können im Falle von Auftreten von Fehlern eine bestimmte Meldung oder E-Mail an den zuständigen Bearbeiter verschicken. Sie werden meistens auf Betriebssystem-Ebene konfiguriert und sollten Aufschluß darüber geben, wo und wann der Fehler aufgetreten ist, wie er genannt wird und welche Ursache er hat, falls das festgestellt werden kann.

Beheben von Fehlern

Server-Crash wegen Fehler im Code eines Hilfsprogramms
Ein unerwarteter interner Fehler des Servers, durch den Source Code `500` (`Internal Error`) dargestellt, kann einem Serverabsturz entsprechen. Ein fehlerhaftes Hilfsprogramm, das vom Server für die Bearbeitung einer Anfrage aufgerufen wird, kann z.B. einen Serverabsturz verursachen. Dann müssen folgende Informationen abgefragt und Funktionen zur Behebung des Fehlers durchgeführt werden:

- ID des Hilfsprogramms
- Stoppen des Hilfsprogramms
- Debuggen des Hilfsprogramms
- Starten des Servers

Server kann CGI-Programme nicht ausführen
Ein konkretes Beispiel im Bereich des Fehlermanagements könnte am Beispiel von CGI-Programmen erläutert werden. Falls der Server keine CGI-Programme ausführt, kann das mehrere Gründe haben:

- Der Server ist nicht dafür konfiguriert, er unterstützt den Dienst nicht.

1. Das CGI-Verzeichnis ist in der Konfigurationsdatei nicht richtig eingetragen.
2. Das Ausführen von CGI-Programmen ist in diesem Verzeichnis nicht ausdrücklich aktiviert worden.
3. Der CGI-MIME-Typ oder -Handler ist nicht richtig konfiguriert.

- Die CGI-Version wird von dem Server nicht unterstüzt.
- Der Pfad in dem URL, mit dem das CGI-Programm aufgerufen wird, ist nicht richtig angegeben.
- Das CGI-Programm kann von einem entfernten Server nicht ausgeführt werden und schickt dem eigenen Server keine Ergebnisse zurück, so daß dieser sie nicht an den Client weiterschicken kann.

Falls der Server nicht dafür konfiguriert ist, wird jedesmal, wenn ein Benutzer ein CGI-Programm aufruft, ein Eintrag in die AccessLog-Datei eingefügt, dem der HTTP Status Code `501 (Not Implemented)` zugeordnet wird. Das gleiche passiert, falls der Server die entsprechende CGI-Version des Programms nicht unterstützt. Für den Fall, daß der Pfad vom Benutzer im URL nicht richtig angegeben wurde, wird das sowohl in der AccessLog-Datei entweder mit dem Status Code `400` (`Bad Request`) oder `404` (`Not Found`) registriert, als auch in der Error-log-Datei mit dem Fehler-Grund, daß die gesuchte Datei nicht gefunden werden konnte.

4.1.7 Link-Verwaltung

Die Tätigkeiten im Bereich der Link-Verwaltung haben stets das Ziel, ein aktuelles und in sich geschlossenes Angebot an Web-Dokumenten zur Verfügung zu stellen. Für Firmen kann es aus Marketinggründen sehr wichtig sein, daß in den öffentlich angebotenen Web-Seiten keine Lücken auftreten, die aufgrund von fehlerhaften Links oder fehlenden Dateien entstehen. Der Fall, daß zum Zugriff angebotene Dateien auf dem Server nicht gefunden werden können oder daß aktuelle Links existieren, die auf nicht vorhandene Dateien verweisen, wäre mit einem Imageverlust der Firmen verbunden.

Folgende Tätigkeiten sind im Rahmen der Verwaltung der Dokumentstruktur eines Servers erforderlich:

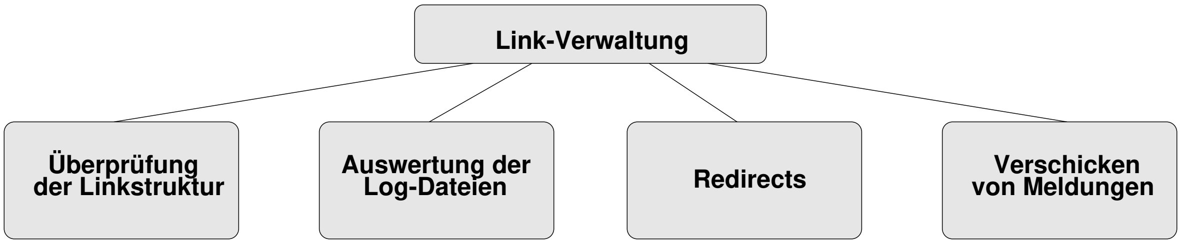

Abbildung 4.9: Link-Verwaltung innerhalb eines Web-Servers

Die permanente Überprüfung der Linkstruktur eines Servers oder innerhalb einer Domäne ist sehr wichtig. Fehlerhafte Links müssen entdeckt und entfernt werden. Falls eine Datei entfernt wird, auf die von irgendwoher verwiesen wird, so ist es wichtig, diese Links zu kennen und sie in den entsprechenden Dateien zu ändern oder zu entfernen. Auch der Server kann mit Hilfe von Redirects (siehe dazu Abschnitt 4.1.2 zu der Ressourcen-Konfiguration) so konfiguriert werden, daß er im Falle eines Zugriffs auf verschobene Dateien die neue Location an den Client schickt.

Wird eine Datei vom Server entfernt, so können an die Betreuer der Dateien, die in der Liste der Vater-Links aufgezählt sind, Meldungen verschickt werden, daß diese Datei nicht mehr auf dem Server vorhanden ist. Das wäre eine präventive Maßnahme, um Aktualitätslücken vorzubeugen. Vielleicht wäre es sinnvoll, das Umkopieren von Dateien in andere Verzeichnisse innerhalb der Document Root mit der Generierung einer damit verbundenen Meldung an den zuständigen Betreuer dieser Datei zu verknüpfen, was allerdings auch als störend empfunden werden kann, falls nur Dateien ersetzt oder erneuert und deswegen in andere Verzeichnisse umkopiert werden.

Eine andere Möglichkeit, fehlende Dateien auf dem Server zu entdecken, wäre die Auswertung der Log-Dateien. Anhand der entsprechenden Fehlercodes zu den einzelnen Dateien kann auf die Ursache der fehlerhaften Links geschlossen werden. Falls nicht-gefundene Dateien nur an einen anderen Ort kopiert wurden, so sollte das entweder dem Client mitgeteilt werden, oder dem Betreuer derjenigen Datei, die den fehlerhaften Link enthält, um diesen zu aktualisieren, oder als dritte Möglichkeit, den Server mit Hilfe der Redirects entsprechend zu konfigurieren. Falls eine Datei gar nicht gefunden wird, so kann diese entweder nicht mehr vorhanden sein oder der Link mag falsch geschrieben sein. Auf jeden Fall ist der Betreuer der Datei, die den falschen Link enthält, zu benachrichtigen.

Um die Dokumente auf dem Server oder innerhalb einer Domäne angemessen verwalten zu können, sollten zu jeder einzelnen Datei folgende Daten abrufbar sein:

- Links, die auf diese Datei verweisen (Väter)
- Links, die von dieser Datei ausgehen (Söhne)
- Links nach außerhalb der Domäne, die von dieser Datei ausgehen
- Betreuer, der für diese Datei zuständig ist

Mit Hilfe von Auswertungstools können die Links überprüft und als aktuell oder fehlerhaft gekennzeichnet werden. Für jeden fehlerhaften Link kann dann der Name der Datei und des Betreuers entnommen werden, der für den Inhalt und deshalb für die darin enthaltenen Links dieser Datei zuständig ist. Solch ein Auswertungstool wurde am Lehrstuhl im Rahmen zweier Forschungspraktika entwickelt (siehe dazu [ES96] und [Sch96]).

4.2 Proxy-Server

Ein Proxy-Server übernimmt gegenüber mit ihm agierenden Komponenten sowohl die Rolle eines Clients als auch die eines Servers (siehe Abbildung 4.10). Wenn ein Client so konfiguriert ist, daß er einen Proxy benutzt, so werden die von ihm verschickten Anfragen nicht direkt an die in den URLs genannten Server gesendet, sondern erst an den Proxy. Dieser übernimmt dann die Anfragen und sendet sie an die entsprechenden Server weiter. Die Antworten gehen den gleichen Weg über den Proxy zurück. Somit übernimmt ein Proxy gegenüber einem Client die Rolle des Servers und gegenüber einem Server die Rolle des Clients.

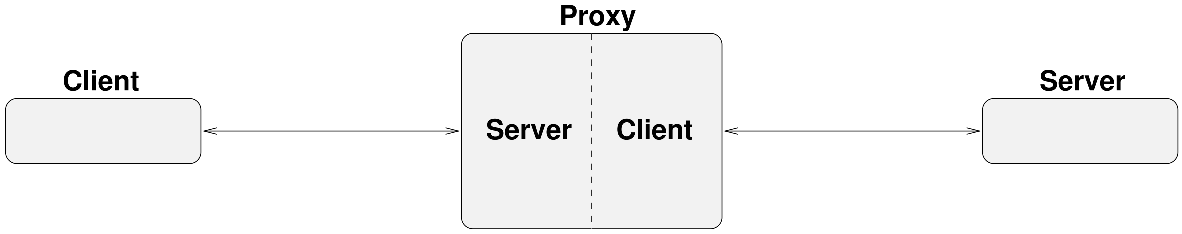

Abbildung 4.10: Rollenverteilung eines Proxy-Servers

Proxies sind Server mit spezifischen Funktionalitäten. Falls ein Client seine Anfragen über einen Proxy verschickt, so sendet er nicht nur den relativen Pfad eines URL mit der Anfrage (z.B. `GET /verzeichnis/datei.html`), sondern den gesamten URL samt Protokoll- und Serverangabe (`GET http://www.server.com/verzeichnis/datei.html`). Ein als Proxy konfigurierter Server versteht diese URLs und kann sie bearbeiten. Er muß lediglich so konfiguriert werden, daß er Anfragen, die mit einer Protokollangabe starten, erkennt und an die entsprechenden Server weiterleitet. Falls ein Web-Server zusätzlich als Proxy konfiguriert wird, so kann er normale Anfragen, die nicht mit einer Protokollangabe starten, weiterhin in der gewohnten Art und Weise bearbeiten, während er Anfragen mit Protokollangabe als Proxy-Anfragen erkennt. So kann ein Server sowohl als Web-Server als auch als Proxy handeln. Zusätzlich kann ein Proxy noch als Caching-Proxy konfiguriert werden, so daß er die Dateien in einem Cache speichert, die er als Anfragen bearbeitet und an Clients zurückgeschickt hat. Greift später ein Client auf die gleichen Dateien, so liefert er die gespeicherten Kopien aus dem Cache und muß nicht eine erneute Anfrage an den Web-Server starten. Es ist sehr wichtig, zwischen einem reinen Proxy und einem Caching-Proxy zu unterscheiden, da das Caching eine Erweiterung des Proxy-Servers ist und nicht jeder Proxy gleich ein Caching-Proxy ist.

Im Rahmen dieser Diplomarbeit werden speziell Proxy-Server betrachtet, die als Web-Server installiert und mit erweiterten Funktionalitäten als Proxy konfiguriert werden können. Um einen Proxy zu betreiben, muß der Schwerpunkt auf die Konfiguration eines Web-Servers gelegt werden, da so die gesamten verfügbaren Funktionen eines Proxy eingestellt werden können. Der Schritt der Installation kann in diesem Fall also übersprungen werden, da sie identisch mit der Installation eines Web-Servers ist (siehe Abschnitt 4.1.1). Auch in den Bereichen des Leistungs-, Fehler- und Abrechnungsmanagements unterscheiden sich Proxies nicht von Web-Servern. Ebenso kann der Bereich der Link-Verwaltung für einen Proxy nicht in Frage kommen, da dieser im Sinne eines Web-Servers keine Dokumente zur Verfügung stellt, die einer eingehenden Ordnung und Verwaltung bedürfen. Ein Proxy ist *bloß* ein Vermittler von Dokumenten zwischen einem Client und anderen Servern oder Proxies.

4.2.1 Konfiguration

Die Konfiguration eines Proxy-Servers erfolgt im Rahmen der Grundkonfiguration eines Web-Servers als Ergänzung in der **Server-Konfigurationsdatei**. Um den Web-Server als Proxy zu aktivieren, muß ihm nur mittels einer **Pass**-Anweisung mitgeteilt werden, welche Protokolle er verstehen soll.

Will man ihn als Caching-Proxy initialisieren, so muß die Caching-Funktionalität mittels einer **Caching**-Anweisung ein- oder ausgeschaltet werden. Dazu muß das **CacheRoot**-Verzeichnis und die maximale Größe der Datenmenge, die in diesem Verzeichnis abgelegt wird, mittels einer **CacheSize**-Anweisung festgelegt werden. Wird diese Obergrenze an Daten in der CacheRoot überschritten, so kann der belegte Speicher immer wieder auf diese Mengenangabe angeglichen werden. In festgelegten Intervallen, die mittels einer Konfigurationsanweisung **CacheInterval** meistens in Stunden angegeben werden können, soll eine Überprüfung gestartet werden, nach der alle Daten, die über die festgelegte Speichergröße hinaus abgespeichert sind, gelöscht werden. Dieser Vorgang wird als Garbage Collection bezeichnet und wird gebraucht, um nicht den gesamten zur Verfügung stehenden Speicher zu überfüllen.

Wenn Caching-Proxies verwendet werden, um Dateien zu speichern und sie ohne neuen Zugriff auf den Original-Server den Benutzern zur Verfügung zu stellen, stellt sich die dringende Frage, ob die Daten noch aktuell sind, wenn sie den Benutzern übertragen werden. Darum kann mit Hilfe einiger Anweisungen festlegt werden, wann der Caching-Proxy Dateien als gültig beziehungsweise aktuell betrachtet, ohne einen erneuten Zugriff auf den Original-Server zu starten. Normalerweise legt ein Proxy anhand der HTTP-Header `Expires` und `Last-Modified` das Ablaufdatum einer Datei fest. Findet er keinen der beiden Header in der Antwort, so gibt er die erhaltene Datei nur an den Client weiter, ohne sie zwischenzuspeichern. Manche Proxies erlauben das Setzen einer **CacheDefaultExpire**-Variable, mit der die Zeit angegeben wird, die für Dateien angewendet wird, die entweder ohne die beiden HTTP-Header vom Server geschickt oder mit einem Protokoll übertragen werden, das keine Kennzeichnung einer Gültigkeitsdauer kennt. Die Anweisung **CacheMaxExpire** legt eine maximale Zeitgrenze fest, wie viele Stunden eine Datei als aktuell betrachtet werden soll. Nach Ablauf dieser Zeitspanne wird bei einem Zugriff auf die Datei ein erneuter Zugriff auf den Original-Server gestartet, davor wird die Kopie aus dem Proxy-Cache zurückgeliefert.

Außerdem kann mittels einer **NoCache**-Anweisung festgelegt werden, welche Dateien vom Caching-Proxy nicht zwischengespeichert werden sollen. Das ist vor allem für die Dateien sinnvoll, die aus der eigenen Domäne abgefragt werden. Ebenso ist der umgekehrte Fall sinnvoll, daß angegeben werden kann, aus welchen Domänen ausschließlich Dateien zwischengespeichert werden sollen.

Proxy-Server können auch verkettet werden, so daß entweder abhängig vom verwendeten Protokoll im Rahmen einer eingehenden Anfrage oder abhängig vom Host, an den die Anfrage gerichtet ist, die Anfragen an einen anderen Proxy-Server weitergeleitet werden. Beim Apache-Server kann mit Hilfe der **ProxyRemote**-Anweisung für jeden gewünschten Host ein Proxy angegeben werden, an den die Anfragen für diesen Host weitergeleitet werden. Die gleiche Anweisung gilt auch für sämtliche Protokollangaben, falls Anfragen abhängig vom benutzten Protokoll an andere Proxies weitergeleitet werden sollen. Die Vorteile von verketteten Proxies liegen darin, daß angeforderte Dokumente teilweise schneller geliefert werden dadurch, daß nicht auf die Original-Server zugegriffen wird oder auch der entfernte Verkehr über das weltweite Internet um einiges reduziert werden kann, falls viele Anfragen auf gleiche Seiten erfolgen, die schon im Cache liegen. Ein weiterer Vorteil entsteht daraus, daß Anfragen auf bestimmte Wege *geroutet* werden können, so daß dadurch je nach Bedarf z.B. ein kostengünstigerer oder schnellerer Weg gewählt werden kann.

Weitere Konfigurationsmöglichkeiten eines Proxy-Servers sind von jedem Server einzeln abhängig und können in den jeweiligen Dokumentationen eingesehen werden.

4.2.2 Sicherheitsmanagement

Im Rahmen des Sicherheitsmanagements eines Proxy-Servers muß der Personenkreis eingegrenzt werden, der Zugriff auf den Server erhält. Dafür stehen die gleichen Konfigurationsanweisungen zur Verfügung, die auch für das Regeln der Zugriffskontrolle auf einen Web-Server gelten. Sie können in der Haupt-Access-Konfigurationsdatei des Web-Servers eingetragen werden oder in eigenen *Directory Access Control Files*, die sich auf die CacheRoot beziehen. Es kann auch eine eigene Access-Konfigurationsdatei für den Proxy definiert werden, in der dann globale Einstellungen eingetragen sind.

Zusätzlich zu den Sicherheitseinstellungen des Web-Servers können auch die ausgehenden Anfragen kontrolliert werden. Der Zugriff auf gesamte Domänen kann über einen Proxy gesperrt werden. Erhält der Proxy eine Anfrage, die an eine ”verbotene” Domäne gerichtet ist, so wird kein Zugriff auf den gewünschten Server gestartet. Es können (meistens abhängig vom Proxy-Server) Zugriffe auf bestimmte Domänen, IP-Adressen oder Sites abgefangen werden. Der Apache-Server erlaubt auch die Angabe eines Wortes, das nicht im Namen des Host einer angeforderten Seite enthalten sein darf.

4.3 Clients

Im Rahmen des Managements von Clients ist deren Installation und Verteilung in großen Systemumgebungen und Netzen und damit verbunden auch deren Konfiguration von großer Bedeutung. Eine überwachende Funktion haben die Aufgaben im Rahmen des Performance- und Leistungsmanagements, da Aussagen über die Auslastung der Clients gewonnen werden können. Aufgaben des Sicherheitsmanagements entfallen im Zusammenhang mit den Clients, da diese dafür meistens keine Funktionalität zur Verfügung stellen. Die Aufgaben im Rahmen des Fehlermanagements erstrecken sich auf Ausbesserungen von Fehlern, die während der Installation oder Konfiguration begangen wurden. Deshalb wird auf deren nähere Erläuterung verzichtet.

4.3.1 Installation und Verteilung

Die Installation eines Clients bedarf normalerweise keiner größeren Aufmerksamkeit, falls es sich um einzelne Clients handelt, die noch per Hand installiert werden können. Anders ist es aber in größeren Unternehmen, wenn eine beträchtliche Anzahl an Clients verteilt auf einzelne Rechner unternehmensweit installiert werden muß. Dabei muß vor allem auf die richtige Verteilung der Software-Komponenten geachtet werden, so daß jeder Benutzer die entsprechenden Konfigurationsdateien korrekt erhält und sie behandeln kann, um persönliche Präferenzen einzutragen.

Bei der Installation eines Clients, egal ob auf einen Rechner oder auf mehreren in einer verteilten Umgebung, muß das **Zielverzeichnis** angegeben werden, in dem die Dateien plaziert werden sollen, die der Client während des Betriebs benötigt. Die meisten Clients benötigen eine **Preferences**-Datei, in der die Konfigurationseinstellungen des Clients eingetragen sind. Diese werden normalerweise von dem Systemadministrator vorab eingestellt, wobei sie vom jeweiligen Benutzer überschrieben werden können, um den Client nach eigenen Belieben zu konfigurieren. Eine sinnvolle Anweisung, die der Administrator einstellen sollte, ist z.B. die Angabe einer Proxy-Konfigurationsdatei, in der Einstellungen über die Benutzung der Proxies im System vorkonfiguriert sind. Bei der Überspielung einer neuen Version des Clients kann die Preferences-Datei überschrieben werden.

Eine andere Datei, die mit der Installation des Clients verteilt werden kann, ist eine **Bookmarks**-Datei, in der Web-Adressen gespeichert sind, die ein Benutzer öfter braucht und die als Lesezeichen in dieser Datei abgespeichert werden können, um sie nicht jedesmal neu eingeben zu müssen. Ein Unternehmen kann dadurch seinen Mitarbeitern gleich ein paar wichtige Web-Adressen mitteilen, wie z.B. die Adresse der Intranet-Seiten oder der Tochterunternehmen. Existiert nach der Installation eines Clients noch keine Bookmarks-Datei auf dem Benutzer-System, so wird sie von den meisten Browsern, die Bookmarks unterstützen, gleich angelegt, sobald das erste Bookmark abgespeichert wird.

4.3.2 Konfiguration

Wie bereits erwähnt, können Clients normalerweise schon vorab mit Hilfe einer Konfigurationsdatei konfiguriert werden oder erst nach der Installation von den jeweilgen Benutzern. Da Clients der Firma Netscape die am weitesten verbreiteten und genutzten Browser sind, sollen einige Konfigurationsmöglichkeiten an deren Beispiel erläutert werden.

Neben den allgemeinen Einstellungen, die die **Erscheinung der Browser-Oberfläche** des Clients beeinflussen und auf die hier im Rahmen der Diplomarbeit nicht näher eingegangen wird, gibt es die technischen Einstellungen, die den **Betrieb und die Umgebung des Clients** bestimmen. Eine wichtige Funktionalität diesbezüglich betrifft die MIME-Typen, die ein Client unterstützt. Wie im Kapitel über Server schon erklärt wurde, benötigt ein Client die Angabe des MIME-Typs einer Datei vom Server, um zu wissen, wie er das erhaltene Dokument weiterverarbeiten soll. Dazu führt ein Browser eine **MIME-Datei**, in der jeder MIME-Typ mit der dazugehörigen Anwendung aufgezählt ist, die für das Bearbeiten der Datei aufgerufen wird. Dabei kann systemweit eine Datei mit den häufigsten Typen eingerichtet werden, die von jedem Browser als Default angewandt wird, und zusätzlich eine benutzerspezifische, in die jeder weitere Benutzer MIME-Typen eintragen kann, die seinen Bedürfnissen entsprechen. Diese wird vom Browser nur dann erzeugt, falls der Benutzer eigene MIME-Typen definiert.

Eine weitere wichtige Datei, die der Browser während des Betriebs erstellt und benötigt, ist die **History**-Datei, in der die Adresse jeder besuchten Seite eingetragen wird. Der Browser benutzt die darin enthaltenen Daten, um dem Benutzer mit Hilfe der `Back`- und `Forward`-Buttons eine Navigation durch bereits besuchte Seiten zu ermöglichen, ohne deren Adressen neu eingeben zu müssen.

Nicht nur Proxy-Server unterstützen das **Caching von Dateien**, sondern auch Browser selbst, um auf bereits besuchte Dateien schneller zugreifen zu können. Dadurch wird die History-Navigation erst angenehm und schneller, da nicht jedesmal eine erneute Anfrage an den Server oder Proxy-Server gestartet, sondern die (meistens) vor ein paar Minuten geladene Seite aus dem Browser-eigenen Cache geladen wird. Will der Benutzer eine Seite neu laden und nicht die Kopie aus dem Cache, so muß er das im Browser ausdrücklich tun. Der **Browser-Cache** kann folgendermaßen konfiguriert werden:

- Die Größe des **Memory-** und des **Disk Cache** muß eingetragen werden.
- Das **Cache Directory** gibt das Disk-Cache Verzeichnis an, in dem Dateien lokal zwischengespeichert werden.
- Es kann meistens eine **Zeit** angegeben werden, wie oft ein Dokument aus dem Cache überprüft werden soll, bevor es neu von einem Server geladen wird. Dabei gibt es beim Netscape-Browser die drei Möglichkeiten, daß ein Dokument einmal während des Betriebs des Browsers aktualisiert werden soll, jedesmal dann, wenn eine Seite in den Browser geladen wird, was aber als nicht im Sinne dieser Einrichtung gesehen werden kann, oder nie, so daß ein Dokument, das sich im Cache befindet, nie neu aufgerufen wird, nur bei ausdrücklicher Anforderung danach.
- Der **Cache-Inhalt** kann auch bei Bedarf jedesmal **gelöscht** werden.

Weitere Konfigurationsanweisungen können bezüglich der **Proxies** eingestellt werden. Ein Browser sendet ja dann seine Anfragen nicht mehr direkt an den Ziel-Server nur durch Angabe des Pfades der gewünschten Datei im HTTP-Header, sondern schickt den gesamten URL an den Proxy, der die Anfrage weiter bearbeitet (siehe dazu Abschnitt 4.2 zu Proxies). Einem Browser muß ausdrücklich mitgeteilt werden, daß er die Anfragen über Proxies verschicken muß. Folgende Einstellungen können im Netscape-Browser zu den Proxies vorgenommen werden:

- Es kann festgelegt werden, daß **kein Proxy** verwendet werden soll.
- Der Proxy kann abhängig von dem verwendeten Protokoll der Anfrage ausgewählt werden. Dazu ist eine **manuelle Einstellung** des Proxy möglich, indem zu jedem Protokoll (http, ftp, telnet usw.) der Name oder die IP-Adresse des Proxy und der dazugehörige Port eingetragen wird.
- Andererseits kann eine **automatische Einstellung** der Proxies vorgenommen werden, indem eine Proxy-Konfigurationsdatei angegeben wird, in der die notwendigen Eintragungen enthalten sind.

Zu den **Netzverbindungen**, die ein Client während des Betrieb öffnen kann, können im Netscape-Browser folgende Einstellungen vorgenommen werden:

- Die **maximale Anzahl an Verbindungen**, die gleichzeitig geöffnet werden dürfen, kann festgelegt werden. Ein Client kann gleichzeitig mehrere Verbindungen zu Servern aufbauen, was aber die Performance jeder einzelnen einschränken kann und deswegen ist die obere Zahl nicht zu groß zu wählen.
- Die **Größe des Puffers**, in dem Daten einer Netzverbindung abgelegt werden, gibt an, wie viele Daten während einer Verbindung empfangen werden können. Große Puffer bedeuten mehr Daten, können den Rechner aber auch überfüllen.

Der Netscape-Browser erlaubt dem Benutzer weitere spezifische Konfigurationseinstellungen, die nicht von jedem Browser unterstüzt und deshalb hier nur als Beispiel aufgeführt werden:

- Das Aktivieren bzw. Deaktivieren der Unterstützung für Java-Programme: **Enable/Disable Java**

- bzw. für JavaScript-Programme: **Enable/Disable JavaScript** kann benutzerspezifisch konfiguriert werden.
- Jeder Benutzer kann eine **Startseite** einstellen, auf die stets beim Aufruf des Browsers zugegriffen wird.
- Das **automatische Laden von Bildern** beim Zugriff auf Web-Seiten kann ein- und ausgeschaltet werden, wobei Bilder jederzeit während des Betriebs des Browsers nachgeladen werden können, falls das automatische Laden deaktiviert ist.

Die meisten neueren Browser unterstützen auch noch zusätzliche Funktionalitäten, wie z.B. das Lesen und Schreiben von Mails oder News. Dafür müssen entsprechende Einstellungen vorgenommen werden, wie die Angabe der Mail- oder News-Server oder der Benutzerkennung und des Paßwortes, was aber im Zusammenhang mit dem Dienst WWW nicht weiter verfolgt wird.

4.3.3 Leistungsmanagement

Die Auslastung eines Clients kann in dem Sinn interessant sein, daß untersucht wird, wie viele Anfragen der Client in einem bestimmten Zeitintervall abschickt, wie viele Anfragen an die gleichen Server gehen oder wie oft die gleichen Dateien angefordert werden. Clients stellen dazu keine direkten Daten und Informationen zur Verfügung wie Server das im Rahmen der Log-Dateien tun. Gezielte Untersuchungen wären z.B. durch geeignetes Bearbeiten der History-Liste möglich. Dabei könnten folgende Zähler Aufschluß über die Auslastung des Clients geben:

- Anzahl der gesamten Anfragen, die der Client abgeschickt hat
- Anzahl der Anfragen, die der Client abgeschickt hat und für die eine neue Verbindung aufgebaut wurde, bei denen die Seite also nicht aus dem Cache geladen wurde
- Anzahl der Anfragen, die an bestimmte Server geschickt wurden
- Anzahl der Anfragen, die auf eine bestimmte Seite zugegriffen haben
- Anzahl der richtig bearbeiteten Anfragen, für die eine Antwort kam

Weitere Informationen über die Auslastung des Clients können gleichzeitig Aufschluß über das System geben, in dem der Client ausgeführt wird. Der Zeitunterschied vom Abschicken einer Anfrage bis zur Ankunft der Antwort kann z.B. ein Hinweis darauf sein, ob das Netz, über das die Anfragen abgeschickt werden, sehr ausgelastet ist oder nicht, falls das Zeitintervall groß ist. Hierfür soll mit den Möglichkeiten des integrierten Managements festgestellt werden können, ob solche Probleme am Netz, am System oder an der Anwendung selbst liegen. Normalerweise liefert ein Client Fehlermeldungen dazu, falls der Fehler auf der Seite des Servers liegt. Ist das nicht der Fall, so müssen systemspezifische Informationen untersucht werden, um der Ursache auf den Grund zu gehen. Liegt das am Client, so kann eine falsche Konfiguration des Clients der Grund dafür sein oder die Tatsache, daß der Client das angeforderte Dokument nicht unterstützt, dessen Format nicht versteht, oder keine Hilfsanwendung dafür definiert hat, um es weiterbearbeiten und entsprechend darstellen zu können. Werden keine Anfragen abgeschickt, so kann ein fehlender Eintrag in der Proxy-Konfiguration der Grund dafür sein, da das Netz so konfiguriert ist, über das die Anfragen abgeschickt werden. Das ist z.B. dann der

Fall, wenn ein Unternehmen ausgehende Anfragen aus dem internen Netz an außenstehende Server nur über einen Proxy-Server zuläßt.

4.4 Zusammenfassung

Die Behebung von Fehlern, die während des Betriebs von Servern, Proxies oder Clients auftreten, erfordert den Einsatz von Mitteln des integrierten Managements. Ebenso wird die Installation und Konfiguration dieser Komponenten dadurch erleichtert, daß einheitliche Werkzeuge die Vielfalt der möglichen Szenarien reduzieren sollen. Die Untersuchung der Auslastung der Komponenten und der während des Betriebs notwendigen System- und Anwendungsressourcen kann durch Sammeln sowohl von system- als auch von anwendungsspezifischen Daten vereinheitlicht und erleichtert werden. Auch Aufgaben im Rahmen des Sicherheits- und Abrechnungsmanagements für die Komponenten des Dienstes WWW erfordern unbedingt sowohl system- als auch anwendungsspezifische Informationen und Funktionen, so daß zusammengefaßt gesagt werden kann, daß im Hinblick auf das integrierte Management jeder Aspekt der Anwendung selbst und der Umgebung, in der sie läuft, untersucht und überwacht werden muß.

Kapitel 5

Entwicklung eines Objektmodells

5.1 Top-Down-Modellierung

Um ein integriertes Management von möglichst vielen verteilten Anwendungen realisieren zu können, müssen gemeinsame, allgemeine Merkmale und Eigenschaften definiert werden, die eine große Menge verteilter Anwendungen charakterisieren und dennoch ein effizientes Management erlauben. Diese generischen Eigenschaften werden in einem objektorientierten Informationsmodell in Klassen zusammengefaßt, die allgemein verteilte Anwendungen definieren. Im Rahmen dieser Diplomarbeit werden die generischen Eigenschaften verteilter Anwendungen in Anlehnung an das Referenzmodell für Open Distributed Processing definiert. Dieses Referenzmodell gibt einen Rahmen vor, der das Entwickeln von verteilten Anwendungen in offenen, verteilten, heterogenen Systemumgebungen erlaubt und somit die allgemeinen Komponenten und Merkmale von verteilten Anwendungen definiert, von deren Entwicklung bis hin zur Laufzeit und zum Betrieb der Anwendung. So können generische Basisklassen für das Management von verteilten Anwendungen gewonnen werden.

Diese allgemeinen Eigenschaften können dann für konkrete Anwendungen durch weitere Attribute, die anwendungsspezifische Merkmale definieren, spezifiziert werden. Oberklassen werden durch Verfeinerung der generischen Attribute und Hinzufügen neuer, anwendungsspezifischer Attribute in Unterklassen aufgeteilt. Die Unterklassen erben jede Information der Oberklassen, von denen sie abgeleitet werden. So kann von allgemeinen Informationen über verteilte Anwendungen in oberen Ebenen des Objektmodells durch eine immer genauere Unterteilung und Verfeinerung von Oberklassen mit jeder nach unten neu hinzugefügten Ebene von Unterklassen eine konkretere Art von verteilter Anwendung spezifiziert werden.

Wird z.B. eine Oberklasse für Server eingeführt, müssen für jede Art von Server geltende Eigenschaften gefunden werden. Diese Klasse kann abhängig vom Typ des Servers (ob z.B. ftp-Server, NFS-Server oder WWW-Server) in Unterklassen aufgeteilt werden, für die weitere, für die Art von Server typische Eigenschaften definiert werden. Diese neuen Unterklassen können z.B. abhängig vom Hersteller des Servers weiter spezifiziert werden, indem herstellerspezifische Merkmale für jede Unterklasse dieses Servertyps hinzugefügt werden. WWW-Server der Firma Netscape haben andere Eigenschaften als WWW-Server der Firma Microsoft, wobei WWW-Server auch über zahlreiche gemeinsame, herstellerunabhängige Eigenschaften verfügen. Anhand dieses Beispiels würde eine Oberklasse *Server* in eine Unterklasse *WWW-Server* spezialisiert werden, die wiederum weiter in die Unterklassen *Microsoft-WWW-Server* bzw. *Netscape-WWW-Server* spezialisiert werden kann. Die Spezialisierung kann aufgrund beliebiger Merkmale erfolgen, die sinnvoll gewählt werden müssen.

Im Rahmen dieser Diplomarbeit wird ein schon bestehendes Objektmodell verwendet, das zum Management von verteilten Systemdiensten, und zwar der speziellen Dienste NFS und NIS, am Lehrstuhl entwickelt wurde. Das Objektmodell wird von Tobias Müller in seiner Diplomarbeit ausführlich beschrieben (siehe [Mue98]) und soll hier nur grundlegend für die Bereiche erklärt werden, die für das Management des Dienstes WWW relevant sind. Als Aufgabe dieser Diplomarbeit soll speziell der WWW-Dienst analysiert und in das bestehende Objektmodell eingefügt werden.

5.1.1 Generische Management-Objektklassen

Wie oben schon erwähnt, sollen anhand des ODP-Referenzmodells generische Klassen für das integrierte Management von verteilten Anwendungen entwickelt werden. Mit Hilfe der Konzepte des Computational Viewpoints werden die Oberklassen definiert, die die funktionale Dekomposition einer verteilten Anwendung in Komponenten erlauben. Da der Engineering Viewpoint die Infrastruktur analysiert, mit deren Hilfe die Verteilung einer Anwendung realisiert wird, die aber für die Realisierung des Dienstes WWW keine besonderen anwendungsspezifischen Voraussetzungen erfüllen muß, wird dieser hier nur ansatzweise betrachtet.

Computational Viewpoint

Im folgenden Abschnitt werden die Basisklassen *Computational Object*, *Computational Object Template*, *Computational Interface*, *Computational Interface Template* und *Interaction Info* des Computational Viewpoints beschrieben.

Computational Object

Wie in Kapitel 3.2.1 schon erläutert, besteht eine verteilte Anwendung aus Komponenten, den *Computational Objects*, die an Schnittstellen miteinander kommunizieren. Wird eine Management-Schnittstelle für ein Computational Object implementiert, so kann eine Managementanwendung über diese Schnittstelle Managementinformation über das Objekt abfragen oder auch steuernd darauf einwirken. Agenten einer verteilten Managementanwendung können auch als Computational Objects gesehen werden, die mit den zu steuernden und zu überwachenden Objekten Informationen austauschen oder sie über Operationen beeinflussen.

Im Objektmodell für das Management von verteilten Systemen wird die Klasse **compObject** des Computational Object von der Oberklasse **object** abgeleitet (siehe Abbildung 5.1). Da grundsätzlich jedes Objekt über einen eindeutigen Identifikator verfügt, enthält die Oberklasse **object** das Attribut `ID`. Die Methoden `create()` und `delete()` bezeichnen die Aktionen, mit denen ein Objekt erzeugt und gelöscht werden kann.

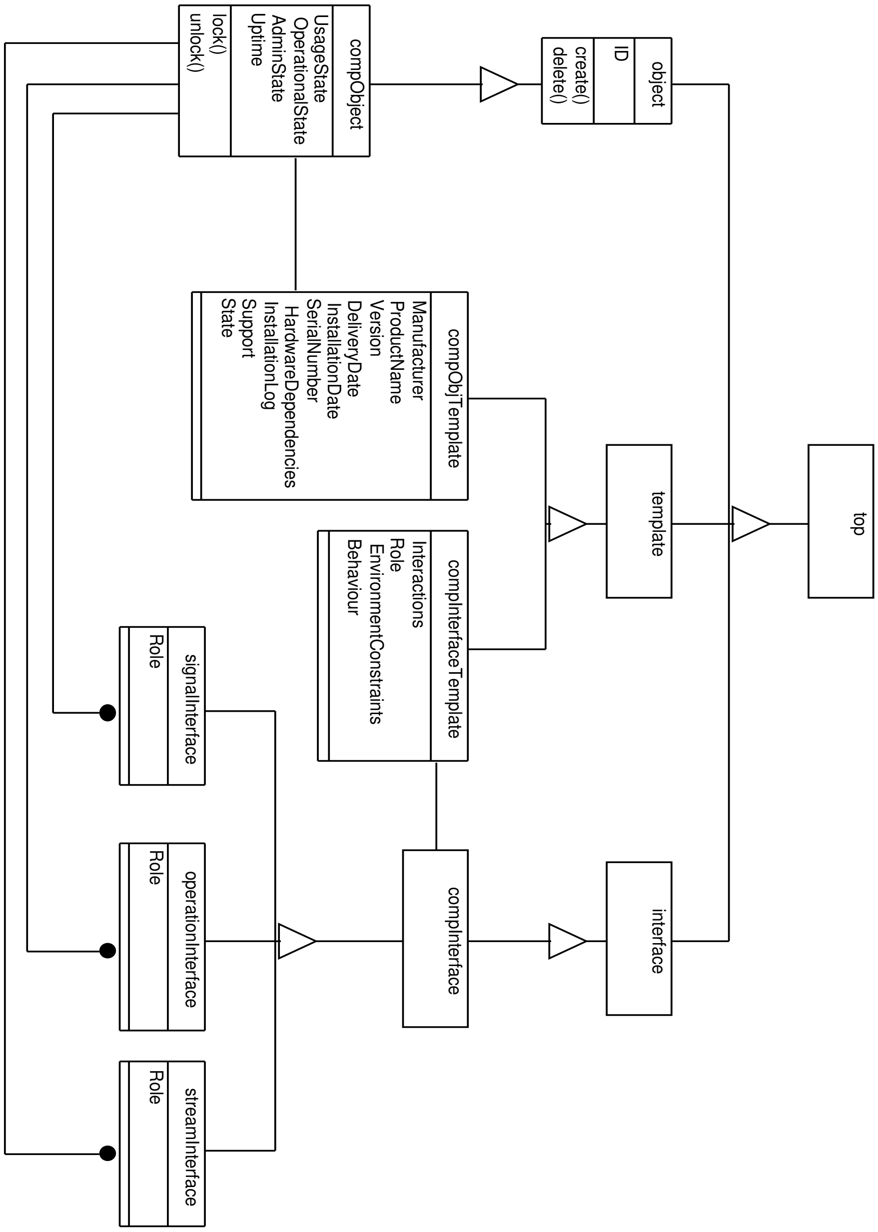

Abbildung 5.1: Generische Objektklassen des Computational Viewpoint

Jedes Computational Object befindet sich in einem Zustand, der darüber Auskunft gibt, ob das Objekt in Benutzung ist, ob es betriebsbereit ist oder nicht, oder ob es für die Benutzung durch andere Objekte gesperrt ist oder nicht. Der jeweilige Zustand kann aus den Attributen `UsageState`, `OperationState` und `AdminState` ausgelesen werden. Außerdem wird in dem Attribut `Uptime` der Zeitpunkt festgehalten, an dem die Instantiierung dieser Software-Komponente einer verteilten Anwendung erfolgt. Die Methoden `lock()` und `unlock()` erlauben das Sperren bzw. Entsperren der Dienste eines Objekts für die Benutzung durch andere Objekte.

Computational Object Template

Templates legen die Eigenschaften von Computational Objects und Computational Interfaces fest. Ein *Computational Object Template* spezifiziert alle für die Instantiierung eines Objektes erforderlichen Eigenschaften. Die dazugehörige Klasse **compObjTemplate** legt durch Attribute bestimmte Informationen fest, die z.B. bei der Installation einer Software-Komponente von Bedeutung sind. Sie wird von der Oberklasse **template** abgeleitet (siehe Abbildung 5.1). Allgemeine Informationen über die Komponente werden in den Attributen `Manufacturer`, `ProductName`, `Version`, `DeliveryDate`, `InstallationDate` und `SerialNumber` festgelegt. Das Attribut `HardwareDependencies` beschreibt Anforderungen der Komponente an die System-Hardware, auf der sie eingesetzt wird. `InstallationLog` gibt den Namen einer Logdatei an, in der Meldungen über den Vorgang der Installation gespeichert werden, während das Attribut `State` Informationen darüber enthält, ob die Installation erfolgreich abgelaufen ist oder nicht. Das Attribut `Support` gibt Daten darüber an, auf welche Art und Weise eine Betreuung oder Unterstützung zu der Komponente angeboten wird.

Computational Interface

An einer Schnittstelle stellen Objekte Dienste zur Verfügung oder nehmen darüber Dienste anderer Objekte in Anspruch. Für ein *Computational Interface* wird die Klasse **compInterface** eingeführt, wobei diese sich in weitere drei Unterklassen verfeinern läßt, je nachdem, um welche Art von Schnittstelle es sich handelt (siehe Abbildung 5.1 und 5.2). Für jede Schnittstelle wird je eine Rolle festgelegt, die durch das Attribut `Role` bestimmt ist. Ein Signal wird über ein **signalInterface** ausgetauscht, wobei das Schnittstellen-Objekt die Rolle des Initiators (*Initiator*) oder des Antwortenden (*Responder*) annehmen kann. Eine Operation erfolgt über ein **operationInterface**, wobei hier das beteiligte Objekt die Rolle eines Clients (*Client*) oder die eines Servers (*Server*) annehmen kann, je nachdem, ob es Dienste in Anspruch nimmt oder zur Verfügung stellt. Die dritte Art von Schnittstelle wird als **streamInterface** bezeichnet und ermöglicht den Austausch von Datenströmen. Dabei ist ein Objekt entweder ein Erzeuger (*Producer*) oder ein Verbraucher (*Consumer*) des Datenstroms. Zusätzlich ist noch anzumerken, daß das Attribut `Role` in den jeweiligen Klassen implementiert werden muß und nicht in der Klasse **compInterface**, da OMT eine strikte Vererbungshierarchie verlangt. Das heißt, daß Attribute und Methoden einer Oberklasse in den Unterklassen nicht weggelassen werden dürfen und daß der Datentyp der Attribute und die Signatur der Methoden nicht verändert werden darf. In diesem Fall ist der Typ des Attributs `Role` für jede Art der Schnittstelle so unterschiedlich, daß das Attribut in die jeweilige Klasse aufgenommen wird.

Mit Hilfe dieser Klassen und deren Attribute kann eine Managementanwendung überwachen, welche Interaktionen an der Schnittstelle der Objekte möglich sind. So wie die Agenten einer Managementanwendung als Computational Objects gelten, sind alle Managementschnittstellen von beteiligten Objekten Computational Interfaces, über die Managementinformation ausgetauscht wird und auch steuernde Operationen auf Objekte erfolgen.

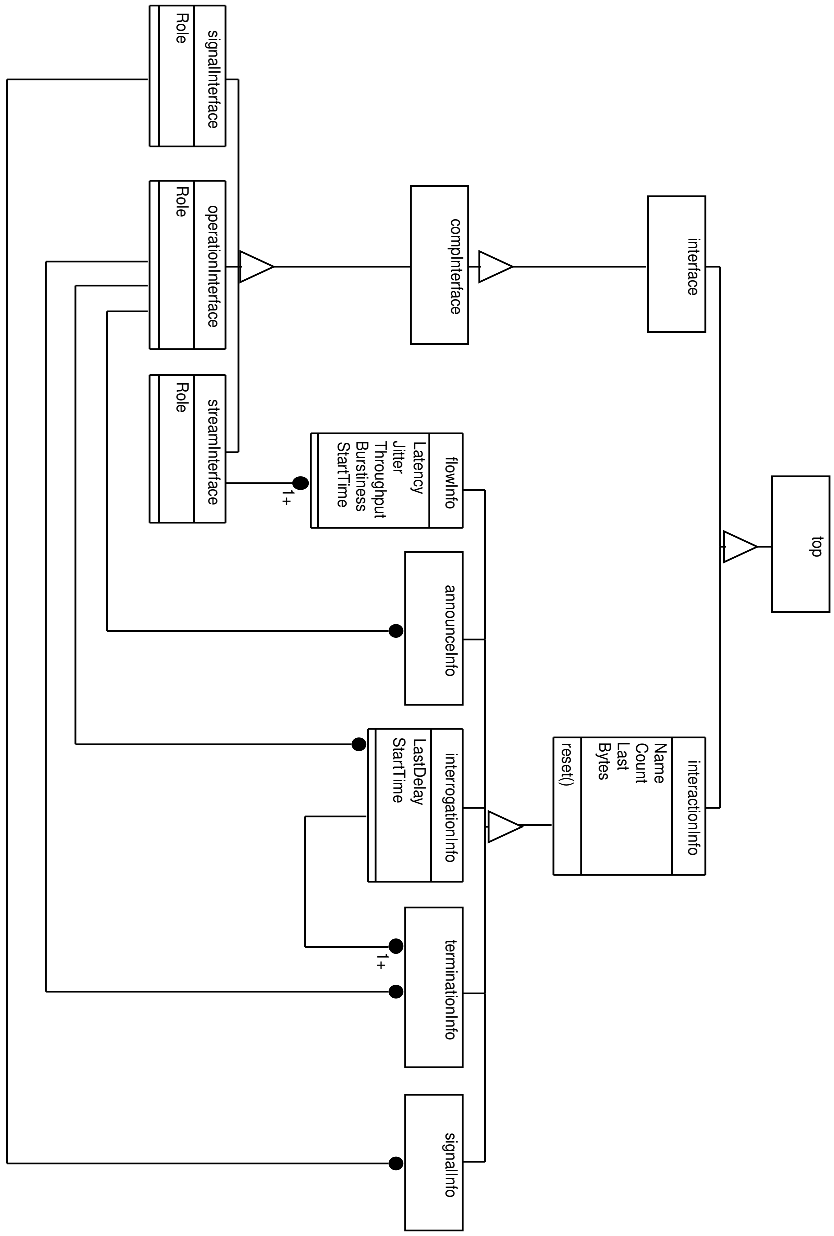

Abbildung 5.2: Generische Objektklassen für Schnittstellen

Computational Interface Template

Auch zu jeder Schnittstelle werden mit Hilfe eines Template deren Eigenschaften festgelegt. Die dazugehörige Klasse **compInterfaceTemplate** leitet sich auch von der Oberklasse **template** ab (siehe Abbildung 5.1).

Das Attribut `Interactions` gibt diejenigen Interaktionen an, die ein Objekt an dieser Schnittstelle ausführen kann. Die Rolle, die es dabei annehmen kann, legt das Attribut `Role` fest. Die Dienstgüteanforderungen an die Umgebung werden in dem Attribut `EnvironmentConstraints` angegeben und von dem Objekt vorausgesetzt, damit es selber die eigenen Dienste mit einer bestimmten Güte anbieten kann. `Behaviour` beschreibt das beobachtbare Verhalten des Objekts bezüglich der ausgeführten Interaktionen.

Interaction Info

Für das Anwendungsmanagement ist nicht nur die Definition der Schnittstellen wichtig, die Objekte zur Verfügung stellen, sondern auch Informationen über die dort stattfindenden Interaktionen. So wird die Oberklasse **interactionInfo** eingeführt (siehe Abbildung 5.2), die Informationen über alle Typen von Interaktionen zur Verfügung stellt. Jede Interaktion wird über einen eindeutigen Namen identifiziert, der in dem Attribut `Name` gespeichert wird. Ein Zähler `Count` wird definiert, der die Häufigkeit der Interaktion festhalten soll. Das Attribut `Last` gibt den Zeitpunkt an, zu dem die Interaktion zuletzt erfolgt ist, und `Bytes` die Anzahl der beim Ausführen der Interaktion übertragenen Bytes. Die Methode `reset()` ermöglicht ein Zurücksetzen aller Attribute dieser Klasse.

Von der Klasse **interactionInfo** werden die Unterklassen **flowInfo**, **announceInfo**, **interrogationInfo**, **terminationInfo** und **signalInfo** abgeleitet, die somit die Attribute und Funktionen der Oberklasse erben und weitere eigene spezifizieren.

Die Klasse **flowInfo** spezifiziert die Art der Interaktionen, bei denen ein kontinuierlicher Datenstrom übertragen wird. Um die gute Qualität der ankommenden Daten zu sichern, muß die Dienstgüte der Übertragung garantiert und gegebenenfalls überprüft werden. Dazu sind neben dem Zeitpunkt, an dem die Übertragung begonnen hat (`StartTime`), auch Informationen über die Übertragungsverzögerung (`Latency`), die Varianz der Übertragungsverzögerung (`Jitter`), den Durchsatz (`Throughput`) und das Verhältnis zwischen Spitzen- und durchschnittlicher Datenübertragungsrate (`Burstiness`) von Interesse. Diese Klasse wird über eine 1:n-Assoziation mit der Klasse **streamInterface** in Relation gesetzt, d.h. daß über ein **streamInterface** ein oder mehrere Datenströme übertragen werden.

Interaktionen, bei denen auf eine Anfrage eine Antwort gesendet wird, werden im ODP-Referenzmodell als *interrogations* bezeichnet, die Anfragen als *announcements* und die Antworten als *terminations*. Die Klassen **announceInfo** und **terminationInfo** spezifizieren die Art der Interaktionen, bei denen eine Anfrage bzw. eine Antwort jeden Typs übertragen wird. Für Interaktionen, bei denen in einer Operation auf eine Anfrage eine Antwort zurückgesendet wird, ist die Klasse **interrogationInfo** eingeführt worden. Die Attribute `LastDelay` und `StartTime` geben Aufschluß über die Antwortzeit und den Absendezeitpunkt der Anfrage. Diese drei Klassen werden jeweils durch eine 1:n-Assoziation mit der Klasse **operationInterface** in Relation gesetzt, da einer Instanz der Schnittstelle zur Laufzeit eine oder mehrere Instanzen der Interaktionen-Info zugeordnet werden können.

Wird nur ein Signal über eine Schnittstelle übermittelt, so wird für diese Art der Interaktion die Klasse **signalInfo** eingeführt. Auch diese Klasse wird über eine 1:n-Assoziation mit der entsprechenden Schnittstelle **signalInterface** verbunden. Einen Überblick über die Schnittstellen-Objektklassen und die entsprechenden Interaktionen gibt Abbildung 5.2.

Engineering Viewpoint

Im folgenden Abschnitt werden die Basisklassen *Basic Engineering Object*, *Capsule*, *Cluster*, *Nucleus*, *Node* und *Channel* des Engineering Viewpoints beschrieben, die in Abbildung 5.3 zusammengefaßt sind.

Basic Engineering Object, Cluster und Capsule

Das Referenzmodell bezeichnet einen Prozeß im virtuellen Speicher eines Rechners als Capsule. Die Klasse **capsule** ist die Abstraktion eines sich in Ausführung befindenden Prozesses. Das Attribut `ID` stellt den systemspezifischen Identifikator dar, `ElapsedTime` enthält Informationen über die Laufzeit des Prozesses und `CPUTime` über die Prozessorzeit. Der momentane Zustand des Prozesses ist in dem Attribut `State` gespeichert, `Memory` gibt die Menge des belegten RAM-Speichers und `Owner` den Besitzer des Prozesses an. Die Methoden `stop()`, `resume()`, `signal()` und `terminate()` erlauben jeweils das Stoppen und Wiedereinsetzen eines Prozesses, das Senden von Signalen an einen Prozeß und das Terminieren des Prozesses. Die Klasse **capsule** kann bezüglich des Systems, auf dem ein Prozeß realisiert wird, in Unterklassen spezifiziert werden (siehe Abbildung 5.3).

Die Klassen **basicEngObject** und **cluster** werden ohne eigene Attribute und Funktionen in das Objektmodell eingefügt. Lediglich die Beziehungen zwischen den Objekten werden mit Hilfe von Aggregationsbeziehungen dargestellt. Ein Capsule besteht aus einem oder mehreren Clusters, die ihrerseits aus mehreren Basic Engineering Objects bestehen (siehe dazu Abschnitt 3.2.2).

Nucleus

Das Betriebssystem eines Rechners wird im ODP-Referenzmodell auf die Klasse **nucleus** abgebildet. Die Attribute `AdminState`, `OperationalState` und `UsageState` geben Auskunft über den aktuellen Status des Betriebssystems. Die Werte, die die jeweiligen Attribute annehmen können, sind die gleichen, die schon im Rahmen eines Computational Objects definiert wurden (siehe Abschnitt 5.1.1). Der Hostname des Rechners, auf dem das System läuft, ist in dem Attribut `Name` gespeichert, der Name und die Version des Betriebssystems sind in dem Attribut `OS` gespeichert, die Systemzeit, die Ländereinstellungen und die Zeitzone sind in den Attributen `Date`, `CountryInfo` und `Timezone` gespeichert. Das Attribut `Uptime` enthält die Zeitspanne seit dem letzten Neustart. Mit Hilfe der Methoden `shutdown()` und `restart()` sind jeweils der Abschluß oder Neustart des Systems möglich.

Node

Die Klasse **node** beschreibt die Hardware eines Rechners. Das Attribut `ClockRate` enthält Informationen über die Taktrate eines Rechners, in dem Attribut `Hardware` sind Daten über die Hardwareausstattung des Rechners gespeichert, in `Location` Angaben über den Aufstellungsort des Rechners, in `Contact` der zuständige Administrator und in `Uptime` die Zeitangabe, seit wann der Rechner läuft.

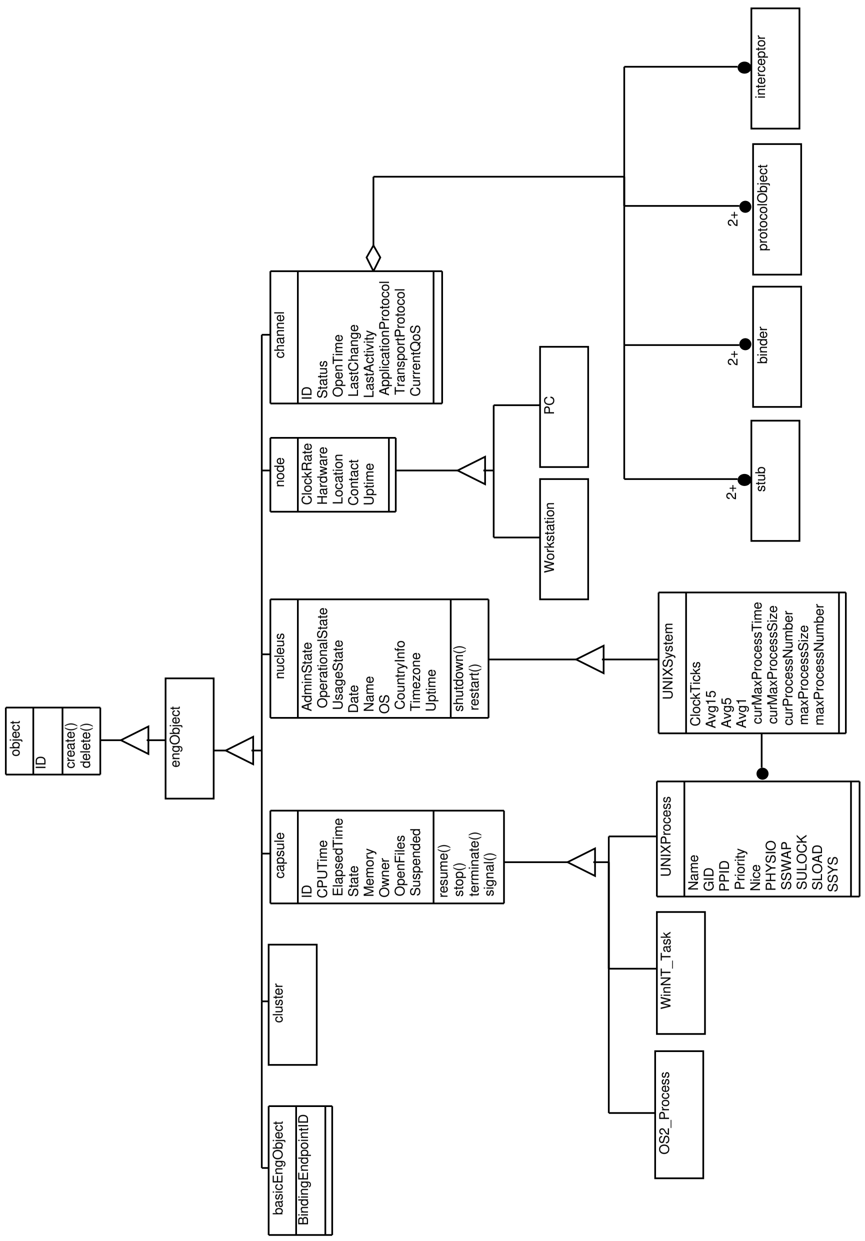

Abbildung 5.3: Generische Objektklassen des Engineering Viewpoints

Channel

Kommunikationsverbindungen zwischen Komponenten einer verteilten Anwendung werden im RM-ODP auf Channels abgebildet. Die Klasse **channel** ist die Abstraktion einer solchen Verbindung. Jede Verbindung wird durch einen eindeutigen Identifikator gekennzeichnet, der in dem Attribut `ID` gespeichert ist. Das Attribut `Status` enthält den aktuellen Status einer Verbindung und kann die Werte *up*, *down*, *congested* oder *unknown* annehmen. Der Zeitpunkt, zu dem die Verbindung kreiert wurde, ist in dem Attribut `OpenTime` gespeichert. Die letzte Konfigurationsänderung und den Zeitpunkt der letzten Aktivität der Verbindung kann man aus den Attributen `LastChange` bzw. `LastActivity` auslesen. Angaben über die Protokolle der Anwendungs- bzw. der Transportschicht sind in den Attributen `ApplicationProtocol` und `TransportProtocol` gespeichert. Die realisierte Dienstgüte der Verbindung ist in dem Attribut `CurrentQoS` festgehalten.

Ein Channel besteht im Referenzmodell aus Stubs, Binders, Protocol Objects und bei Bedarf, falls sich die kommunizierenden Objekte in unterschiedlichen Domänen befinden, aus Interceptors, die eine Art Gateway darstellen. Die interne Struktur eines Channels wird im Objektmodell durch die Aggregationsbeziehung der Klasse **channel** zu den Klassen **stub**, **binder**, **protocolObject** und **interceptor** dargestellt (siehe Abbildung 5.3). Das Attribut `CommDomain` der Klasse **protocolObject** gibt die Kommunikationsdomäne an (z.B. OSI oder Internet), in der das Protokoll-Objekt etabliert ist.

5.1.2 Generische Management-Objektklassen für verteilte Systemdienste

Nach der Festlegung von generischen Klassen für das Management von verteilten Anwendungen sollen speziell für verteilte Systemdienste typische Merkmale betrachtet und in generischen Klassen zusammengefaßt werden. Systemdienste stellen die Basis für verteilte Anwendungen dar, damit diese in einem verteilten System kommunizieren und korrekt funktionieren können. Die meisten Systemdienste sind nach dem Client/Server-Prinzip aufgebaut. Ein Server ist ein Anbieter von Diensten, während Clients die Dienstnutzer sind. Ein Client sendet eine Anfrage an den Server, der daraufhin diese Anfrage bearbeitet und das Ergebnis als Antwort an den Client zurückschickt. Je nachdem, wie Clients realisiert sind, warten sie auf die Antwort des Servers, ohne dabei weitere Aufträge verschicken zu können, oder sie haben die Möglichkeit, mehrere Anfragen zu versenden und die Antworten in einer beliebigen Reihenfolge zu empfangen. Normalerweise werden Server und Clients als Software-Komponenten realisiert und deshalb im Objektmodell als Computational Objects betrachtet.

Um möglichst viele Systemdienste mit einer Managementanwendung kontrollieren zu können, müssen im Objektmodell diese Klassen, deren Attribute und Funktionen so generisch wie möglich gestaltet werden. Die speziellen Systemdienste können dann durch weitere Untergliederung und Einführung neuer Klassen, die sich von den allgemeinen ableiten, näher bestimmt werden. Das Objektmodell zeichnet sich also dadurch aus, daß von allgemeinen Eigenschaften verteilter Anwendungen aus den oberen Ebenen mit jeder absteigenden Ebene immer speziellere und anwendungsspezifische Eigenschaften abgeleitet werden, die ganz bestimmte verteilte Anwendungen charakterisieren.

Von der Klasse **Server**, die nur allgemein jede Art von Server charakterisiert, können spezielle Server abgeleitet werden. Das schon vorhandene Objektmodell enthält spezielle Klassen für Server, die die Dienste NFS und NIS anbieten. Im Rahmen dieser Diplomarbeit sollen speziell WWW-Server betrachtet werden, die auf Basis des HTTP spezielle Informationsdienste zur Verfügung stellen. WWW-Clients nutzen diese Dienste, indem sie von Servern Informationen, die in Form von HTML-Dokumenten gespeichert sind, anfordern. So wird von der Klasse

Client, die im Objektmodell jede Art von Client charakterisiert, eine neue Klasse abgeleitet, die speziell WWW-Clients kennzeichnet.

Durch Einführung der generischen Klassen **Server** und **Client** sollen also die gemeinsamen Eigenschaften möglichst vieler Systemdienste zusammengefaßt werden, um mit *einer* Managementanwendung integriertes Management betreiben zu können. Weitere Klassen, die sich von diesen beiden Klassen ableiten lassen, sollen eine speziellere Abstufung verteilter Systemdienste erlauben und ermöglichen somit eine feinere Abstufung der Management-Aufgaben, die die Managementanwendung zusätzlich realisieren kann.

Server

Ein Server stellt seine Dienste an einer oder mehreren Schnittstellen zur Verfügung. Generische Attribute, die für jede Art von Servern gelten, werden in der Klasse **Server** zusammengefaßt. Die meisten Attribute sind Zähler, deren Werte sehr wichtige Daten über die Performance und die Qualität der Dienste bereitstellen. Sämtliche Zähler berechnen Werte über ein bestimmtes festgelegtes Zeitintervall. Der Anfangszeitpunkt der Messung ist entweder der Startzeitpunkt der Server-Instanz oder der Zeitpunkt, an dem die Zähler zuletzt zurückgesetzt wurden. Der aktuelle Abfragezeitpunkt bedeutet das Ende des Meßintervalls.

Das Attribut `Request` zeigt die Gesamtanzahl der in einem Intervall beim Server angekommenen Aufträge an. `Reject` gibt die Anzahl der Aufträge an, die der Server in dem Zeitintervall wegen Mangel an Ressourcen nicht bearbeiten konnte, während `Accept` die Aufträge mitzählt, die der Server empfangen und akzeptiert hat. Es gilt `Accept = Request - Reject`. Akzeptierte Aufträge können vom Server später auch noch verworfen werden, falls der Client z.B. keine Berechtigung hat, an den Server Aufträge zu verschicken. Ein anderes Attribut `NotAuthorized` kann die Anzahl dieser Aufträge mitzählen, die der Server aufgrund mangelnder Berechtigung von Clients verworfen hat.

Das Attribut `Depart` zählt die Anzahl der akzeptierten Aufträge, die der Server im festgelegten Zeitintervall schon *bearbeitet* hat, dessen Bearbeitung also schon beendet ist, unabhängig von dem Ergebnis der Bearbeitung. In `Depart` sind also auch die Aufträge enthalten, die der Server aufgrund von Prüfungen später verwirft. Die Anzahl der in dem festgelegten Intervall *nicht bearbeiteten* Aufträge wird in dem Attribut `Utilization` festgehalten. Der Wert errechnet sich folgendermaßen: `Utilization = Accept - Depart`. D.h. daß der Wert `Utilization` die Anzahl der zum Zeit**punkt** der Abfrage noch nicht bearbeiteten Aufträge anzeigt. Um diesen Wert nicht durch inkorrekte Werte von `Accept` und `Depart` zu verfälschen, die aufgrund von unterschiedlichen Abfragezeitpunkten der beiden Werte entstehen können, wird er vom Server zum gleichen Zeitpunkt errechnet und bereitgestellt.

Ein weiteres Attribut `Delay` gibt Aufschluß über die Bearbeitungszeit des letzten im Intervall beendeten Auftrags. Dieser Wert könnte für Performancetests in bestimmten Intervallen abgelesen und mit früheren oder mit Durchschnittswerten verglichen werden, um einen Vergleich über die aktuelle Auslastung des Servers zu erhalten.

Zwei weitere wichtige Attribute sind `Bad` und `Error`, die unterschiedliche Arten von Fehlern mitzählen. `Bad` gibt die Art von Fehlaufträgen an, die der Server als fehlerhaft erkennt und deswegen nicht mehr bearbeitet. Dagegen zählt `Error` die Aufträge, die der Server aufgrund

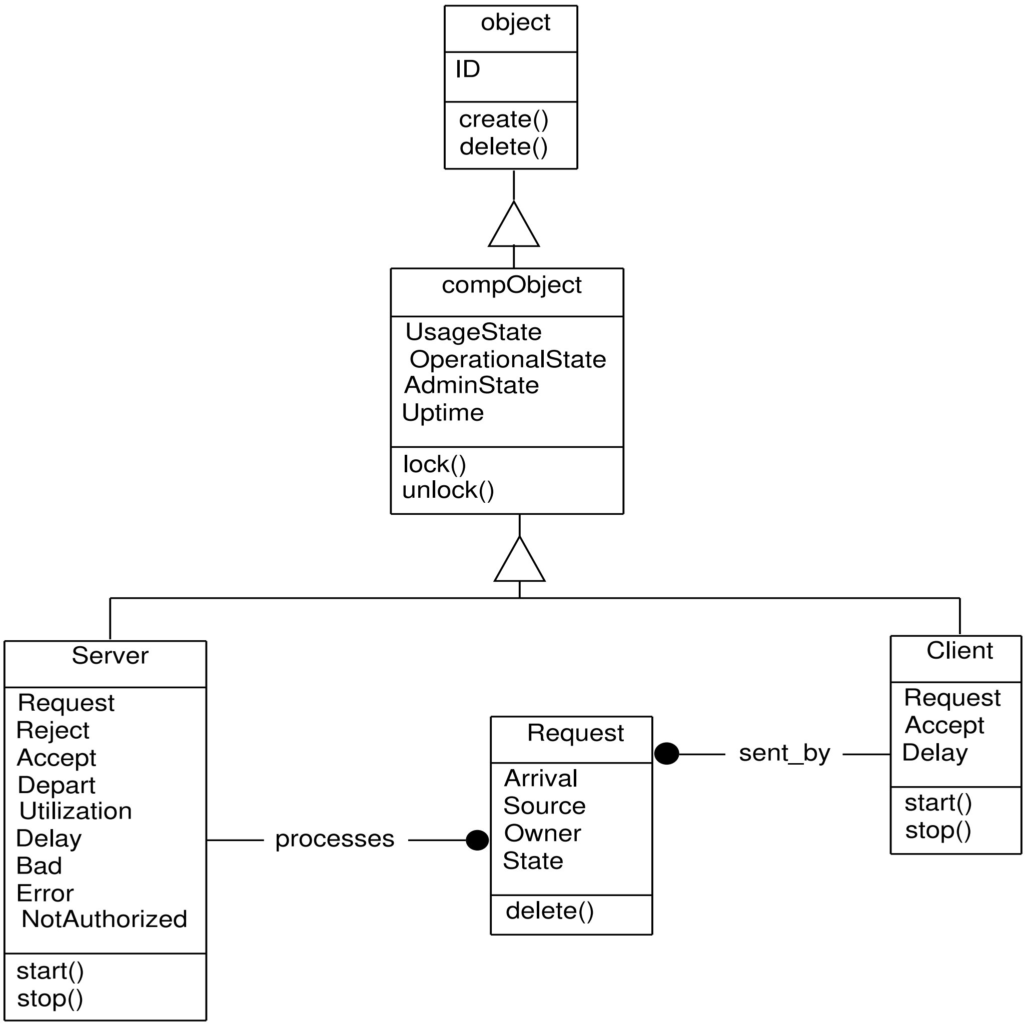

Abbildung 5.4: Systemspezifische Objektklassen

von eigenem Verschulden (interner Serverfehler usw.) nicht bearbeiten konnte. Die Methoden `start()` und `stop()` erlauben das ordnungsgemäße Starten und Stoppen des Servers.

Weiter sind für die Klasse **Server** noch die Attribute wichtig, die sie von den Oberklassen **object** und **compObject** erbt. Das Attribut `ID`, das **Server** von der Klasse **object** erbt, ist für jedes Objekt gleich und stellt den eindeutigen Identifikator des Objektes dar. Das Attribut

`Uptime` aus der Klasse **compObject** legt für jedes Computational Object den Startzeitpunkt der Instanz bzw. der Software-Komponente fest, im diesem Fall also den Startzeitpunkt des Servers. Das Attribut `UsageState` zeigt die aktuelle Nutzung der Server-Dienste an und ist nur lesbar, kann also nicht verändert werden. Es kann folgende Werte annehmen:

- *idle*, falls der Server gerade nicht benutzt wird
- *active*, falls der Server einen oder mehrere Aufträge bearbeitet, aber noch über freie Kapazitäten zur Bearbeitung weiterer Aufträge verfügt
- *busy*, falls er völlig ausgelastet ist
- *unknown*, falls der Zustand nicht bestimmt werden kann

Das Attribut `OperationalState` gibt Aufschluß über die Betriebsbereitschaft des Servers und kann folgende Werte annehmen, die auch nur lesbar sind:

- *enabled*, falls der Server in der Lage ist, Aufträge entgegenzunehmen
- *disabled*, falls er keine Aufträge entgegennehmen kann
- *unknown*, falls die Betriebsbereitschaft nicht bestimmt werden kann

Der Administrationszustand des Servers kann über das Attribut `AdminState` abgefragt, also ausgelesen und über die Methoden `lock()` und `unlock()` verändert werden. Das Attribut kann folgende Werte annehmen:

- *locked*, falls der Server keine Dienste erbringen darf
- *unlocked*, falls er Dienste erbringen darf
- *shutting down*, falls der Server die vorhandenen Aufträge noch bearbeitet, aber keine neuen mehr annehmen darf
- *not applicable*, falls diese Funktionalität nicht anwendbar ist und der Server darüber nicht administriert werden kann
- *unknown*, falls der Zustand nicht bestimmt werden kann

Client

Ein Client nutzt Dienste, die von einem Server angeboten werden. Die generische Klasse **Client** ist genauso wie **Server** eine Unterklasse von **compObject** und erbt von dieser die Statusattribute `UsageState`, `OperationalState` und `AdminState`, sowie `Uptime`, aus dem der Startzeitpunkt des Clients abgelesen werden kann. `UsageState` gibt den aktuellen Nutzungsstatus des Clients an, wobei dieser folgende Bedeutung für Clients haben kann:

- *idle*, falls der Client keinen Auftrag abgesendet hat und auch nicht auf eine Antwort für einen früher gesendeten Auftrag wartet
- *active*, falls der Client sequentiell mehrere Aufträge abgesendet hat und nicht blockiert ist, bis die Antwort auf die ersteren Aufträge ankommt, er also weiterhin Aufträge versenden kann

- *busy*, falls der Client blockiert ist, da er auf die Antwort für einen abgesandten Auftrag wartet und somit keine weiteren Aufträge versenden kann
- *unknown*, falls der Zustand nicht bestimmt werden kann

Die Betriebsbereitschaft des Clients kann wie beim Server aus dem Attribut `Operational State` ausgelesen werden und hat die gleiche Bedeutung:

- *enabled*, falls der Client in der Lage ist, Aufträge abzuschicken
- *disabled*, falls er keine Aufträge abschicken kann
- *unknown*, falls die Betriebsbereitschaft nicht bestimmt werden kann

Das Attribut `AdminState` zeigt den Administrationszustand des Clients an und nimmt auch die gleichen Werte wie der Server an, wobei diese dann folgende Bedeutung haben:

- *locked*, falls der Client keine Aufträge absenden darf
- *unlocked*, falls er Aufträge absenden darf
- *shutting down*, falls der Client im Nutzungszustand *active* oder *busy* ein Stopsignal empfängt und somit keine neuen Aufträge mehr absenden darf; danach wechselt der Client in den Administrationszustand *locked* über
- *not applicable*, falls diese Funktionalität nicht anwendbar ist und der Client darüber nicht administriert werden kann
- *unknown*, falls der Zustand nicht bestimmt werden kann

Der Administrationszustand des Clients kann mit Hilfe der zwei Methoden `lock()` und `unlock()` aus der Oberklasse **compObject** verändert werden.

Für die Klasse **Client** werden clientspezifische Attribute eingeführt. Die Anzahl der Aufträge, die in einem bestimmten Zeitintervall vom Client abgeschickt wurden, kann aus dem Attribut `Request` abgelesen werden. Die Anzahl der *korrekt* bearbeiteten Aufträge, für die also eine Antwort ankam, wird im Attribut `Accept` festgehalten. Das Attribut `Delay` zeigt die Antwortzeit des zum Abfragezeitpunkt zuletzt abgesandten Auftrages an. Die Antwortzeit umfaßt neben der Bearbeitungszeit, die der Server gebraucht hat, auch die Übertragungszeiten, die der Auftrag bis zum Server und zurück gebraucht hat.

Request

Die generische Klasse **Request** wird im Rahmen des Auftragsmanagements eingeführt, um einen Überblick über die Einzelaufträge zu gewinnen, die ein Server empfängt. Sie steht in einer *processes*-Assoziation zu der Klasse **Server** und einer *is sent by*-Assoziation zur Klasse **Client** und wird nicht instantiiert, falls Implementierungen von Agenten das Auftragsmanagement für einen Server nicht unterstützen.

Das Attribut `Arrival` zeigt den Zeitpunkt an, an dem der Auftrag beim Server angekommen ist. Die Attribute `Owner` und `Source` geben Auskunft über die Quelle des Auftrags, den Benutzer und den Rechner, von dem der Auftrag abgesandt wurde. Den Bearbeitungszustand, in dem der Auftrag sich gerade befindet, kann man aus dem Attribut `State` auslesen. Mögliche Werte des Attributs sind:

- *waiting*, falls der Auftrag auf seine Bearbeitung wartet
- *processing*, falls der Auftrag gerade bearbeitet wird
- *unknown*, falls der Bearbeitungszustand nicht bestimmt werden kann

Ein Auftrag kann mit Hilfe der Methode `delete()` auch gelöscht werden.

5.2 Bottom-Up-Modellierung

Nachdem bisher generische Objektklassen für das Management von Endsystemressourcen und verteilten Systemdiensten herausgearbeitet wurden, müssen anschließend speziell die Komponenten betrachtet werden, die den Dienst WWW realisieren. Deren spezifische Eigenschaften sollen in Klassen geeignet zusammengefaßt und an die generischen Klassen des bisherigen Objektmodells angehängt werden. Dabei sollen generelle Eigenschaften, die allgemein für Ressourcen dieser Art gelten, so weit wie möglich im Objektmodell nach oben verschoben und typische, speziellere Eigenschaften im Objektmodell nach unten gezogen werden. So kann ein generisches Objektmodell gewonnen werden, das in einer Hierarchie von oben nach unten managementrelevante Information definiert und für das integrierte Management geeignet ist.

5.2.1 Serverspezifische Management-Objektklassen

Als Anbieter des Dienstes WWW sind WWW-Server eine der Hauptkomponenten, die in dem Objektmodell berücksichtigt werden. Anhand der Managementinformation, die WWW-Server allgemein zur Verfügung stellen, muß untersucht werden, welcher Anteil davon auf alle WWW-Server angewandt werden kann. Allgemeine Eigenschaften werden dadurch in der Hierarchie des Objektmodells nach "oben" gezogen, um für alle Arten von WWW-Servern geltende Attribute in oberen Klassen zusammenzufassen. Mit Hilfe einer Bottom-Up-Analyse wurde in Kapitel 4 dieser Diplomarbeit untersucht, welche Managementinformation WWW-Server zur Verfügung stellen. Die herausgearbeiteten Eigenschaften sollen nun in Form von Attributen und Funktionen in geeigneten Klassen geordnet und zusammengefaßt und an die generischen Klassen im bestehenden Objektmodell angehängt werden.

Im Rahmen des ODP-Referenzmodells ist ein WWW-Server ein Computational Object, das seine Dienste an Schnittstellen zur Verfügung stellt. Nachdem ein WWW-Server alle Eigenschaften eines allgemeinen Servers besitzt, wird im Objektmodell eine Klasse **WWW-Server** eingeführt, die alle Attribute und Funktionen von der Klasse **Server** und somit auch von **compObject** und **object** erbt. Sämtliche geerbten Attribute und Funktionen haben für WWW-Server die gleiche Bedeutung wie für die Objekte, in deren Klassen sie definiert werden, so daß sie hier nicht erneut erklärt werden.

Ein WWW-Server bietet in erster Linie seine Dienste an Browser, also WWW-Clients an. Diese rufen mit Hilfe des HTTP Dokumente von ihnen ab, so daß von einer Schnittstelle zwischen Servern und Clients gesprochen werden kann, über die Daten in Form von Anfragen und Antworten ausgetauscht werden. Wird bei einer Anfrage ein CGI-Programm aufgerufen, so stellt der WWW-Server diesem CGI-Programm über eine weitere Schnittstelle Informationen zur Verfügung und empfängt selbst darüber im Gegenzug die Ergebnisse nach Ausführung des CGI-Programms.

Weiterhin können Server mit Hilfe von Hilfsprogrammen (siehe dazu Abschnitt 4.1.2 über Ressourcen-Konfiguration) in ihrer Funktionalität erweitert werden. Die dafür verwendete Schnittstelle (Server-API) ermöglicht den Datenaustausch zwischen WWW-Servern und den entsprechenden Hilfs-Programmen. Abbildung 5.5 gibt einen kurzen Überblick über die Hauptschnittstellen eines WWW-Servers:

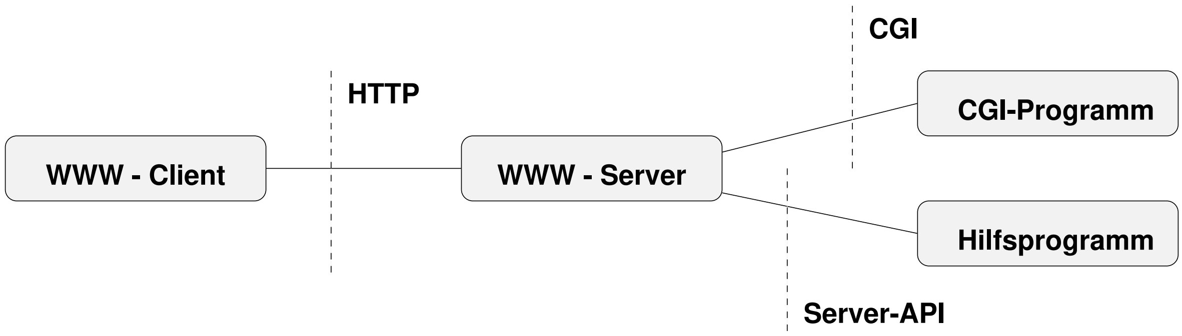

Abbildung 5.5: Kommunikation des Servers über Schnittstellen

In den folgenden Abschnitten wird die serverspezifische Managementinformation aufgrund von drei Kriterien untersucht:

Szenarien-basiert: Anhand der herausgearbeiteten Szenarien und der daraus gewonnenen Managementinformation sollen die Attribute und Funktionen von WWW-Servern in der Klasse **WWW-Server** zusammengefaßt und weitere entsprechend benötigte Klassen hinzugefügt werden.

API-Schnittstelle: Je nach den möglichen Funktionen, die über das API auf den Server ausgeübt werden können, sollen neue Attribute und Funktionen in der Klasse **WWW-Server** ergänzt werden. Ebenso sollen die Schnittstelle und die darüber stattfindenden Interaktionen an sich im Rahmen des Managements untersucht werden.

Schnittstellen zum Server: Hier werden speziell die Schnittstellen des Servers zu Clients oder CGI-Programmen untersucht, die darüber stattfindenden Interaktionen und die ausgetauschte Information.

5.2.2 Szenarienbasierte Definition der Managementinformation

Die Einteilung der Managementaufgaben wird in diesem Kapitel aus Kapitel 4 übernommen und in die Bereiche *Installation und Konfiguration*, *Leistungsmanagement*, *Sicherheitsmanagement*, *Abrechnungsmanagement* und *Fehlermanagement* aufgeteilt.

Installation und Konfiguration

Um den Dienst WWW anbieten zu können, muß der Server installiert und gestartet werden. Vor der Installation muß im DNS ein eindeutiger Hostname für den Server zusammen mit der dazugehörigen IP-Adresse eingetragen werden. Der Hostname eines Servers wird auf das Attribut `ID` der Klasse **object** abgebildet, da jeder WWW-Server über einen eindeutigen Namen identifiziert wird. Dagegen können mehrere Aliasnamen für einen Server definiert werden, die alle auf diesen einen Server verweisen. Diese Aliasnamen werden in den URLs angegeben, mit denen Anfragen an den Server gesendet werden. In der Objektklasse **WWW-Server** wird deshalb das Attribut `AliasName` eingeführt, das die weiteren Namen enthält, die ebenfalls für diesen Server gelten. Da es für einen Server mehrere Aliasnamen geben kann, wird das Attribut als eine Liste von Namen realisiert.

Die zu diesem Server zugehörige IP-Adresse, auf der die Anfragen an den Server empfangen werden, sollte aus Gründen einer leichteren Verwaltung ebenfalls für jeden Server über die Managementanwendung sofort sichtbar sein, um nicht jedesmal eine Suche im DNS oder unter UNIX in der Datei `/etc/hosts` starten zu müssen. Dafür wird das Attribut `IP-Address` eingeführt. Jeder Server empfängt die an ihn gerichteten Anfragen auf einer eindeutig ihm zugewiesenen IP-Adresse. Das gleiche gilt auch für den Port, über den die Anfragen angenommen werden, so daß er im Attribut `Port` abgespeichert wird.

Die Daten, die in diesen vier Attributen gespeichert sind, werden schon vor der Installation eines WWW-Servers festgelegt und systemspezifisch definiert. Nach der Installation kann ein WWW-Server mit Hilfe der regulären Methoden `start()` und `stop()`, die aus der Oberklasse **Server** geerbt werden, gestartet und angehalten werden. Die Methode `restart()`, die den Neustart eines Web-Servers ermöglicht, wird in der Klasse **WWW-Server** implementiert. Dazu muß für jeden Server dessen Prozeß-ID bekannt sein, die in dem Attribut `Pid` gespeichert ist.

Grundkonfiguration

Sämtliche Konfigurationsanweisungen, Log-Daten oder auch Zugriffsbestimmungen, die den Server betreffen, werden in Dateien abgespeichert, so daß in den entsprechenden Attributen die Verzeichnis-Pfade und die Namen dieser Dateien abgespeichert sind:

- `ConfigFile`: Die Anweisungen, die die Grundkonfiguration eines Servers definieren, werden in dieser Datei gespeichert.
- `ResourceConfigFile`: Die Konfigurationsanweisungen, die die Darstellungsarten, Formate und Typen der einzelnen Dateien und Verzeichnisse betreffen, die der Server Clients zur Verfügung stellt, sind in dieser Datei gespeichert.
- `AccessConfigFile`: In dieser Datei sind globale, den ganzen Server betreffende Zugriffsbestimmungen gespeichert.
- `MIMETypesFile`: Jeder MIME-Typ mit der dazugehörigen Anwendung wird in dieser Datei gespeichert.

- `AccessLogFile`: Diese Datei enthält jeden Zugriffsversuch von Clients, die auf den Server zugreifen.
- `ErrorLogFile`: Fehler, die aufgrund von fehlerhaften oder unberechtigten Zugriffen, oder auch aufgrund von internen Serverfehlern entstehen, werden in dieser Datei protokolliert.
- `ScriptLogFile`: Fehler, die während eines fehlerhaften Laufs eines CGI-Programms ausgegeben werden, werden in dieser Datei abgespeichert.

Nach der Installation muß ein Server konfiguriert werden, um die gewünschten Dienste erbringen zu können. Folgende Attribute müssen im Rahmen der Konfiguration eines Web-Servers verfügbar sein:

- `ServerRoot`: In diesem Attribut ist der Verzeichnis-Pfad angegeben, in dem die gesamten Server-Binaries gespeichert sind.
- `DocumentRoot`: Dieses Attribut enthält den Pfad, in dem die Dokumente, auf die der Server zugreifen darf, gespeichert sind.
- `ServerDirectory`: Hier ist der Name des Verzeichnisses gespeichert, in dem sich der Server-Dämon befindet.
- `Type`: Dieses Attribut kann die Werte *standalone* oder *inetd* annehmen und legt damit fest, um welchen Typ von Server es sich handelt. Der Default-Wert des Attributs ist *standalone*.
- `UID`: In diesem Attribut ist die User-Kennung gespeichert, die der Server während des Betriebs benutzt.
- `GID`: Hier ist die Gruppen-Kennung des Servers gespeichert.

Ressourcen-Konfiguration

Wie in Kapitel 4 schon beschrieben, kann im Rahmen der Ressourcen-Konfiguration die Darstellungsart der Dokumente beeinflußt werden, die der Server zur Verfügung stellt. Mit Hilfe von CGI-Programmen können dynamisch HTML-Dokumente generiert werden. Um Verzeichnisse eines Servers als CGI-Verzeichnis zu erkennen, das also nur CGI-Programme enthält, wird ein Attribut `ScriptAlias` in die Klasse **WWW-Server** eingefügt, das eine Liste von Verzeichnisnamen enthält. Das gleiche kann über das Attribut `Includes` auch für Dateien gelöst werden, die nach Server-Side Includes durchgeparst werden sollen. In diesem Attribut wird allerdings eine Datei-Endung gespeichert, so daß jede Datei mit dieser Endung nach Server-Side Includes durchsucht wird.

Endet ein URL beim Zugriff auf einen Server mit einem Verzeichnis-Namen, so wird zuerst versucht, eine Datei namens *index.html* in diesem Verzeichnis zu finden, und diese an den Client zurückgeschickt. Der Name dieser Datei, nach der zuerst gesucht wird, soll in einem Attribut `DirectoryIndex` gespeichert sein, das als Liste von Namen realisiert ist, da mehrere Datei-Namen angegeben werden können.

Weiter kann einem physikalischen Verzeichnis oder einer Datei ein URL-Name zugewiesen werden. In dem Attribut `Alias`, das als Liste von Namen-Paaren realisiert werden soll, sind sämtliche Paare von Verzeichnissen oder Dateien mit den entsprechenden URLs vermerkt. Ebenso werden im Attribut `Redirect` sämtliche Paare von zugehörigen URLs gespeichert, so daß beim Zugriff auf einen der URLs automatisch auf dessen Partner-URL zugegriffen wird.

Auch die Funktionalität der Benutzer-Verzeichnisse muß bei der Konfiguration von Servern berücksichtigt werden. Wird ein Attribut `UserDir` des Typs *Boolean* auf *true* gesetzt, so ist die Einrichtung von Benutzer-Verzeichnissen auf einem Server möglich. In diesem Fall müssen die Werte weiterer Attribute gesetzt werden. Das Attribut `UserURLPrefix` enthält das Sonderzeichen, das vor der Benutzerkennung in einem URL unbedingt eingefügt werden muß, damit der Server erkennt, daß es sich um Dateien aus einem Benutzerverzeichnis handelt. Das Attribut `UserHomeDir` enthält einen Verzeichnisnamen, nach dem der Server im Home-Verzeichnis des Benutzers sucht und in dem die Dateien des Benutzers abgelegt sind. In dem Attribut `PasswdFile` ist der Name einer Datei gespeichert, in der alle Namen der Benutzer und deren Home-Verzeichnis eingetragen sind. Anhand dieser Datei überprüft der Server, wo sich das Home-Verzeichnis des Benutzers befindet, und sucht darin die gewünschte Datei im Unterverzeichnis aus dem Attribut `UserHomeDir`.

Virtuelle Server

Für die Modellierung virtueller Server wird eine eigene Klasse **Virtual-WWW-Server** eingeführt, die das Attribut `Type` enthält. Dieses Attribut definiert den Typ des virtuellen Servers, um den es sich bei dieser Instanz handelt. Das Attribut kann die Werte *name* oder *ip* annehmen, je nachdem, ob es sich um einen namensbasierten oder IP-basierten virtuellen Server handelt. Ein Web-Server kann theoretisch beliebig viele virtuelle Server unterstützen, wobei die virtuellen Server die Konfiguration des Hauptservers erben, auf dem sie realisiert sind. Deshalb wird im Objektmodell die Klasse **Virtual-WWW-Server** von der Klasse **WWW-Server** vererbt. Somit erbt ein virtueller Server sämtliche Attribute und Funktionen der Oberklasse **WWW-Server**, die jedoch für jede Instanz eines virtuellen Servers individuell belegt werden können, falls sie sich von denen des Hauptservers unterscheiden.

Zu einem Web-Server gehörende namensbasierte, virtuelle Server empfangen Anfragen auf einer gemeinsamen IP-Adresse und werden durch ihren Hostnamen unterschieden. Anhand der Modellierung verfügt jeder namensbasierte virtuelle Server über einen eindeutigen Namen, den das Attribut `ID` in der Klasse **object** darstellt, und über die gleiche IP-Adresse wie der Hauptserver, da er dessen Attribute und somit auch dessen IP-Adresse erbt. IP-basierte virtuelle Server verfügen einzeln über eine eindeutige IP-Adresse, was anhand der Modellierung bedeutet, daß das Attribut `IP-Address` aus der Oberklasse **WWW-Server** unbedingt überschrieben werden muß.

Um einen Betrieb mit minimalem Angebot zur Verfügung zu stellen, muß auf jeden Fall für jeden virtuellen Server eine Document Root eingerichtet werden. Das heißt, daß das Attribut `DocumentRoot` der Oberklasse **WWW-Server** unbedingt überschrieben werden muß. Werden für den virtuellen Server mehrere Aliasnamen vergeben, so muß das Attribut `AliasName` der Oberklasse **WWW-Server** ebenfalls überschrieben werden.

Leistungsmanagement

Im Rahmen des Leistungsmanagements wird die Auslastung des Servers und die Erstellung von Langzeitstatistiken anhand der Log-Dateien verfolgt. Außerdem ist die Überprüfung und Steuerung von Einzelaufträgen und der Ressourcen-Auslastung durch den Server für den Betrieb des Servers von großer Bedeutung.

Auswertung der Log-Daten

Um Statistiken über die Auslastung des Servers erstellen zu können, müssen die entsprechenden Daten zur Verfügung stehen. Die Log-Dateien-Einträge müssen im Objektmodell auf Attribute abgebildet werden, die ebenso wie bei der Auswertung von Log-Dateien die Erstellung von Langzeit-Statistiken zu jedem beliebigen Teil-Element erlauben. Das heißt, daß auf jedes Teil-Element eines Log-Eintrags zugegriffen werden kann, um es in individuellen Auswertungen einbeziehen und verarbeiten zu können.

Dazu wird in das Objektmodell eine Klasse **LogEntry** eingeführt, die generische, für Log-Dateien allgemeingültige Attribute definiert. Sie steht in einer n:1-Relation zu der Klasse **WWW-Server** (siehe Abbildung 5.6), da einem Server beliebig viele Log-Einträge zugewiesen werden. Folgende Attribute werden in der Klasse definiert (nicht vorhandene Werte für einzelne Elemente, die der Server nicht abfragen konnte, werden durch einen Bindestrich ersetzt):

- `Host`: Name oder IP-Adresse des zugreifenden Hosts
- `rfc931`: Login-Name des entfernten Benutzers
- `Authuser`: Benutzername, mit dem der Benutzer sich authentifiziert hat
- `Date_Time`: Datum und Uhrzeit, wann die Anfrage ankam
- `Request`: Anfragezeile, wie sie vom Client gesendet wurde

Die beiden Log-Dateien **AccessLog** und **ErrorLog** eines Web-Servers unterscheiden sich jedoch durch einige zusätzliche Einträge, die gesondert untersucht werden. Deshalb wird eine weitere Klasse **AccessLogEntry** in das Objektmodell eingeführt, die von der Klasse **LogEntry** vererbt wird und folgende Attribute umfaßt (siehe Abbildung 5.6):

- `Status`: *HTTP Status Code*, der an den Client nach Bearbeiten der Anfrage zurückgegeben wurde (siehe Anhang A)
- `Bytes`: Länge des übertragenen Dokuments in Bytes

Für Einträge in weitere Log-Dateien werden die Klassen **ErrorLogEntry** und **ScriptLogEntry** eingeführt, die allerdings erst später im Rahmen des Fehlermanagements näher erläutert werden.

Erfolgt ein Zugriffsversuch auf den Web-Server, so wird eine Instanz der Klasse **AccessLogEntry** erzeugt und die Attribute mit den entsprechenden Werten belegt. Somit wird für jede Instanz eines Servers und eines virtuellen Servers eine Anzahl an Instanzen der Klasse **AccessLogEntry** erzeugt, so daß darüber jeder Zugriffsversuch “mitgeloggt” wird. Dadurch kann speziell das Mitloggen von Daten in Log-Dateien als überflüssig angesehen werden, wobei dann die Attribute `AccessLogFile`, `ErrorLogFile` und `ScriptLogFile` der Klasse **WWW-Server** ohne Wert gespeichert werden müssen.

Der Zugriff auf jeden einzelnen Wert der Klasse **AccessLogEntry** ist unbedingt notwendig, um wertabhängig Informationen zur Verfügung zu haben. Außerdem ist noch zu erwähnen, daß der Zugriff auf die Attribute der Klassen **AccessLogEntry**, **ErrorLogEntry** und **Script-LogEntry** von Seiten des Managers nur lesend erlaubt ist. Mit Hilfe der Methode `delete()` wird die Instanz der Klasse gelöscht.

Über die einzelnen Daten können Auswertungsfunktionen nach Vorbild der Managementfunktionen des OSI-Funktionsmodells ausgeführt werden. Diese sogenannten *Summarization Functions* sind für das Erstellen von Statistiken über gesammelte Attributwerte geeignet. Dabei können Statistiken über einzelne oder mehrere Attributwerte von Managementobjekten erstellt werden. Auch Mittelwertbildungen oder die Berechnung von Standardabweichungen sind möglich.

Es ist wichtig, den Zeitpunkt und die Häufigkeit der Statistikerstellung bestimmen zu können. Statistiken können mit Hilfe der OSI-Managementfunktionen zu einem bestimmten Zeitpunkt, über ein bestimmtes Zeitintervall oder periodisch über bestimmte Zeitintervalle erstellt werden.

Ein weiteres wichtiges Attribut der Klasse **WWW-Server** ist `HostNameLookups` vom Typ *Boolean*, das, auf *true* gesetzt, die Funktionalität eines Servers für DNS-Look-Ups aktiviert. Somit wird in den Log-Einträgen der Name des zugreifenden Host gespeichert (falls einer existiert), nicht dessen IP-Adresse. Das Setzen des Attribut-Wertes auf *false* deaktiviert die Funktion.

Einzelaufträge

Um Einzelaufträge, die der Server empfängt, steuern und beobachten zu können, ist die generische Klasse **Request** eingeführt worden. Um detailliertere Informationen über die Anfrage an einen Web-Server untersuchen zu können, wird im Objektmodell eine neue Klasse **HTTP-Request** eingeführt. Trift eine Anfrage beim Server ein, so wird diese Klasse instantiiert. Alle Attribute der Klasse **Request** werden mit der gleichen Bedeutung übernommen.

Die Attribute `HTTP_Method` und `HTTP_Version` der neuen Klasse **HTTP_Request** enthalten den Methoden-Namen (`GET`, `POST` usw.) und die Version des HTTP, mit denen die Anfrage gestellt wurde. Das Attribut `Request_URI` enthält den URI, in dem der Name und der Verzeichnispfad der gewünschten Datei angegeben sind. In dem Attribut `General_Header` sind die Header-Felder als *Variable=Wert*-Paare gespeichert, die sowohl für Requests als auch für Responses gelten. Sie enthalten Daten über die gerade zu übertragene Nachricht. Folgende Variablen können dabei einen Wert enthalten: *Cache-Control*, *Connection*, *Date*, *Pragma*, *Transfer-Encoding*, *Upgrade* oder *Via* (siehe dazu auch den Internet Draft zu HTTP 1.1 in [MBLF+97]).

In dem Attribut `Request_Header` sind die Header-Paare abgespeichert, die Informationen über den Client und über die Anfrage an sich enthalten. Diese gelten nur für Requests. Folgende Variablen sind hier vorhanden: *Accept*, *Accept-Charset*, *Accept-Encoding*, *Accept-Language*, *Authorization*, *From*, *Host*, *If-Modified-Since*, *If-Match*, *If-None-Match*, *If-Range*, *Max-Forwards*, *Proxy-Authorization*, *Range*, *Referer* und *User-Agent*.

Das Attribut `Entity_Header` enthält die Paare von Header-Feldern, die Angaben über den Inhalt der Antwort geben, also der zurückgeschickten Datei, die in dem Attribut `Request_URI` genannt wird. Folgende Variablen sind hier möglich: *Allow*, *Content-Base*, *Content-Encoding*, *Content-Language*, *Content-Length*, *Content-Location*, *Content-MD5*, *Content-Range*, *Content-Type*, *ETag*, *Expires*, *Last-Modified* und *extension-header*.

Falls eine Anfrage mittels der HTTP-Methode `POST` gestellt wurde, werden mit der Anfrage Daten übertragen, die der Server mit Hilfe von weiteren Programmen verarbeitet. Diese zurückgeschickten Daten könnten in einem Attribut `Entity_Body` abgespeichert werden. Möchte man das Risiko nicht eingehen, daß die übertragenen Daten einen zu großen Umfang erreichen und dadurch zuviel Speicher verbrauchen, so könnte mit Hilfe eines weiteren Attributs eine Obergrenze in Bytes angegeben werden, die das Attribut `Entity_Body` höchstens umfassen darf. Somit wäre eine detailliertere Bearbeitung der Aufträge an einen Server möglich.

Um sowohl an einen Web-Server eingehende Requests als auch vom Proxy-Server bearbeitete Requests (siehe dazu Abschnitt 5.2.3) mit der gleichen Managementanwendung beobachten und steuern zu können, die ja in diesem Fall beim Proxy sowohl eintreffen als auch von diesem an Web-Server weitergeschickt werden, wurde das Attribut `Incoming` des Typs *Boolean* in der Klasse **HTTP_Request** definiert. Dadurch können eintreffende (Wert: *true*) und ausgehende (Wert: *false*) Requests unterschieden werden.

Eine weitere Fragestellung beim Management von Einzelaufträgen ergibt sich beim Nachprüfen, ob die Aufträge erfolgreich bearbeitet wurden, ob sie in einer bestimmten Zeitspanne bearbeitet wurden usw. Ein Ansatz zum Lösen dieser Aufgaben hat die Firma Tivoli in Form der *Application Response Measurment (ARM)*-Technik entwickelt. Mit Hilfe einer ARM-API wird verteilten Anwendungen ermöglicht, kritische Informationen zu den stattfindenden Transaktionen zu melden. Diese Informationen können Managementanwendungen dazu nutzen, Meldungen zu verschicken, die Fehlfunktionen sichtbar machen.

Um zu überprüfen, ob eine Transaktion in einer bestimmten Zeit durchgeführt wurde, muß eine gewünschte Zeitspanne festgelegt werden. Im Rahmen des ARM wird die Zeit vom Absenden der Anfrage bis zum Ankunftzeitpunkt der Antwort betrachtet. Diese Zeit umfaßt jegliche Übertragungsverzögerungen durch den Netzverkehr und Bearbeitungsprobleme durch Ressourcenmangel auf dem Ziel-System, auf dem die Antwort generiert wird. Dabei kann auch nachgeprüft werden, wo die meiste Zeit während der Transaktion verbraucht wurde oder verlorenging.

Um die Ressourcen-Auslastung des Systems zu überprüfen, wird im Rahmen des ARM auch die Anzahl der stattfindenden Transaktionen betrachtet und der Benutzer, die diese Transaktionen anwenden. Die Methoden der ARM-Technik können als Anregung für die Entwicklung des Objektmodells übernommen und auf entsprechende Klassen darin abgebildet werden. Im nächsten Abschnitt über die **Ressourcen-Auslastung** und in dem Abschnitt **Schnittstellen zum Server** später im Kapitel wird konkret darauf eingegangen.

Ressourcen-Auslastung

Die hier untersuchten Ressourcen sind die *Prozesse* und *Verbindungen*, die der Server beim Start oder später erzeugt und die während des Betriebs laufen bzw. offen sein dürfen. Für ein Management dieser Ressourcen sind die Klassen **capsule** und **channel** mit ihren entsprechenden Unterklassen von Bedeutung. Die Klasse **capsule** ist die Abstraktion eines sich in Ausführung befindenden Prozesses und liefert Informationen zu den aktuell laufenden Prozessen des Systems. Damit können die Prozesse des Servers und alle Kind-Prozesse beobachtet und auch gesteuert werden. Mit dem Methoden der Klasse kann ein Neustart des Server-Dämons durchgeführt werden, Kind-Prozesse des Servers, die Anfragen bearbeiten, können gestoppt oder nur angehalten werden, usw. Außerdem können Informationen rund um die Prozesse abgefragt werden, wie z.B. den Status des Prozesses, die Menge des belegten RAM-Speichers usw. (siehe dazu Abschnitt 5.1.1 über die generischen Klassen des Engineering Viewpoints).

Das gleiche gilt für die Verbindungen, die ein Server für die Bearbeitung der Anfragen öffnet. Mit Hilfe der generischen Klassen **channel**, **compInterface** und **interactionInfo** sowie der dazugehörigen Unterklassen aus dem Objektmodell können jede Art von Verbindungen betrachtet werden, die auf dem System geöffnet wurden. Darauf wird später im Kapitel im Abschnitt über **Schnittstellen zum Server** näher eingegangen.

Durch die Konfiguration eines Web-Servers können zusätzlich nach Wunsch (bei den meisten neueren Versionen von Web-Servern) Einstellungen vorgenommen werden, um so die Server an das eigene System und die verfügbaren Ressourcen anzupassen. Weitere Daten über systemspezifische Ressourcen, wie z.B. weitere Prozesse oder Verbindungen, die der Server während des Betriebs nutzt, wurden im Objektmodell in oberen Ebenen in Form der generischen Klassen und deren Attribute und Funktionen modelliert. Nachfolgend werden nur noch anwendungsspezifische Attribute von WWW-Servern untersucht und ergänzt.

In dem Attribut `StartServers` wird die Anzahl der Kindprozesse gespeichert, die der Server beim Start erzeugen soll. In den Attributen `MinSpareServers` und `MaxSpareServers` wird die minimale bzw. maximale Anzahl der Serverprozesse gespeichert, die während des Betriebs als Reserve laufen sollen, um schneller auf Anfragen antworten zu können. Das Attribut `MaxRequestsPerChild` enthält die maximale Anzahl der Anfragen, die ein Kindprozeß des Servers vor seinem automatischen Beenden bearbeiten soll. In dem Attribut `MaxClients` ist die maximale Anzahl an Kindprozessen gespeichert, die jemals gleichzeitig auf dem System laufen dürfen.

Durch das Attribut `KeepAlive` des Typs *Boolean* soll für einen Web-Server die Unterstützung der persistenten Verbindungen zu einem Client aktiviert bzw. deaktiviert werden können. Ist der Wert des Attributs auf *true* gesetzt, so kann in dem Attribut `MaxKeepAlive` die Anzahl der aufeinanderfolgenden Anfragen gespeichert werden, die maximal über eine persistente Verbindung erlaubt werden. Die Zeit, die über eine persistente Verbindung auf weitere Anfragen gewartet werden soll, soll in dem Attribut `KeepAliveTimeout` gespeichert werden. In einem weiteren Attribut `Timeout` kann die Zeitspanne in Sekunden festgelegt werden, in der eine Client-Verbindung folgender Art bearbeitet wird (siehe dazu auch 4.1.3):

- Die Gesamtzeit zwischen dem Aufbau der Client-Verbindung und der Anfrage darf nicht länger als die Timeout-Zeitspanne sein.

- Die Zeit zwischen dem Erhalt von TCP-Paketen bei `POST`- oder `PUT`-Anfragen darf nicht länger als die Timeout-Zeitspanne sein.
- Die Zeit zwischen den Bestätigungen von TCP-Paketen, die als Antwort vom Server geschickt wurden, darf nicht länger als die Timeout-Zeitspanne sein.

Wenn bei einem dieser Fälle die Zeitspanne überschritten wird, so schließt der Server die Verbindung zum Client.

Sicherheitsmanagement

Wie in Kapitel 4.1.4 im Rahmen des Sicherheitsmanagements festgestellt wurde, unterteilt sich das Aufgabengebiet in Sicherheitsvorkehrungen auf Dateisystemebene, Regelung des öffentlichen Zugriffs von Benutzern auf das Angebot des Servers und die Einrichtung von Kodiermechanismen auf dem Server, um die zwischen Clients und Server übertragenen Daten zu verschlüsseln.

Einstellungen auf Dateisystemebene können im Objektmodell nicht auf Attribute innerhalb der Klasse **WWW-Server** abgebildet werden und sollen in diesem Zusammenhang nicht weiter betrachtet werden.

Einstellungen für den öffentlichen Zugriff von Clients auf Dokumente des Servers können entweder global für den ganzen Server oder verzeichnisbasiert mit Hilfe von *Directory Access Control Files* eingerichtet werden. Zugriffsrechte können aufgrund der IP-Adresse oder des Hostnamen des zugreifenden Systems oder aufgrund des Namen des Benutzers vergeben werden. Werden die Zugriffsrechte global für den ganzen Server vergeben, so kann mit Hilfe eines Attributs `AllowOverride` des Typs *Boolean* die Funktionalität aktiviert bzw. deaktiviert werden, daß jegliche Zugriffsbestimmungen global gelten sollen und Directory Access Control Files gar nicht beachtet werden. Ist das Attribut auf *true* gesetzt, so kann in dem Attribut `IPList` eine Liste von IP-Adressen gespeichert werden, denen der Zugriff auf den Server verwehrt wird. Das gleiche gilt für das Attribut `HostList`, falls der Zugriff Host-basiert verwehrt werden soll. Soll der Zugriff auf den Server global nur für bestimmte Benutzer erlaubt werden, so kann in dem Attribut `AuthPasswdFile` eine Liste von Dateinamen eingetragen werden, in denen sämtliche Benutzer mit ihren gültigen Paßwörtern eingetragen sind. Der Wert des Attributs `AuthType` soll die Art der Authentifikation festlegen und kann *Basic* oder *Digest* sein (siehe dazu Abschnitt 4.1.4).

Sollen die Zugriffsrechte verzeichnisbasiert vergeben werden, so ist das Attribut `AllowOverride` auf *false* gesetzt. In dem Attribut `AccessFileName` ist dann der Name eingetragen, den jedes Directory Access Control File haben muß, um als solches vom Server erkannt zu werden. Die Attribute `IPList`, `HostList` und `AuthPasswdFile` enthalten dann keine Werte. Sie können mit den Methoden `resetIPList()`, `resetHostList` und `resetAuthPasswdFile()` zurückgesetzt werden.

Das Einrichten von Verschlüsselungsmechanismen kann nicht auf Attribute der Klasse **WWW-Server** abgebildet werden, da generell nicht jeder Web-Server diese Funktionalität unterstützt und es außerdem noch sehr viele Sonderfälle und anwendungsspezifische Merkmale pro Verschlüsselungsmechanismus zu beachten gibt.

Abrechnungsmanagement

Im Rahmen des Abrechnungsmanagements ist es schwer, geeignete Attribute zu definieren, die allgemein für Web-Server gelten sollen. Hierzu können die Daten der Klasse **AccessLogEntry** zur Auswertung verwendet werden. Nach Art der Statistikerstellung bezüglich der Auslastung des Servers und der Zugriffsanzahl auf Dokumente können die Attribute aus **AccessLogEntry** mit Managementfunktionen des OSI-Funktionsmodells bearbeitet werden, um individuelle Abrechnungsmodelle zu errechnen. Eine benutzerbasierte Abrechnung ist dabei schwierig, da über zugreifende Benutzer selten bis gar nicht Daten gesammelt werden können (siehe dazu Abschnitt 4.1.5).

Falls in einem Netzwerk nur der Datenverkehr abgerechnet werden soll, der durch den Web-Server verursacht wird, ist eine Auswertung der Bytemenge wichtig, die durch die Übertragung von Dokumenten durch den Server an Clients entsteht. Firmen können an dieser Art von Abrechnung interessiert sein, falls sie ihren Web-Server bei einem Internet Provider stehen haben. Für sie ist dann nur wichtig, wie viele Bytes aus ihrem Dokumentenangebot über eine bestimmte Zeitspanne an Clients übertragen wurden. Sogenannte *Accounting Metering* Funktionen des OSI-Funktionsmodells ermöglichen benutzerorientiertes Sammeln von Daten bezüglich der Benutzung von Ressourcen. Dabei kann unter anderem bestimmt werden, welche Daten zu welchen Zeiten gesammelt werden sollen usw. In diesem Fall müssen die Instanzen der Klasse **AccessLogEntry** für den jeweiligen Web-Server nach dem Wert des Attributs `Bytes` durchsucht und ausgewertet werden.

Fehlermanagement

Jeder Fehler, der während des Betriebs eines Web-Servers entsteht, wird in die ErrorLog-Datei eingetragen. Um die einzelnen Einträge auf entsprechende Attribute abzubilden, wird die Klasse **ErrorLogEntry** in das Objektmodell eingeführt. Sie wird von der Klasse **LogEntry** vererbt (siehe Abbildung 5.6) und enthält das Attribut `Reason`, das den Fehlergrund oder die Bezeichnung des Fehlers in Worte enthält. Die anderen Attribute und die Methode `delete()` werden von der Klasse **LogEntry** mit der gleichen Bedeutung übernommen.

Auf Werte der Attribute der Klasse **ErrorLogEntry** können ebenfalls Statistikfunktionen ausgeführt werden, um eine Übersicht über die Häufigkeit von Fehlern zu haben. Dabei können individuelle, an eigene Bedürfnisse angepaßte Statistiken zusammengestellt werden.

Eine andere Art von Fehlerstatistik kann mit Hilfe des Wertes des Attributs `Status` aus der Klasse **AccessLogEntry** erstellt werden. Dieser Status-Code gibt das Ergebnis der Bearbeitung jeder Anfrage, die an den Web-Server gerichtet wurde, an. Durch Berücksichtigung der Status-Codes, die eine fehlerhafte Bearbeitung der Anfrage bedeuten, werden Informationen über die Häufigkeit von Server-bedingten oder auch Benutzer-bedingten Fehlern gesammelt.

Die Erstellung der Statistiken kann auch konkret für die Überwachung des Servers genutzt werden. Mit Hilfe von *Workload Monitoring Functions* des OSI-Funktionsmodells können Schwellwertüberwachungen durchgeführt werden. Hier wäre die Einführung von Attributen in die Klasse **WWW-Server** denkbar, die als Zähler einzurichten sind und die Häufigkeit von Status-Codes mitrechnen. Mit Hilfe der Workload Monitoring Functions können diese

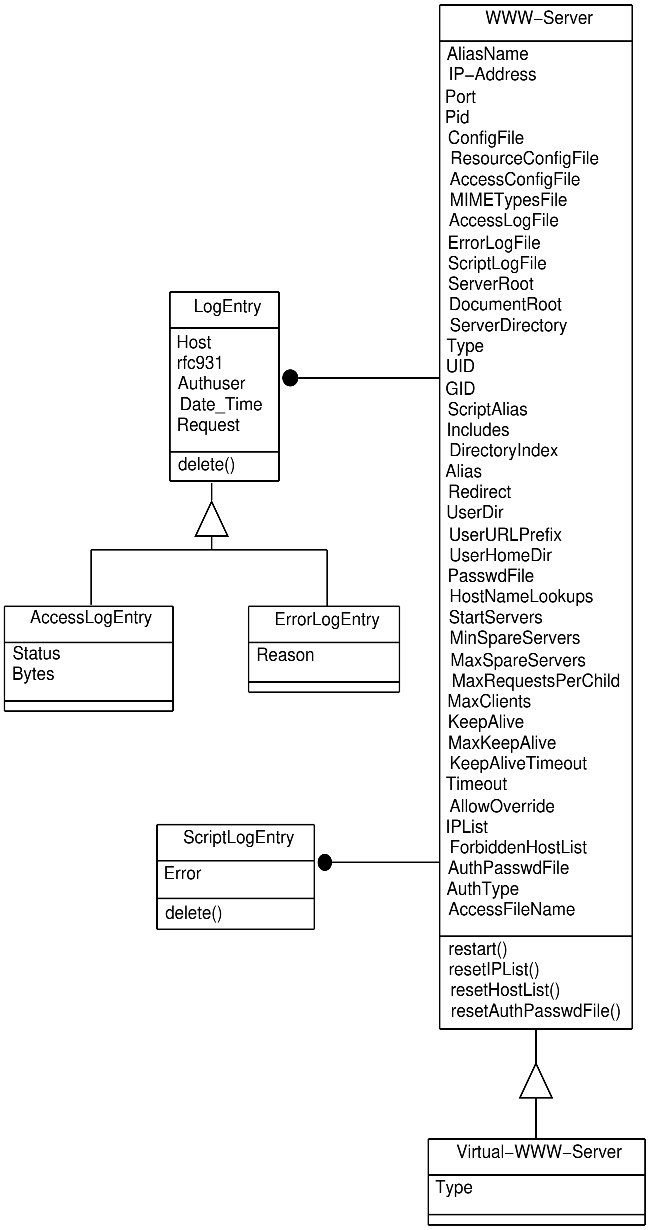

Abbildung 5.6: Objektklasse **Server**

dynamischen Attribute laufend überwacht werden, um bei Überschreiten eines Schwellwertes geeignete Alarme auszulösen. Eine Nutzung der zur Verfügung stehenden Methoden muß an die Bedürfnisse des Systems und auch des Administrators angepaßt und kann nicht verallgemeinert werden, weshalb hier auf das Hinzufügen von weiteren Attributen in diesem Zusammenhang verzichtet wird.

Um Fehler aufzuzeichnen, die bei der Ausführung von CGI-Programmen generiert werden, wird eine weitere Klasse **ScriptLogEntry** in das Objektmodell eingeführt. Sie enthält nur das Attribut `Error`, in dem die erzeugte Fehlermeldung gespeichert ist, da solche Log-Dateien für jeden Servertyp von Hersteller zu Hersteller sehr unterschiedlich gestaltet sein können. Mit Hilfe der Methode `delete()` wird das Objekt gelöscht.

Abbildung 5.6 gibt einen Überblick über sämtliche bisher erläuterten Attribute, Methoden und Unterklassen der Klasse **WWW-Server**.

API-Schnittstelle

Wie in Kapitel 4.1.2 schon erklärt, kann über die API-Schnittstelle mit Hilfe von Anwendungsprogrammen die Funktionalität eines Servers beeinflußt werden. Vordefinierte Funktionen können genutzt werden, um verschiedene Eigenschaften eines Web-Servers an eigene Bedürfnisse anzupassen. Diese Funktionen können auch dazu verwendet werden, um den Server zu konfigurieren oder zu steuern, also um Managementaufgaben darauf auszuführen. Es ist allerdings schwer, hier allgemeingültige Aussagen darüber zu treffen, da sich die Hilfsfunktionen je nach Hersteller und auch je nach Programmiersprache in ihrer Funktionalität stark unterscheiden.

Im Rahmen dieser Diplomarbeit wird anhand des Netscape Enterprise Servers dessen API untersucht, für deren Nutzung sowohl C- als auch Java-Funktionen zur Verfügung stehen. Diese Funktionen können z.B. zur Steuerung und Überwachung von Threads, zum Senden von Daten an den Server über Sockets oder zum Abfragen von Daten über eintreffende Requests verwendet werden, um diese nach eigenen Bedürfnissen weiterverarbeiten zu können. Das Untersuchen jeder Hilfsfunktion würde allerdings den Rahmen der Diplomarbeit sprengen, so daß nur beispielhaft anhand einiger Funktionen die Abbildung der Funktionalität auf Attribute und Funktionen der Klassen im Objektmodell gezeigt werden soll.

So gibt es z.B. Hilfsfunktionen, die die Daten, Header- und Statusinformationen, die an Clients zurückgeschickt werden, abfragen, bearbeiten oder beeinflussen können. Die Funktion **protocol_set_finfo** fragt z.B. die Länge des Inhalts, den Inhalt und das Datum einer Datei ab, wann sie zuletzt geändert wurde. Die gewonnen Daten können nach eigenem Bedarf weiterverarbeitet werden. Um solche Daten zu Managementzwecken zur Verfügung zu haben, wird im Objektmodell die Klasse **Response** eingeführt, von der die Klasse **HTTP-Response** vererbt wird. Generiert ein Web-Server einen Response, so wird eine Instanz der Klasse **HTTP-Response** erzeugt, die Attribute werden mit den entsprechenden Werten belegt. Sobald der Response abgeschickt ist, wird auch die Instanz beendet. So ist immer eine aktuelle Sicht der gerade generierten Antworten gegeben, wobei vor allem der HTTP-Status-Code, der in dem Attribut `HTTP-StatusCode` gespeichert ist, von großer Bedeutung ist. Damit hat man aktuell immer eine Sicht auf richtig oder fehlerhaft bearbeitete Anfragen und kann gleich kontrollieren, ob die vorhandenen Fehler aufgrund von Server-internen Problemen oder aufgrund von fehlerhaft gestellten Anfragen entstanden sind.

Das Attribut `Reason_Phrase` enthält die zum Status Code gehörende Erklärung in Worte (siehe dazu Anhang A). In dem Attribut `General_Header` sind die Header-Felder als "*Variable - Wert*"-Paare abgespeichert, die Angaben zu der übertragenen Nachricht enthalten, aber nicht zu dem Inhalt der Nachricht. Diese Header-Variablen können sein: *Cache-Control*, *Connection*, *Date*, *Pragma*, *Transfer-Encoding*, *Upgrade* oder *Via* (siehe dazu HTTP 1.1 in [MBLF+97]). In dem Attribut `Response_Header` sind die Header-Paare abgespeichert, die Informationen über den Server und über die Zugriffsberechtigung auf die gewünschte Ressource enthalten: *Age*, *Location*, *Proxy-Authenticate*, *Public*, *Retry-After*, *Server*, *Vary* und *WWW-Authenticate*. Das Attribut `Entity_Header` enthält die Paare von Header-Feldern, die Angaben über den Inhalt der Antwort geben, also der zurückgeschickten Datei. Folgende Variablen sind hier möglich: *Allow*, *Content-Base*, *Content-Encoding*, *Content-Language*, *Content-Length*, *Content-Location*, *Content-MD5*, *Content-Range*, *Content-Type*, *ETag*, *Expires*, *Last-Modified* und *extension-header*. Der Inhalt der zurückgeschickten Datei könnte in einem Attribut `Entity_Body` abgespeichert werden, wobei das nicht unbedingt notwendig ist, zumal bei großen Dateien viel Speicher dafür benötigt werden würde. Analog zu der Klasse **HTTP_Request** wird auch in der Klasse **HTTP_Response** das Attribut `Incoming` des Typs *Boolean* definiert, um zwischen eingehenden und ausgehenden Responses zu unterscheiden.

Zwei weitere Funktionen **protocol_uri2url_dynamic** und **protocol_uri2url** bilden physikalische URIs auf virtuelle URIs ab oder setzen aus zwei gegebenen URI-Strings einen URL zusammen. Damit können Anfragen, die an eine bestimmte Adresse gerichtet sind, auf eine neue Adresse umgeleitet oder virtuelle URLs auf konkrete physikalische Verzeichnisse abgebildet werden. Beide Funktionalitäten werden mit den in der Klasse **WWW-Server** vorhandenen Attributen `Alias` und `Redirect` abgedeckt.

Ebenso kann die Funktionalität anderer Funktionen, die die Portnummer oder den Hostnamen des Servers abfragen, auf schon vorhandene Attribute der Klasse **WWW-Server** abgebildet werden. Informationen über die eingehenden Request-Daten können auf Attribute der Klasse **HTTP-Request** abgebildet werden. Eine letzte Funktion, die hier aufgegriffen wird, ist **system_rename**. Mit ihr können Dateien umbenannt werden. Diese Funktionalität kann für die Administration der Log-Dateien sehr nützlich sein. Falls man eine aktuelle Log-Datei umbenennen möchte, um auf den alten Namen eine neue zu öffnen, die mit neuen Log-Daten gefüllt wird, ist diese Funktion anzuwenden. Dazu wird die Funktion `rename(oldFile,newFile)` in die Klasse **WWW-Server** aufgenommen.

Schnittstellen zum Server

Um die Schnittstelle und die darüber stattfindenden Interaktionen zwischen einem Server und einem CGI-Programm oder einem Client zu beobachten, müssen die Instanzen der Schnittstelle beobachtet werden. Zum besseren Verständnis ist die Abbildung 5.2 der generischen Schnittstellen-Klassen des Computational Viewpoints (Seite 87) in Erinnerung zu rufen.

Wird eine Anfrage von einem Client an den Web-Server gerichtet, so instantiiert dieser die Klasse **operationInterface** und kann mit Hilfe von Operationen, den sogenannten HTTP-Methoden, Interaktionen mit dem Client austauschen bzw. Aufträge entgegennehmen. Zu jedem Anfragetyp wird eine Instanz der Klasse **interrogationInfo** erzeugt. Findet z.B. eine `GET`-Anfrage statt, so wird der Zähler `Count` der Klasse **interrogationInfo** um 1 erhöht und das Attribut `Name` erhält den Wert *GET*. Das Attribut `Last` zeigt den Zeitpunkt an, an dem die letzte `GET`-Anfrage stattgefunden hat, und das Attribut `Bytes` gibt die Anzahl der Bytes an, die bei `GET`-Anfragen übertragen wurden. Die Bearbeitungszeit des letzten Auftrags steht in dem Attribut `LastDelay` und der Anfragezeitpunkt in dem Attribut `StartTime`. So kann festgestellt werden, wie viele `GET`-Anfragen über einen bestimmten Zeitraum stattgefunden haben, wie viele Daten über die Schnittstelle übertragen wurden oder wann die letzte GET-Anfrage beim Server angekommen ist und wie lange ihre Bearbeitung gedauert hat. Diese Daten können im Rahmen des Leistungsmanagements genutzt werden.

Werden bei einer GET-Anfrage zusätzlich Daten in dem Request-URL mitgeschickt, so schreibt der Server diese Daten in die Umgebungsvariable QUERY_STRING. Die GET-Anfrage ist dann an ein auszuführendes Programm gerichtet, ein sogenanntes CGI-Programm, das die Daten weiterverarbeiten kann. Der Server erzeugt daraufhin eine Kopie von sich selbst. Der neue Prozeß wird als CGI-Prozeß bezeichnet, der den Code des CGI-Programms ausführt. Die von diesem CGI-Prozeß erzeugten Ergebnis-Daten werden anschließend dem Server übergeben und dieser schickt sie, nachdem er die notwendigen Header-Informationen hinzugefügt hat, über die vorhandene Verbindung an den Client zurück. Sendet der CGI-Prozeß Daten an den Server, so wird eine neue Instanz der Klasse **streamInterface** erzeugt und gleichzeitig die Werte der Klassen-Instanz **flowInfo** aktualisiert. Falls es noch keine Instanz der Klasse **flowInfo** gibt, wird eine erzeugt.

Wird eine POST-Anfrage an den Server geschickt, so enthält diese auch mitgeschickte Daten. Hier handelt es sich allerdings um eine größere Menge an Daten, die im Gegensatz zur GET-Anfrage nicht an dem URL angehängt mitgeschickt wurden. Der Server übergibt diese Daten über das STDIN an den CGI-Prozeß, er erzeugt also eine neue Instanz der Klasse **streamInterface**, wobei die Werte der Klasse **flowInfo** aktualisiert werden. Die Übergabe der Daten zurück an den Server erfolgt auf die gleiche Art und Weise wie bei der GET-Anfrage.

Über die Anzahl der offenen Instanzen von Computational Interfaces kann also ein guter Überblick über die gerade offenen Verbindungen des Servers gewonnen werden und auch über die darüber stattfindenden Interaktionen, über ihren Typ, ihre Häufigkeit und auch über die darüber ausgetauschte Datenmenge.

5.2.3 Proxyspezifische Management-Objektklassen

Proxies sind Vermittler von Dokumenten zwischen einem Client und anderen Servern oder Proxies. Proxies sind entweder allgemein Server mit spezifischen Funktionalitäten oder speziell Web-Server, die durch erweiterte Funktionalitäten zusätzlich als Proxy konfiguriert werden können. Da im Rahmen dieser Diplomarbeit die Proxies betrachtet wurden, die als Web-Server mit erweiterten Funktionalitäten ausgestattet sind, wird die Modellierung der Klassen darauf ausgerichtet.

Für Proxy-Server wird eine neue Klasse **Proxy-Server** eingeführt, die von der Klasse **WWW-Server** vererbt wird. Für die andere Art von Proxy-Server würde die Klasse **Proxy-Server** von der allgemeinen Klasse **Server** abgeleitet werden. In diesem Fall würden auch noch weitere Attribute und Funktionen in der Klasse enthalten sein, die hier aber nicht weiter untersucht werden sollen. Im Folgenden werden also die Eigenschaften von Web-Servern untersucht, die zusätzlich als Proxy-Server konfiguriert werden.

Wird ein Web-Server als Proxy konfiguriert, so muß ihm mittels einer **Pass**-Anweisung mitgeteilt werden, welche Protokollangaben er in den Anfragen verstehen und bearbeiten soll. Anfragen, die mit einem dem Proxy bekannten Protokoll anfangen, werden von ihm als gültige Anfragen erkannt und an die entsprechenden Server weitergereicht. Deswegen sind in dem Attribut `PassProtocol` der Klasse **Proxy-Server** die gültigen Protokollnamen als Liste gespeichert.

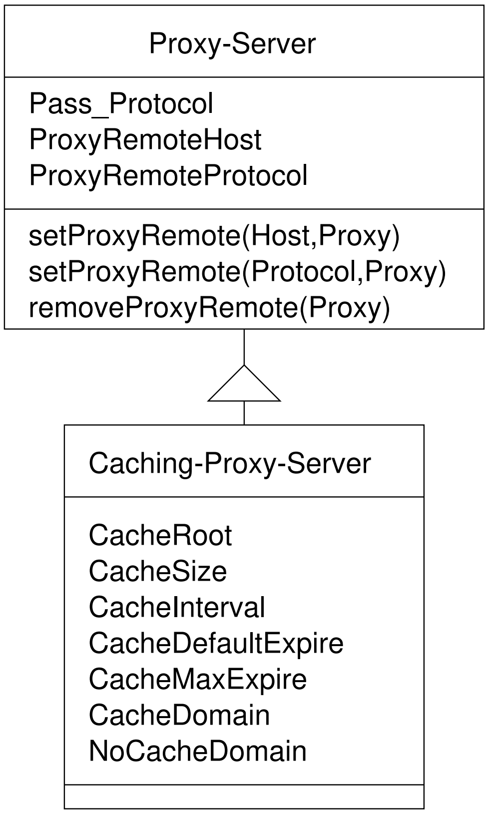

Abbildung 5.7: Objektklasse **Proxy-Server**

Proxies können auch in Reihe hintereinander geschaltet werden, so daß abhängig vom Protokoll der Anfrage oder vom Host, an den die Anfrage gerichtet ist, bestimmte Anfragen an bestimmte Proxies weitergereicht werden, statt sie direkt an die entsprechenden Server weiterzuleiten. In dem Attribut `ProxyRemoteHost` sind "*Host - Proxy*"-Paare in Form einer Liste gespeichert, die einem bestimmten Host einen Proxy zuordnen. Alle Anfragen, die an einen Host aus der Liste gerichtet sind, werden an den dazugehörigen Proxy weitergeleitet. Das gleiche gilt für das Attribut `ProxyRemoteProtocol`, nur daß hier Paare der Form "*Protocol - Proxy*" gespeichert sind, so daß die Anfragen abhängig vom angewendeten Protokoll weitergeleitet werden. Einzelne Einträge in die Liste können mit Hilfe der Funktion `setProxyRemote(Host,Proxy)` beziehungsweise `setProxyRemote(Protocol,Proxy)` eingefügt werden. Die Funktion `removeProxyRemote(Proxy)` erlaubt das Entfernen eines Paares aus beiden Listen, falls der Wert für den Proxy dem *Proxy*-Eingabeparameter der Funktion entspricht.

Proxies können auch als Caching-Proxies eingerichtet werden, so daß sie schon einmal bearbeitete Dateien in einem Cache zwischenspeichern und diese bei erneuten Anfragen nach der gleichen Datei aus dem Cache auslesen. So können Anfragen an Server gespart werden. Da ein Proxy nicht unbedingt gleich ein Caching-Proxy ist, sondern erst dazu konfiguriert werden muß, wird im Objektmodell die Klasse **Caching-Proxy-Server** eingeführt, die von der

Klasse **Proxy-Server** vererbt wird. In dem Attribut `CacheRoot` wird der Verzeichnisname gespeichert, unter dem die Dateien abgespeichert werden. Die Größe der Datenmenge, die in diesem Verzeichnis höchstens abgelegt werden soll, ist in dem Attribut `CacheSize` zu finden. Wird diese Menge überschritten, werden Daten aus dem Speicher gelöscht. Das Zeitintervall, in dem die Überprüfung der Datenmenge erfolgen soll, ist in dem Attribut `CacheInterval` gespeichert.

Das Aktualitätsproblem, das sich beim Speichern von Dateien stellt und das schon in Kapitel 4.2.1 erläutert wurde, soll mit Hilfe der beiden Attribute `CacheDefaultExpire` und `CacheMaxExpire` gelöst werden. Das Attribut `CacheDefaultExpire` gibt die Gültigkeitsdauer einer Datei an, für die keine eigene, normalerweise vom Server mitgeschickte Gültigkeitsdauer ermittelt werden konnte. Das Attribut `CacheMaxExpire` legt eine *maximale* Zeitangabe fest, in der eine im Cache gespeicherte Datei als aktuell betrachtet werden soll. Zusätzlich kann für die Mehrzahl von Proxies festgelegt werden, ob nur Dateien aus bestimmten Domänen gespeichert oder *ausdrücklich nicht* gespeichert werden sollen. In dem Attribut `CacheDomain` ist die Liste der Domänen gespeichert, deren Dateien als einzige in dem Cache gespeichert werden. Andere Dateien aus weiteren Domänen werden vom Proxy zwischen Client und Server nur vermittelt. Sind in dem Attribut `NotCacheDomain` Domänen als Werte eingetragen, so werden Dateien, die durch Anfragen nach Dateien aus diesen Bereichen vom Caching-Proxy übertragen werden, *auf keinen Fall* im Cache gespeichert.

5.2.4 Clientspezifische Management-Objektklassen

Um Clientspezifische Managementinformation zu gewinnen, wird die Klasse **WWW-Client** in das Objektmodell eingeführt. Diese Klasse leitet sich von der Oberklasse **Client** ab. Die Attribute `Request`, `Accept` und `Delay` der Klasse **Client** übernehmen für die neue Klasse die gleiche Bedeutung und definieren die Anzahl der in einem Intervall abgeschickten Anfragen (`Request`), die Anzahl der Anfragen, für die eine Antwort ankam (`Accept`), und die Zeitangabe, die zur Beantwortung der zum Abfragezeitpunkt zuletzt abgeschickten Anfrage benötigt wurde (`Delay`). Solche Angaben werden für die Überprüfung der Auslastung des Clients und der Güte der Anfragen oder des Netzes verwendet. Ist der Wert von `Delay` groß, so kann das entweder an den Übertragungszeiten im Netz liegen oder an der Geschwindigkeit des Servers, der die Anfrage bearbeiten muß. Zumindest das Netzwerk kann auf Mängel untersucht werden, um diese Verzögerungskomponente mit Sicherheit ausschließen zu können. Die von der Klasse **compObject** geerbten Attribute (`UsageState`, `OperationalState` und `AdminState`) zeigen den Nutzungsstatus, die Betriebsbereitschaft und den Administrationszustand des Clients an. Sämtliche Attribute wurden schon im Rahmen der generischen Objektklassen für verteilte Systemdienste erklärt (Abschnitt 5.1.2).

Um Informationen über die Konfigurationseinstellungen eines Clients zu erhalten, werden einige davon auf Attribute abgebildet. In dem Attribut `PreferencesFile` ist der Name und Verzeichnispfad der Datei gespeichert, in der die gesamten Konfigurationsanweisungen eingetragen sind, die für die jeweilige Instanz des Clients gelten. Der Name und Ort der Datei, die die Lesezeichen des Benutzers enthält, der den Client gestartet hat, ist in dem Attribut `BookmarksFile` gespeichert. In dem Attribut `MIMEFile` ist der Name der Datei enthalten, die die gesamten Paare "*MIME-Typ - Anwendung*" enthält und für die jeweilige Instanz des Clients gelten. In dem Attribut `MIMETypes` sind die Paare selber in Form einer Liste von

"*MIME-Typ - Anwendung*"-Paaren gespeichert. Neue Einträge können mit Hilfe der Funktion `addType(MIME,Application)` hinzugefügt werden.

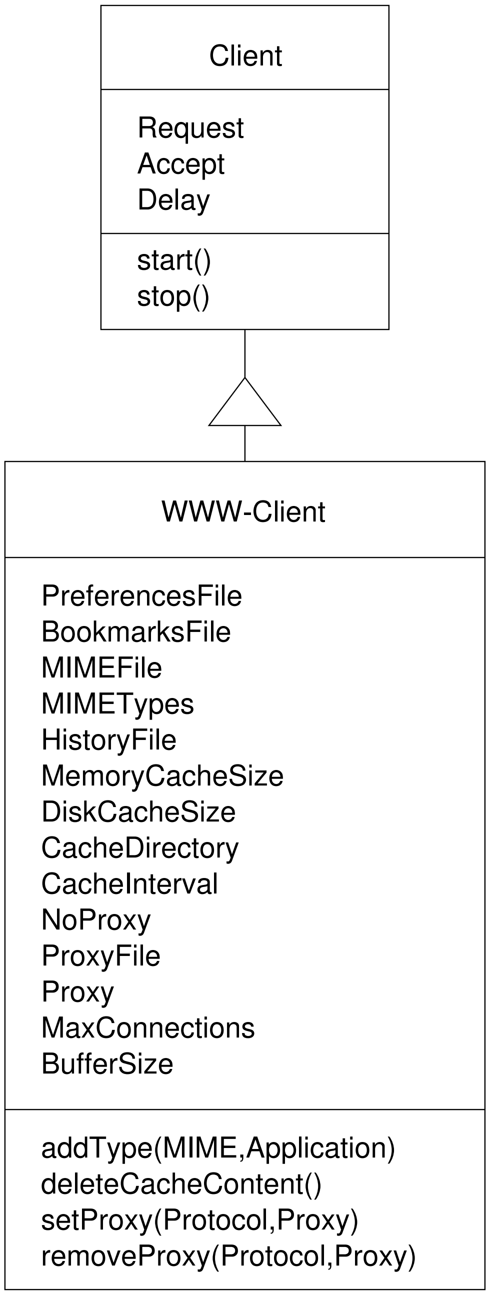

Abbildung 5.8: Objektklasse **Client**

In dem Attribut `HistoryFile` sind der Name und Ort der Datei eingetragen, wo die Adressen der vom Benutzer seit dem Start der Klassen-Instanz schon besuchten Seiten gespeichert sind. Dadurch wird die Navigation durch bereits besuchte Seiten erleichtert, da die Adressen nicht mehr neu eingegeben werden müssen. Durch das Cachen von bereits geladenen Dateien wird das Navigieren durch schon besuchte Dokumente erheblich beschleunigt. So wird nicht für jede

Seite eine erneute Anfrage an den Web-Server oder Proxy gestartet, sondern die Datei aus dem Browser-eigenen Cache geladen. Die meisten Browser unterstützen zwei Arten von Cache: den *Memory*-Cache und den *Disk*-Cache. Die Größe der beiden Caches, die frei eingestellt werden kann, ist in den Attributen `MemoryCacheSize` und `DiskCacheSize` zu finden. In dem Attribut `CacheDirectory` ist das Verzeichnis eingetragen, unter dem der Disk-Cache angelegt ist. Der Cache-Inhalt kann auf jedem Client lokal gelöscht werden, so daß mit Hilfe einer Funktion `deleteCacheContent()` das auch entfernt möglich sein sollte. Das Problem der Aktualität von Dokumenten bei Anwendung von Caches kann teilweise durch Angabe eines Zeitintervalls gelöst werden, nach dem das Dokument vom Server erneut angefordert werden soll. Dieses Zeitintervall ist im Attribut `CacheInterval` angegeben.

Soll ein Client die Anfragen über Proxies umleiten, so muß er dafür konfiguriert werden. Soll er *keinen* Proxy anwenden, so muß das Attribut `NoProxy` des Typs *Boolean* den Wert *true* haben. Ist das Attribut hingegen auf den Wert *false* gesetzt, so kann entweder mittels einer Datei die Konfiguration automatisch vorgenommen oder durch manuelle Einstellung einem Protokoll, mit dem die Anfrage gestellt wird, ein Proxy und dessen Port zugewiesen werden. Verwendet man eine Proxy-Konfigurationsdatei, so müssen deren Name und Verzeichnispfad im Attribut `ProxyFile` gespeichert sein; sonst enthält das Attribut `Proxy` eine Liste von Paaren der Form "*Protocol - Proxy - Port*". Mit Hilfe der Funktionen `setProxy(Protocol,Proxy)` und `removeProxy(Protocol,Proxy)` können einzelne Einträge in die `Proxy`-Liste eingefügt bzw. daraus gelöscht werden.

Normalerweise kann für Clients eine maximale Anzahl an Verbindungen angegeben werden, die gleichzeitig zwischen dem Client und mehreren oder einem Server geöffnet sein dürfen. Diesen Wert findet man in dem Attribut `MaxConnections`. Die Größe eines Puffers, in dem die empfangenen Daten aus einer offenen Verbindung zwischengespeichert werden, kann auch für die meisten Browser angegeben und soll in dem Attribut `BufferSize` gespeichert werden.

5.3 Zusammenfassung

Das im Rahmen der Diplomarbeit gewonnene Objektmodell wird den Anforderungen des integrierten Managements bezüglich verteilten Anwendungen, die WWW-Dienste realisieren, gerecht. Die mit Hilfe des *Computational Viewpoints* gewonnenen generischen Basisklassen definieren speziell Managementinformation, die für das Software-Management genutzt werden kann. Konzepte des *Engineering Viewpoints* liefern Objektklassen, die für das Management von Betriebssystemressourcen, wie Kommunikationsverbindungen oder Prozesse, genutzt werden. Die generischen Klassen für das Management des WWW-Dienstes, die durch die Verfeinerung der generischen Basisklassen hervorgehen, liefern die nötige Managementinformation, um das Management möglichst vieler Anwendungen im Umfeld der WWW-Dienste durchzuführen. Eine Gesamtübersicht der definierten, anwendungsspezifischen Objektklassen findet sich in Abbildung 5.9. Dabei ist anzumerken, daß spezielle, für konkrete WWW-Anwendungen geltende Merkmale im Objektmodell nicht vollständig berücksichtigt werden können. Dazu ist die Realisierung der einzelnen Anwendungen viel zu verschieden, abhängig vom Hersteller der Anwendung bis hin zu den Systemen, auf denen die Anwendungen laufen sollen.

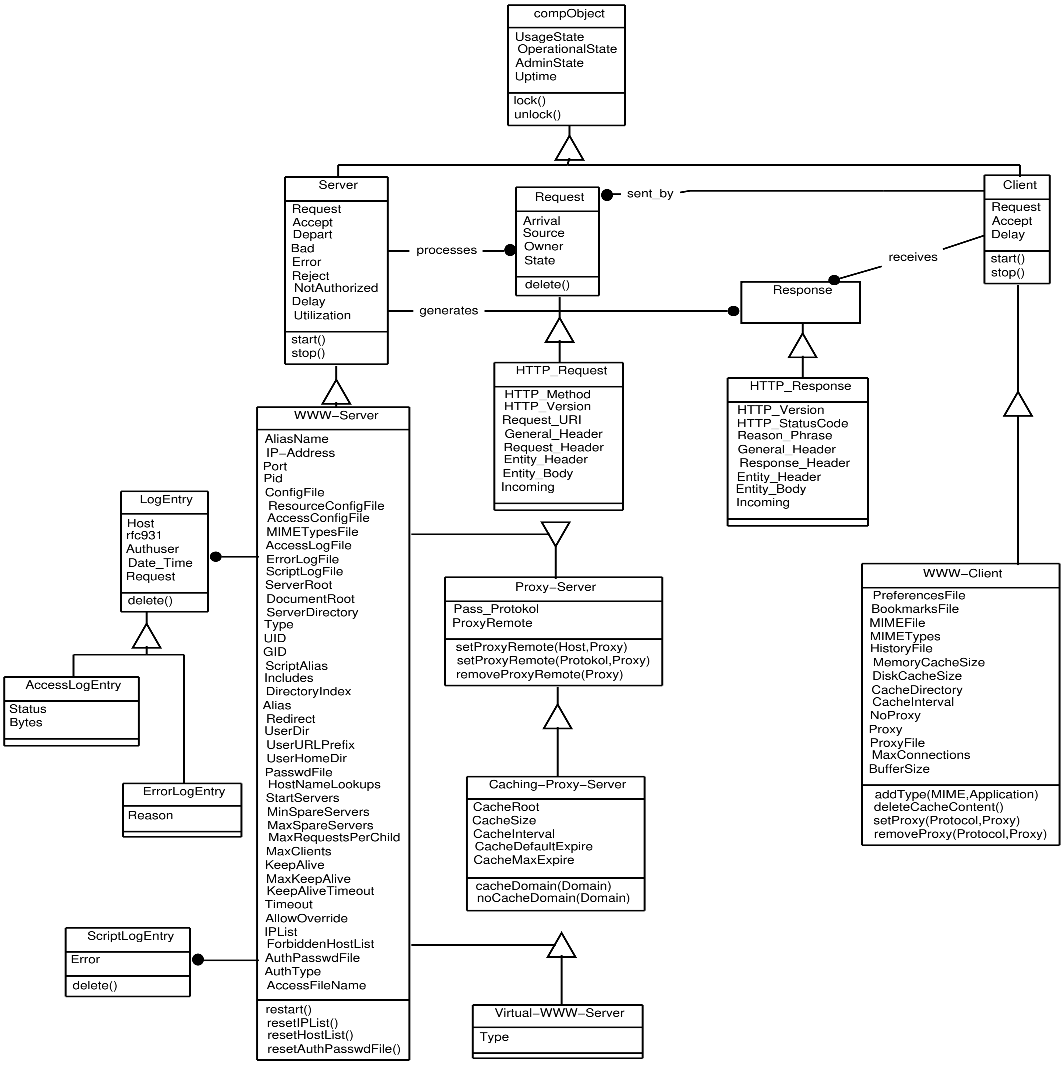

Abbildung 5.9: Anwendungsspezifische Objektklassen

Kapitel 6

Zusammenfassung und Ausblick

Das rapide Wachstum heutiger IT-Infrastrukturen hat zur Folge, daß vermehrt verteilte, offene Systeme eingesetzt werden, deren Hauptmerkmal in der Heterogenität der Komponenten besteht. In einer verteilten Umgebung finden immer mehr verteilte Anwendungen Gebrauch, die nach dem Client/Server-Prinzip agieren. Durch die Komplexität und Heterogenität der Systeme können die Aufgabengebiete des Netz-, System- und Anwendungsmanagements nicht mehr klar voneinander abgegrenzt werden. Wird eine Komponente einer verteilten Anwendung nicht mehr richtig ausgeführt, so kann es entweder an den Netzkomponenten liegen, über die die Kommunikation zwischen den einzelnen Anwendungskomponenten erfolgt, oder an einer Fehlfunktion des Endsystems, auf dem die Komponente gerade ausgeführt wird, oder auch an der Komponente selbst.

Aufgrund der Heterogenität der Systeme ist es sehr wichtig, von der verwendeten Managementarchitektur unabhängig zu sein. In verschiedenen Teilbereichen von Netzen können aufgrund von unterschiedlichen Anforderungen jeweils andere Architekturen eingesetzt werden. Das ideelle Ziel der Heterogenität bezüglich der eingesetzten Ressourcen besteht darin, die beste Lösung durch Kombination unterschiedlicher, je nach Anforderungen ausgewählter Hard- und Software-Komponenten zu erreichen.

Ein Managementmodell, das den Belangen des integrierten Managements gerecht wird, muß deshalb generisch genug sein, um möglichst viele Ressourcen darauf abbilden zu können und somit deren Heterogenität und Spezifika zu verschatten, andererseits muß es genügend anwendungsspezifische Informationen enthalten, um es auf konkrete Ressourcen oder Anwendungen abbilden zu können. Das im Rahmen der Diplomarbeit in Kapitel 5 entwickelte Objektmodell wurde speziell für das Management der verteilten Anwendung WWW entwickelt. Durch die steigende Beliebtheit des Internet und speziell des Informationsdienstes World Wide Web steigt der Bedarf an quallifizierten Managementanwendungen für verteilte Anwendungen, die WWW-Dienste realisieren.

Bestehende Lösungen für die Problematik wurden aufgegriffen und mit eigenen Methoden erweitert beziehungsweise umgesetzt. Anhand einer Top-Down-Analyse wurden mit Konzepten des ODP-Referenzmodells generische, systemunabhängige Objektklassen definiert, die für das Management von verteilten Anwendungen allgemein angewendet werden können. Das Referenzmodell ist in diesem Sinne für die Lösung dieser Aufgabenstellung hervorragend geeignet, da es ein mächtiges, standardisiertes Rahmenwerk für die Entwicklung und Beschreibung von verteilten Anwendungen zur Verfügung stellt. Der Entwicklungsprozeß von verteilten Anwendungen wird aus fünf verschiedenen, abstrakten Sichten (Viewpoints) betrachtet, so daß die Komplexität der Anwendung und der an sie gestellten Anforderungen aufgeteilt werden kann. Speziell Konzepte des Computational Viewpoints erlauben die funktionale Zerlegung einer verteilten Anwendung in Software-Komponenten, die über festgelegte Schnittstellen miteinander kommunizieren. Die damit definierten generischen Objektklassen decken Belange des Anwendungsmanagements ab. Mit Hilfe des Engineering Viewpoints werden die Infrastruktur-Komponenten festgelegt, die für die Kommunikation der verteilten Anwendungskomponenten notwendig sind. Die hiermit definierten Objektklassen erlauben ein Monitoring der Systemressourcen, wie z.B. der Prozesse oder Netzverbindungen.

Im nächsten Schritt wurden zu den generischen Attributen und Funktionen der Basisklassen neue, anwendungsspezifische Merkmale hinzugefügt und so das Objektmodell erweitert, um es auf konkrete Ressourcen in einem System abbilden zu können. Dabei wurde anhand einer Bottom-Up-Analyse untersucht, welche Art von Information von den Anwendungen selber geliefert wird und welcher Teil der Information als für alle Klassen dieser Ressource allgemeingültig betrachtet werden kann. Das entwickelte Objektmodell ermöglicht auch ohne Instrumentierung der WWW-Anwendungen eine relativ breite Abdeckung der Managementszenarien und erlaubt die Entwicklung von Managementanwendungen, die auf unterschiedlichen Ebenen ein effektives und integriertes Management der verteilten Anwendungen für WWW-Dienste durchführen.

Für die herstellerunabhängige Modellierung der Objektklassen wurde die Object Modeling Technique verwendet. Sie bietet einen mächtigen objektorientierten Ansatz, um die Klassen, deren Attribute und Funktionen sowie die Beziehungen zwischen den Klassen in einem Objektmodell darzustellen. Mit Hilfe des CASE-Tools Software through Pictures wurde anschließend das Objektmodell erstellt. Dieses Tool erlaubt u.a. das Entwerfen von Objektmodellen und die automatische Generierung von Schnittstellenbeschreibungen der Managementobjektklassen. Im Rahmen dieser Diplomarbeit wurde CORBA als Managementarchitektur untersucht. Als Grundlage für CORBA-Agenten werden IDL-Beschreibungen generiert. Dadurch kann u.a. die Unabhängigkeit der Objekte von der verwendeten Programmiersprache garantiert werden. Außerdem ist man nicht an bestimmte Kommunikationsprotokolle gebunden, da nicht nur Objekte von *allgemeinen* verteilten Anwendungen über den ORB miteinander kommunizieren können, sondern auch verteilte *Management*anwendungen. Weitere Vorteile von CORBA gegenüber anderen Architekturen wurden in Kapitel 3.4 erläutert.

Zuletzt wird eine aktuelle, in der Praxis angewendete Lösung auf dem Gebiet des Managements von WWW-Diensten beschrieben. Hierbei handelt es sich um die Einbindung einiger WWW-Server der BMW AG in eine zentrale Verwaltungsumgebung der Firma Tivoli.

6.1 Ausblick

Bei der BMW AG werden mehrere Web-Server eines Zuständigkeitsbereichs in eine zentrale Verwaltungsumgebung eingebunden, um so eine Fernüberwachung realisieren zu können. Bei den eingebundenen Servern handelt es sich ausschließlich um ein Produkt der Firma *Netscape*, den *Netscape Enterprise Server*. Aufgrund der homogenen Struktur der Rechner wird die gesamte Verwaltung der Server unter eine gemeinsame, zentrale Oberfläche gestellt, die sich auf Produkte der Firma *Tivoli* abstützt.

Das Produkt **TME 10** der Firma Tivoli ermöglicht die Überwachung und Administrierung von verteilten Systemen unter einer einheitlichen Oberfläche. Im Rahmen der Anwendung erlauben *Monitore* die Überwachung von Ressourcen, wie z.B. des Filesystems, von Prozessen oder von Anwendungen. Monitore werden automatisch in bestimmten Zeitabständen durchgeführt. Mit Hilfe von *Tasks*, die zu beliebigen Zeitpunkten vom Administrator ausgeführt werden können, ist das Auslösen von bestimmten Aktionen auf entfernten Rechnern möglich. Weiterhin kann mittels TME 10 die Softwareverteilung automatisch vollzogen sowie die Bestandsverwaltung des Systems oder auch die Benutzerverwaltung durchgeführt werden.

Für das Management der Netscape-Server wird das Modul *TME 10 Module for SuiteSpot* in das Framework von TME 10 integriert. Dieses Modul vollzieht das Management der eingebundenen Informationsserver eines Unternehmens, unabhängig von der Verteilung auf unterschiedliche Rechner. Das Modul legt den Schwerpunkt auf Management-Probleme rund um HTTP-, Directory-, Mail-, News- und Proxy-Server und Firewalls sowie die Verteilung von Intranet-Clients. Für diese Arbeit sind nur die Lösungen für die Verteilung von Clients sowie für das Management von HTTP- und Proxy-Servern von Bedeutung.

Um Web-Clients Intranet-weit auf Rechner, die der TME 10-Umgebung bekannt sind, verteilen zu können, muß das Quell- und Zielverzeichnis angegeben werden, von dem bzw. in das die selbstextrahierende Datei kopiert werden soll, die das Client-Softwarepaket enthält. Die Verzeichnisse bzw. Dateinamen werden systemabhängig angegeben. Die automatische Verteilung der Clients wird bisher nur für Windows-Systeme unterstützt.

Für jede Art von Server kann mit Hilfe von allgemeinen Monitoren (*Generic/All Managed Server Monitors*) der Status des Server-Prozesses überprüft werden (*up* oder *down*), ebenso die Größe des Server-Prozesses (Verbrauch an physikalischem RAM-Speicher in Kilobytes), die Größe aller Log-Dateien, der freie Speicherplatz des Filesystems, auf dem das Log-Verzeichnis angelegt ist und die Anzahl der aktiven Server-Prozesse. Der Zeitabstand, zwischen der jeweiligen Aysführung eines Monitors, kann beliebig gewählt werden und ist mit Defaultwerten vorbelegt. Allgemeine Tasks (*All Managed Servers Tasks*) ermöglichen ein aktives Eingreifen in einige Belange von Servern. Damit können sämtliche Log-Dateien archiviert, "rotiert[1]" oder gelöscht, Informationen über den aktuellen Status des Servers, über dessen Namen, Typ, Hostnamen und Log-Verzeichnis abgerufen sowie alle zu verwaltenden Server gestartet, gestoppt oder restartet werden. Einzelne Server können damit leider nicht angesprochen werden.

Spezielle Monitore für Web-Server (*HTTP Server Monitors*) liefern den von der Document Root verbrauchten und den noch freien Speicherplatz auf dem Filesystem, des weiteren Angaben über die Log-Dateien und die Anzahl der Einträge, die Menge der übertragenen Bytes und die Anzahl der Fehler aus der Log-Datei. Mit Hilfe von speziellen Tasks für Web-Server (*Web Server Tasks*) können neben den gleichen Tätigkeiten, die - nur speziell auf Web-Server angewandt - auch mit Hilfe der allgemeinen Server-Tasks durchgeführt werden, zusätzlich die Antwortzeiten auf verschiedenen Ports überwacht werden.

Speziell für das Management von Proxy-Servern können mit Hilfe von Monitoren (*Proxy Server Monitors*) u.a. die durchschnittliche und die maximale Antwortzeit für Anfragen abgefragt

[1] zyklisches Umbenennen von Dateien

werden sowie die durchschnittliche Rate der übertragenen Bytes, die Anzahl der Log-Einträge und die Häufigkeit, mit der bestimmte URLs aufgerufen wurden. Außerdem kann festgestellt werden, wieviel Speicherplatz schon belegt bzw. noch frei ist, welche Größe die Log-Dateien habenund wie oft bestimmte Status Codes auftreten. Besondere Tasks für Proxy-Server (*Proxy Server Tasks*) ermöglichen die gleichen Aktionen wie die Web-Server Tasks. Es gibt allerdings auch Tasks, die speziell für das Management von Proxies der Firma Netscape entwickelt wurden (*Netscape Proxy Server Tasks*). Damit kann ein Proxy zu einem Caching Proxy (oder umgekehrt) erweitert werden, sowie die Zugriffsversuche auf den Proxy beobachtet werden. Weiter können Einstellungen für den Cache vorgenommen und URL-Filter eingebaut werden.

Insgesamt bietet die Lösung von Tivoli gute Möglichkeiten, die gesamten Server und speziell die Web-Server und Proxies in einem Unternehmensnetz zu überwachen und zu verwalten. Im Gegegsatz zu den MIBs, die für das Management von WWW-Diensten entwickelt werden, hat man hier auch die Möglichkeit, aktiv auf die Komponenten einzuwirken. Allerdings sind die Möglichkeiten diesbezüglich noch sehr beschränkt und unausgereift, da mit dem Werkzeug z.B. nur *alle* Server angesprochen werden können, aber nicht jeder einzeln. Außerdem werden nur wenige Managementszenarien berücksichtigt, wodurch die angebotene Lösung für ein effektives Management von WWW-Diensten als noch unbefriedigend betrachtet werden kann.

Anhang A

HTTP 1.1 Status Code

200	OK
201	Created
202	Accepted
203	Partial Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Moved Temporarily
303	See Other
304	Not Modified
305	Use Proxy
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsopported Media Type
500	Internal Server Error
501	Not Implemented
502	Bad Gateway

503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Anhang B

CGI-Umgebungsvariablen

SERVER_SOFTWARE :
Der Name und die Versionsnummer der verwendeten Server-Software.
Beispiel: *Netscape Enterprise Server*

SERVER_NAME :
Der Hostname des Web-Servers, der die Anfrage für das CGI-Programm empfangen hat.
Beispiel: `www.myserver.top-level-domain`

GATEWAY_INTERFACE :
Die Versionsnummer des CGI, das der Web-Server implementiert.
Beispiel: `CGI/1.1`

SERVER_PROTOCOL :
Der Name und die Versionsnummer des bei der Anfrage verwendeten Protokolls.
Beispiel: `HTTP/1.1`

SERVER_PORT :
Die Portnummer, an die eine Client-Anfrage gesendet wurde.

REQUEST_METHOD :
Die HTTP-Methode, die in der Client-Anfrage verwendet wurde.
Beispiel: `GET`

PATH_INFO :
Enthält Extra-Pfadinformationen, die durch den Client angegeben wurden. Möchte ein Client das aufzurufende CGI-Programm mit Parametern oder zusätzlicher Information versorgen, kann er dies tun, indem er diese, durch einen Schrägstrich getrennt, an den URL des Programms hängt.
Beispiel: Wird ein CGI-Programm `/cgi-bin/script` mit dem URL

`http://www.myserver.tld/cgi-bin/script/weitere+informationen`

aufgerufen, so enthält die Variable PATH_INFO den Wert "`weitere+informationen`".

PATH_TRANSLATED :
Der Server legt den vollständigen Dateipfad des CGI-Programms in dieser Variable ab.

Beispiel: `/usr/local/etc/httpd/cgi-bin/script`

SCRIPT_NAME :
Der URL-Pfad des CGI-Programms, das aufgerufen wurde.
Beispiel: `/cgi-bin/script`

QUERY_STRING :
Die Parameter, die an ein CGI-Programm übergeben wurden, in der Form, wie sie vom Web-Client übermittelt wurden. Diese Variable wird bei Anfragen mittels der HTTP-Methode GET gesetzt.

REMOTE_HOST :
Der Hostname des Client- beziehungsweise des Proxy-Rechners. Der Wert ist nur gesetzt, falls der zugreifende Rechner einen DNS-Eintrag besitzt.
Beispiel: `www.myclient.tld`

REMOTE_ADDR :
Die IP-Adresse des Client- bzw. Proxy-Rechners.

AUTH_TYPE :
Unterstützt der Server Benutzer-Authentifizierung, oder handelt es sich beim aufgerufenen CGI-Programm um ein geschütztes Dokument, enthält diese Variable die Art des verwendeten Authentifikationsverfahrens.
Beispiel: `Basic`

REMOTE_USER :
Die User-ID, die bei der Authentifikation verwendet wurde.

REMOTE_IDENT :
Das Ergebnis einer IDENT-Anfrage an den Client-Host.

CONTENT_TYPE :
Enthält eine Anfrage, die mittels der HTTP-Methoden PUT oder POST gestellt wurde, weitere Daten, enthält diese Variable den zugehörigen MIME-Typ der der Daten. Bei einer GET-Anfrage wird diese Variable nicht gesetzt.

CONTENT_LENGTH :
Die Länge der Daten in Bytes, die mittels einer PUT- oder POST-Methode an den Server geschickt wurden und die dieser dem CGI-Programm über die Standardeingabe übergibt. Bei einer GET-Anfrage wird auch diese Variable nicht gesetzt.

Zusätzlich zu den oben aufgezählten Umgebungsvariablen teilt ein Web-Server, der zu CGI/1.1 kompatibel ist, dem CGI-Programm sämtliche HTTP-Header mit, die bei einer Anfrage vom web-Client an den Server übermittelt wurden. Dabei entsprechen die Namen der Variablen den Namen der Header, nur daß für diese ausschließlich Großbuchstaben verwendet werden und ihnen ein `HTTP_` vorangestellt wird. Zusätzlich werden alle Spiegelstriche "-" durch Unterstriche "_" ersetzt.

HTTP_ACCEPT :
Eine Auflistung der MIME-Typen, die der Client verarbeiten kann.

HTTP_ACCEPT_CHARSET :
Eine Auflistung der Zeichensätze, die der Client verarbeiten kann.

HTTP_ACCEPT_ENCODING :
Die Kodierungsarten, die der Client verarbeiten kann.

HTTP_ACCEPT_LANGUAGE :
Eine Auflistung der Sprachen, die der Client bzw. dessen Benutzer, verarbeiten kann.

HTTP_AUTHORIZATION :
Die Daten einer WWW-Authentifikation.

HTTP_CACHE_CONTROL :
Informationen, ob und wie die Datei gespeichert werden kann bzw. ob von einem Cache die gespeicherte Datei zurückgeliefert werden kann.

HTTP_COOKIE :
Das vom Client übermittelte Cookie, sofern eins vorhanden ist.

HTTP_FROM :
Die E-Mail Adresse des Clients bzw. dessen Benutzers, die nur in seltenen Fällen von einem Client übermittelt wird.

HTTP_HOST :
Der Name des Web-Servers, an dem der Client die Anfrage geschickt hat.

HTTP_REFERER :
Der URL der Seite, von der aus der Link verfolgt wurde.

HTTP_USER_AGENT :
Der Name und die Versionsnummer der Web-Client-Software.

HTTP_VIA :
Informationen zu den Proxy-Servern, über die die Anfrage gestellt wurde.

Anhang C

OMT-Objektmodelle

Die folgenden Seiten enthalten Teile des kompletten Modells für das Management von verteilten Anwendungen, das in dieser Arbeit entwickelt wurde. Die Seiten im DIN A3 Format beinhalten das von Tobias Müller ([Mue98]) übernommene Objektmodell.

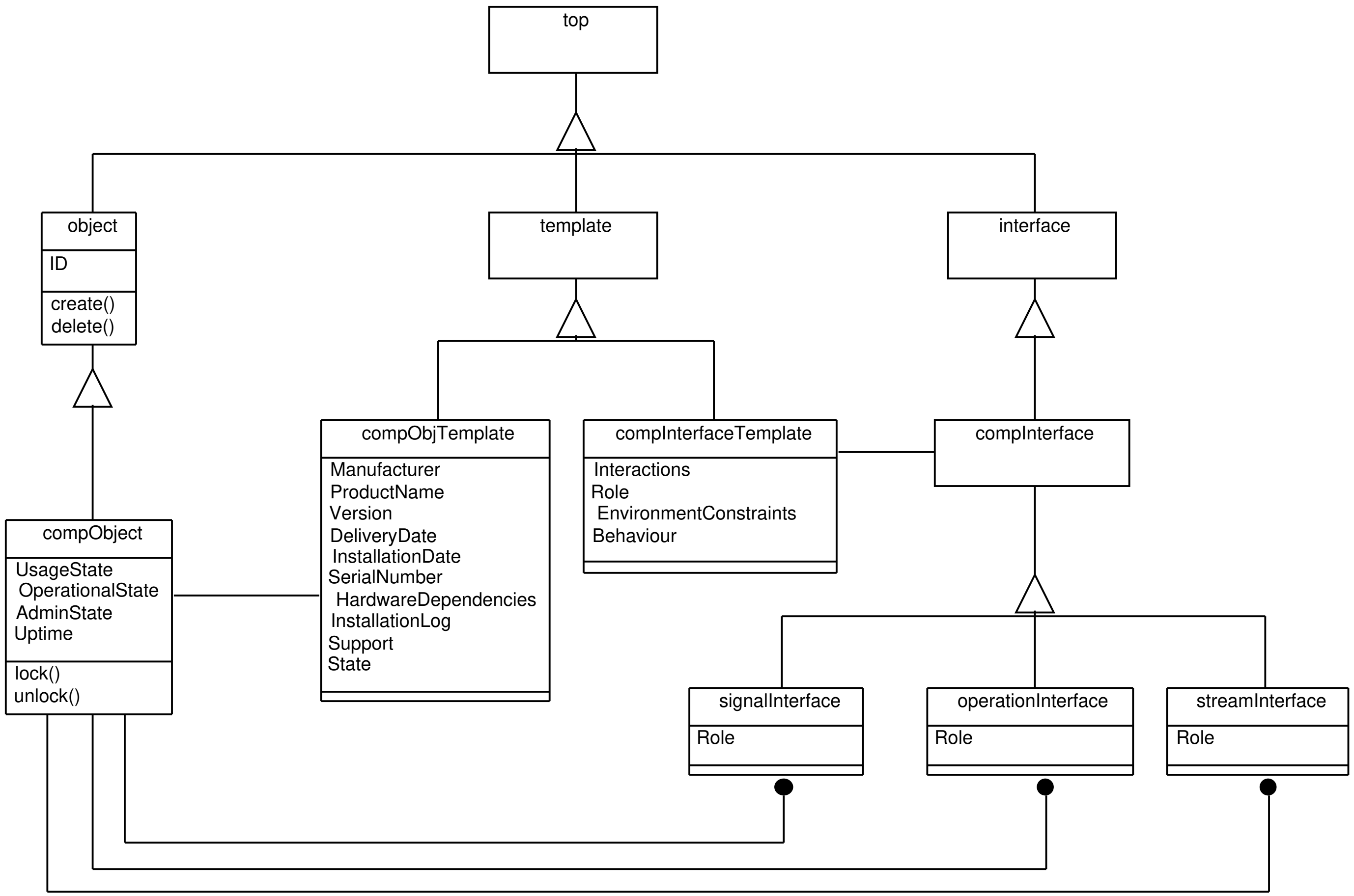

Abbildung C.1: Generische MOCs zum Computational Viewpoint

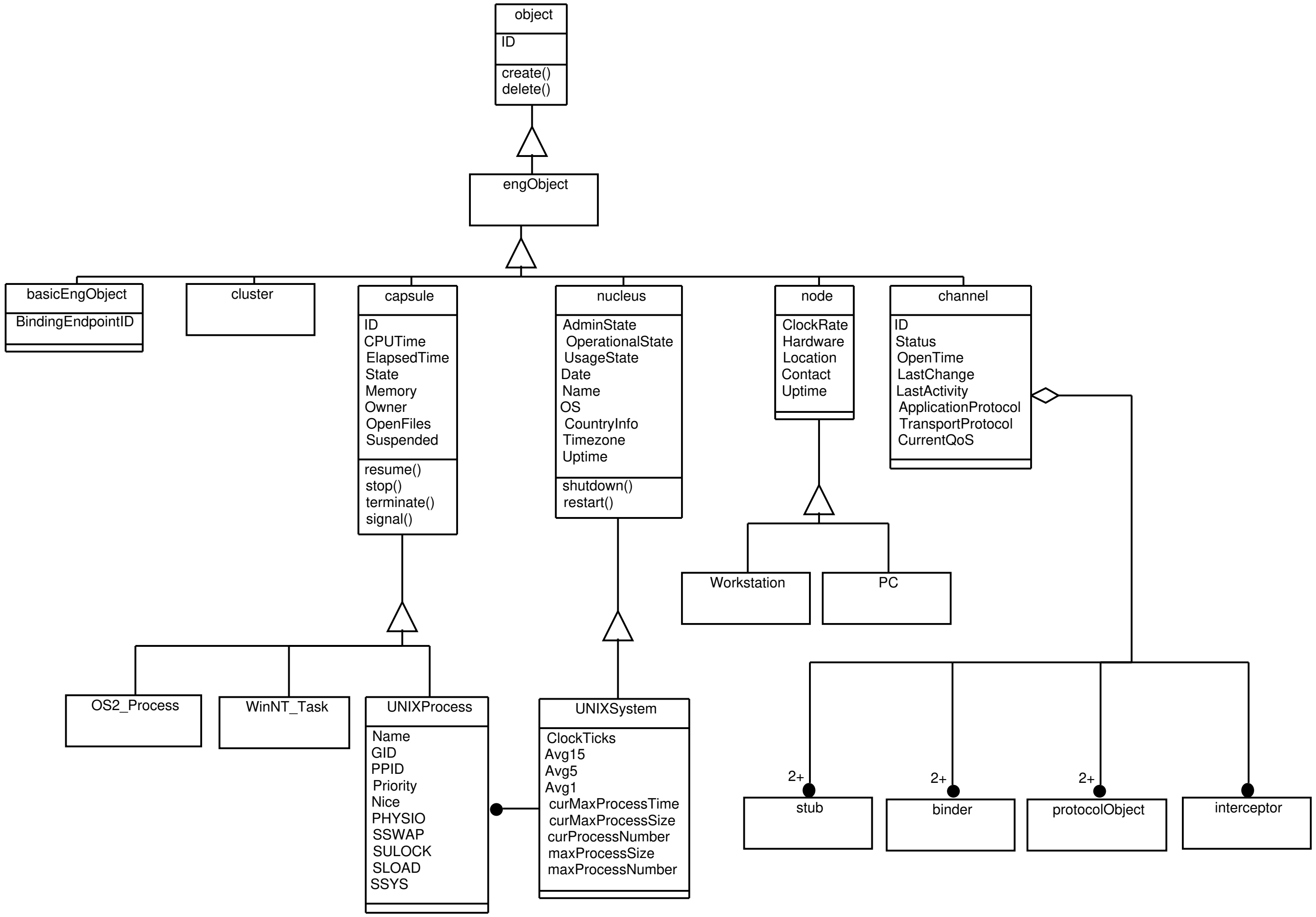

Abbildung C.2: Generische MOCs zum Engineering Viewpoint

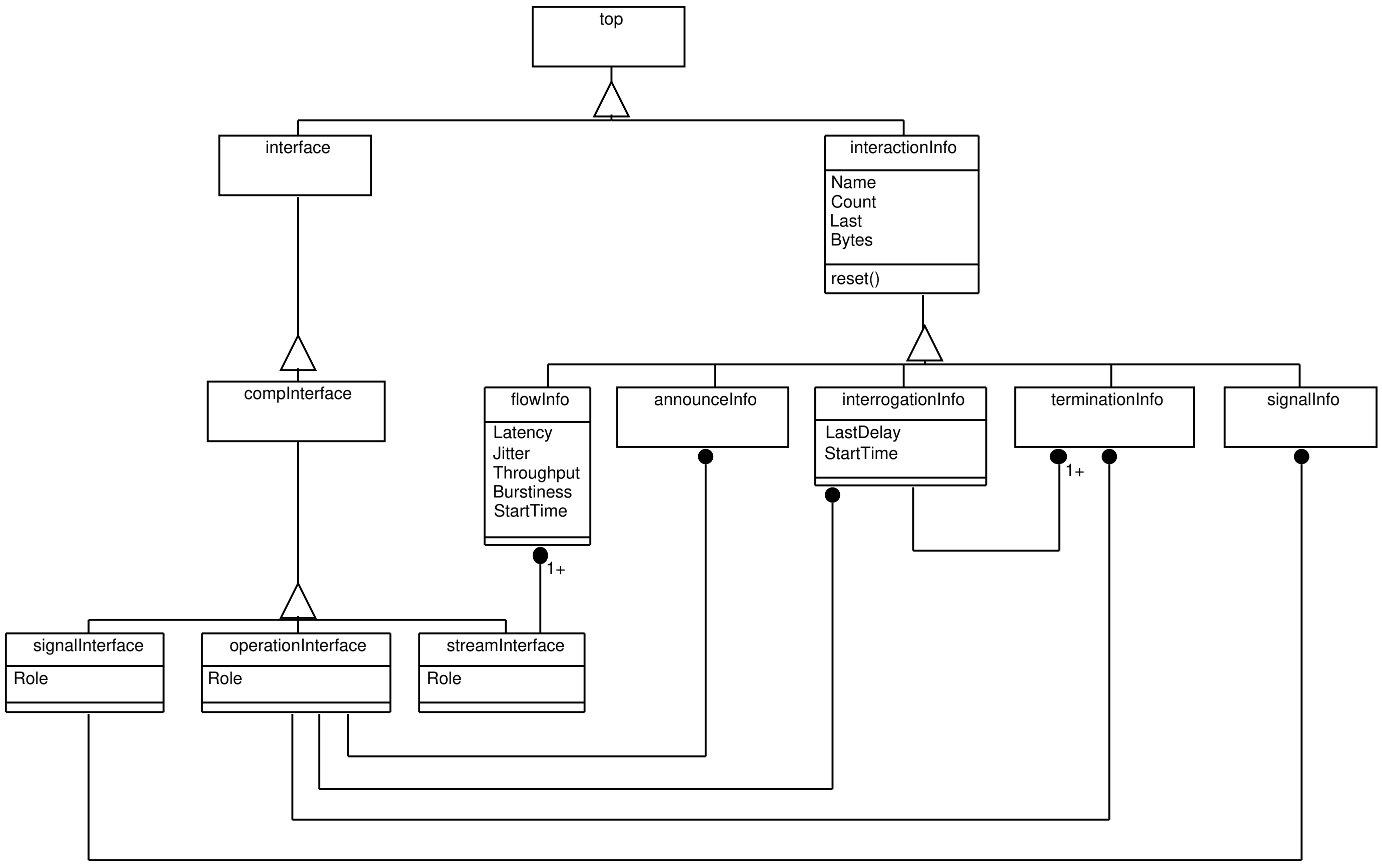

Abbildung C.3: Generische MOCs zu den Schnittstellen

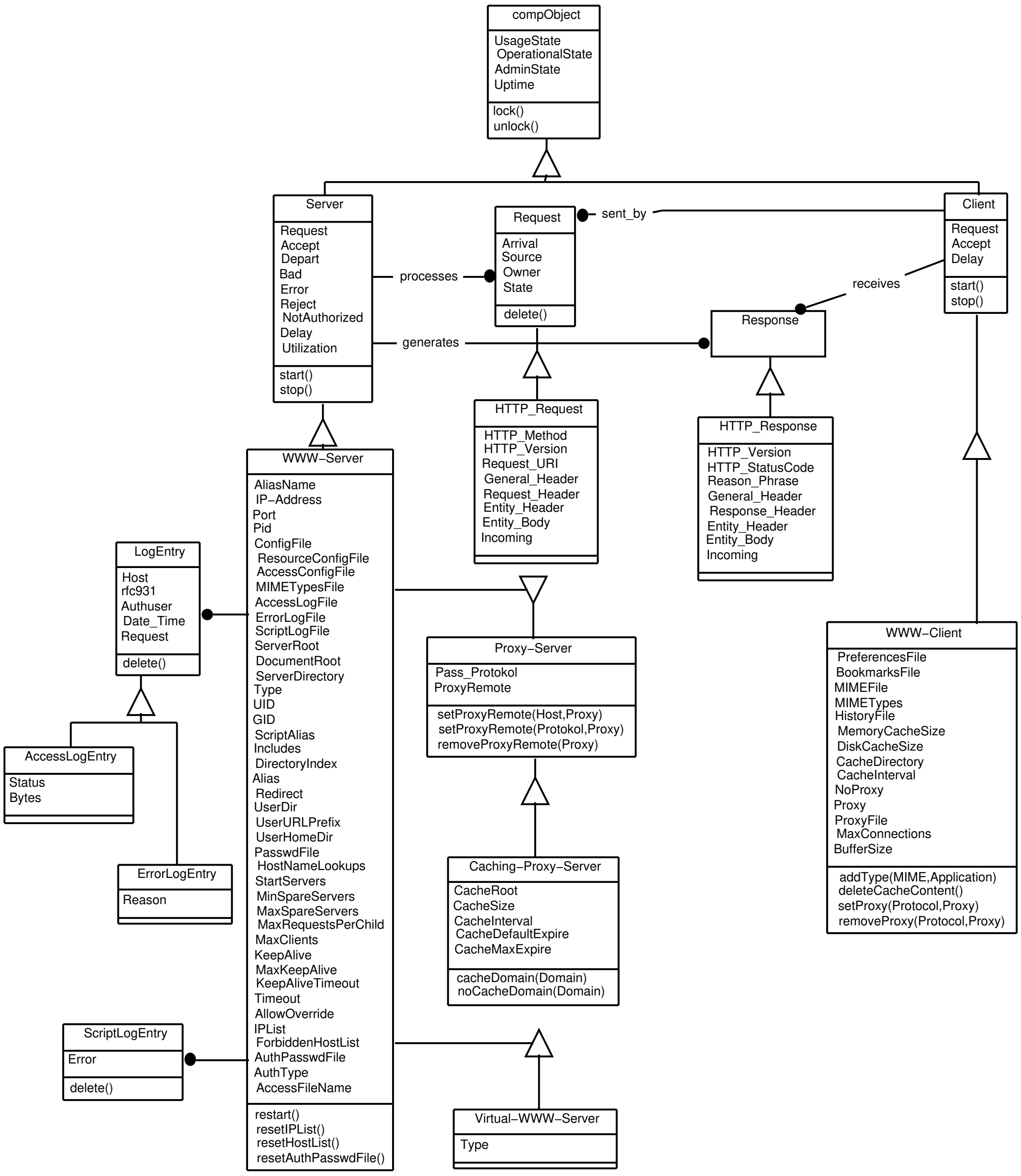

Abbildung C.4: Spezielle MOCs zu WWW-Anwendungen

Abkürzungsverzeichnis

AMS	Applications Management Specification
API	Application Programming Interface
ARM	Application Response Measurment
CASE	Computer Aided Software Engineering
CO	Computational Object
CORBA	Common Object Request Broker Architecture
DNS	Domain Name Service
DMTF	Desktop Management Task Force
FTP	File Transfer Protocol
GAMOC	Generic Application Managed Object Class
GDMO	Guidelines for the Definition of Managed Objects (OSI)
HTTP	HyperText Transfer Protocol
IAB	Internet Activities Board
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
LAN	Local Area Network
MIB	Management Information Base
MIF	Management Information Format
MO	Managed Object
MOC	Management Object Class
NSM	Network Services Monitoring
ODMA	Open Distributed Management Architecture
ODP	Open Distributed Processing
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
OSF	Open Software Foundation
QoS	Quality of Service
QRL	Query and Reporting Language (StP)
RM-ODP	Reference Model of Open Distributed Processing
SMA	Systems Management Architecture
SNMP	Simple Network Management Protocol
StP	Software through Pictures
SW	Software
TCP	Transport Control Protocol

TCP/IP	Transport Control Protocol / Internet Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
WWW	World Wide Web

Literaturverzeichnis

[AK97] Alexander Keller, Bernhard Neumair: *Using ODP as a Framework for CORBA-based Distributed Applications Management.* Technischer Bericht Ludwig-Maximilians-Universität München, Institut für Informatik, 1997.

[Ban96] Ban, Bela: *Open Distributed Processing: A Reference Model For Distributed Computing.* Tutorial, IBM Research Laboratory, 1996. http://www.zurich.ibm.com/~bban.

[DMT95] Desktop Management Task Force, Software Working Committee: *Software Standard Groups Definition, Version 2.0*, November 1995.

[Eil97] Eilebrecht, Lars: *Apache Web-Server.* International Thomson Publishing, Bonn, Erste Auflage, 1997.

[ES96] Euba, A. und D. Stricker: *Implementierung eines Werkzeuges zur graphischen Darstellung von WWW-Link-Strukturen.* Fortgeschrittenenpraktikum, Technische Universität München, Juli 1996.

[Far97] Farooqui, Kazi: *Reference Model Of Open Distributed Processing (ODP) – A Guided Tour.* Tutorial, Department Of Computer Science, University of Ottawa, 1997. ICODP'97.

[HA93] Hegering, Heinz-Gerd und Sebastian Abeck: *Integriertes Netz- und Systemmanagement.* Addison-Wesley, Bonn, Erste Auflage, 1993.

[HNW95] Hegering, H.-G., B. Neumair und R. Wies: *Integriertes Management verteilter Systeme – Ein Überblick über den State-of-the-Art –.* Bericht 9503, Ludwig-Maximilians-Universität München, Institut für Informatik, Januar 1995.

[Int96a] Interactive Development Environments, Inc.: *Software through Pictures Tutorial: Getting Started with StP/OMT, Release 3*, 1996.

[Int96b] Interactive Development Environments, Inc.: *Software through Pictures User Manual: StP Core – Fundamentals of StP, Release 2*, 1996. http://www.aonix.com/Products/StP/stp.html.

[Int96c] Interactive Development Environments, Inc.: *Software through Pictures User Manual: Object Modeling Technique – Generating Code, Release 3*, 1996.

[Int96d] Interactive Development Environments, Inc.: *Software through Pictures User Manual: Creating OMT Models, Release 3*, 1996.

[ISO95a] ISO/IEC JTC1/SC21/WG7: *10746-1 Open Distributed Processing Reference Model Part 1: Overview*, 1995.

[ISO95b] ISO/IEC JTC1/SC21/WG7: *10746-2 Open Distributed Processing Reference Model Part 2: Foundations*, 1995.

[ISO95c] ISO/IEC JTC1/SC21/WG7: *10746-3 Open Distributed Processing Reference Model Part 3: Architecture*, 1995.

[ISO95d] ISO/IEC JTC1/SC21/WG7: *Recommendation X.904: Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semantic Amendment*, 1995.

[Joy] Joyner, Ian: *Open Distributed Processing: Unplugged!* http://www.dstc.edu.au/AU/research_news/odp.

[MBLF+97] Mogul, J, Tim Berners-Lee, R. Fielding, H. Nielsen und J. Gettys: *Hypertext Transfer Protocol – HTTP/1.1.* Internet Draft, IAB, 10 1997. gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-http-v11-spec-rev-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-http-v11-spec-rev-00.txt.

[Mue98] Mueller, Tobias: *CORBA-basiertes Management von UNIX-Workstations mit Hilfe von ODP-Konzepten.* Diplomarbeit, Technische Universität München, Februar 1998.

[NET96a] Netscape Communications Corporation, USA: *Netscape Enterprise Server*, 1996. Programmer's Guide for UNIX.

[NET96b] Netscape Communications Corporation, USA: *Netscape Enterprise Server*, 1996. Administrator's Guide for UNIX.

[Neu97] Neumair, Bernhard: *Ein Managementmodell für verteilte Systemdienste und Anwendungen.* Internes Papier, Ludwig-Maximilians-Universität München, Institut für Informatik, 1997.

[Obj95] Object Management Group: *CORBA 2.0 Specification*, 1995. http://www.omg.org.

[RBP+91] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy und William Lorensen: *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, Erste Auflage, 1991.

[Sch96] Schütz, F.: *Implementierung eines Werkzeuges zur Konsistenzprüfung von HTML-Links.* Fortgeschrittenenpraktikum, Technische Universität München, März 1996.

[Sie96] Siegel, Jon (Herausgeber): *CORBA Fundamentals and Programming.* John Wiley & Sons, New York, Erste Auflage, 1996.

[Ste95] Stein, Lincoln D.: *How to Set Up and Maintain a World Wide Web Site.* Addison-Wesley Publishing Company, Massachusetts, Erste Auflage, 1995.

[Vin97] VINOSKI, STEVE: *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments.* Paper, IONA Technologies, Inc., 1997. vinoski@iona.com.

[Vog97] VOGEL, ANDREAS: *Java, CORBA and the Web.* Tutorial, IFIP WG6.1, 1997. icodp/icdp '97.

[Wim96] WIMMER, PETER: *Implementierung eines Analysewerkzeuges für Logdateien von WWW-Servern.* Fortgeschrittenenpraktikum, Technische Universität München, März 1996.